Software Engineering

Minority Game

Report 3

Group 12

Academic Year 2014-2015

Rutgers University

May 5, 2015

Members: Matthew Chatten, Ameer Fiqri Barahim, Vicent Vindel Dura, Alexander Hill, David Lazaar, Orielle Joy Yu.

Table of Contents

1. Summary of Changes

1. Traceability matrix explanation added for the Functional Requirements section.
2. Added Causal Description of UC-5
3. Added Causal Description of UC-9
4. Interaction Diagrams revised
5. Updated Use Cases to reflect unimplemented section and improved clarity.
6. Updated System Requirements descriptions
7. Updated Customer Statement of Requirements
8. Rewrote Glossary of Terms
9. Rewrote description of user interface, added new mockup
10. Added discussion of changes and implementations of requirements
11. Rewrote section on persistent data storage
12. Updated hardware requirements
13. Fixed minor typos and errors across whole report
14. Updated user interface with new interface pictures
15. Rewrote algorithms to reflect final program structure
16. Updated references
17. Did full effort estimation for final project
18. Redid functional requirements specification, including new use case diagrams
19. System Sequence Diagrams updated
20. Added history of work
21. Changed details in design of tests, added more tests, changed descriptions

2. Customer Statement of Requirements

2.1 Problem Statement

The goal of this project is to better understand the El Farol Bar problem(1) through the analysis of a Minority Game(2). The El Farol Bar problem has many variants, but the core idea remains the same: how to make a decision that is dependent on the decisions of others. In the original formulation, the question is whether or not to go to a bar. Going to the bar is a good decision only if most people decide it is a bad decision, and vice versa. There is no perfect answer to such a question. The Minority Game is an expansion of the problem in which a set of agents must use and test strategies for solving the El Farol Bar problem. If you choose the minority option, you win the round. Strategies are tested with the knowledge of how previous rounds turned out. By running a Minority Game, you can understand how independent agents will choose to answer the El Farol Bar problem, even if a “right” answer is impossible.

This is the real value of Minority Game. The El Farol Bar problem is game theory, and the game helps translate that into practical information. In the real world, you will always have imperfect information on other people’s decisions. In any case where there is competition for a limited resource, or where being in the minority is advantageous, accurately predicting other people’s actions and making the least popular choice is a valuable skill. This is a question with many real life applications beyond the original idea of trying to avoid a bar if it is too crowded. Examples include building a company in an underserved market, choosing a career in a field that needs more workers, finding a vacation spot that isn’t overcrowded, or picking a stock that’s undervalued. A good understanding of how independent actors act in a Minority Game can thus be used to better understand how to act most effectively in real life situations.

With this in mind, our desired product is a Minority Game simulation designed to model agents in a competitive fishing environment. In order to gain the maximum value from such a simulation, it must be customizable, complex, and able to deliver large amounts of relevant data. There is no better way to do this than through software. Only software can make customization easy, hide the complexity of the simulation, and deliver useful and readable data with ease.

The most basic Minority Game involves only a set of agents and a single bar, but real life situations are much more complex. This product should expand on the Minority Game in two main directions. First, there should be multiple fishing locations that the agents can choose from. Second, there should be multiple types of fish, each with their own fishing locations, that the agents can choose from. Ideally, an agent wants to choose both the least crowded fishing spot and the least popular fish species.

The goal of this Fishing Minority Game is to understand how agents should act in such a real life scenario. To facilitate this understanding, the software should have three key features. First, it should have a range of options that will make many unique simulation configurations possible. Choosing different numbers of agents or decreasing the amount of fish, for example, will provide more opportunities for insight than a single static scenario. Second, the simulation should show how the individual agents act over time and what strategies they use so a clear pattern of action and response can be discerned. Third and finally, the simulation should collect all relevant data and present it once complete for analysis.

With the basic framework of the Fishing Minority Game established, it is important that the details be explained as well. Fishing does not only depend on the number of people going fishing, it is also affected by the skills and experience of the fisherman. Factors such as weather, types of fish, and fish population also affect outcomes. The simulator should take into account all of these factors.

Different initial variables can be set for the simulation: number of fishers, number of locations, number of types of fish, fish population, runtime, and weather patterns. Each of these will affect how likely each fisher is to fish on a specific day at a specific location. For example, poor weather will make fishers more likely to stay home. Increased number of locations means fishers will be less crowded and more likely to go fishing. After all these inputs, the simulation will run. The simulation can be “watched” rather than running and displaying the results automatically. All fishing locations and fishers will be represented on screen, and their choices day to day on where to fish will be shown as well throughout the duration of the simulation. When the simulation is done, the simulator should show graphs that can be easily understood by the user and thus, help the user to ease up the decision making. Graphs can display information on each location, showing its occupation percentage, and also information on the most successful fishers.

Of course, the key to all of this is the strategies that the fishers will use to choose fishing locations and types of fish to catch. These strategies should be based on the information available to the fishers: how other actors have acted in the past, and the current conditions. In real life, that is the type of information available, and so it should be here. To make this simulation feasible, however, the strategies will be fairly straightforward. An example strategy would be going to a certain fishing spot if the current conditions are known to be good and it has been overpopulated more than once previously. Another could be choosing a fishing spot if it was overpopulated two rounds ago and underpopulated last round. Strategy development, implementation, and evaluation should follow these steps:

1. (Initialization) Agents randomly choose a set of strategies and a specific fish species to begin.
2. Each agent looks at their strategies and chooses the one with the highest success score, i.e. the one that would have made a good decision the highest number of times.

3. Each agent receives the outcome of the round and updates its strategies based on the new data.
4. Each agent looks at its strategies’ success as a whole and decides whether or not to switch fish species.
5. Each agent returns to step 2, unless the simulation has ended.

With this sort of strategy implementation, the simulation will be sure to cover the two most important aspects of this Minority Game variation. First, it will allow for the testing and evaluation of strategies in direct competition with each other. The point of this product is to understand how different agents act, and the key to that is seeing which strategies work, which don’t, and how strategy effectiveness changes over time. By having every strategy for every actor constantly evaluated, this goal can be achieved. Second, it will allow for a better understanding of the costs and benefits of switching fish species entirely. An agent who switches has a chance to be more successful, but will be taking a large risk since their strategy success scores will be tuned to a different fish species. This simulation should allow better insight into the tradeoffs involved in such a risky decision.

By providing all of this information and all of these options for simulation, the Fishing Minority Game can serve two classes of users. First, it can serve the individual user who is interested in seeing what kind of strategies they might use in a minority game setting such as fishing. By observing which fishers are successful, and by looking at the overall trends in the simulations, they can get an idea of when or when not to go fishing themselves. Second, it can serve organizational users who are interested in the big picture of minority game problems. For example, it could serve a park who controls a number of lakes used for public fishing. Using this simulator, they can see how park patrons are likely to act over time. They can then use this information to help optimize the use of the park’s resources.

The final challenge in building this Minority Game fishing simulation is making it into a customer-friendly product. The simulation won’t give any help understanding the El Farol Bar problem if it is difficult to use, or if the data is hard to read. With that in mind, there are a few basic expectations for this software project. There should be an easy to use graphic user interface for the simulation. This will allow for easy understanding and setting of simulation options, simple representation of the simulation itself, and an interface for the collection and presentation of simulation results. Second, the software should be self-contained. It should be a downloadable product that can be installed and run on its own. It only needs to be used by a single person at a time, and since the simulation only depends on initial parameters it shouldn’t need any outside connections or products to function.

The construction of a Fishing Minority Game will not be an easy task. The problem itself, a question of game theory and complex interactions between independent agents, is only a model of the even more intricate machinations of real life. However, with this simulation we hope that we can come a little closer to understanding.

3. Glossary of Terms

1. **Poor weather** - A set of simulation settings that introduce bad weather, affecting the decisions of fishers.
2. **Overcast** - A poor weather setting, meaning that a storm is likely. Fishers more likely to stay home.
3. **Strategy** - An algorithm for choosing where and when to fish, used by fishers and constantly evaluated for effectiveness.
4. **Simulation settings** – Allows the user to change variables for the simulation.
5. **Fisher** - A single simulation agent, running strategies trying to catch as many fish as possible amongst competition with other fishers.
6. **Simulation** - This is the software's run through of the user-selected scenario, in which fishers each attempt to fish as many fish as possible given the constraints.
7. **El Farol Bar Problem** - A game theory problem involving the decision whether or not to go to a bar based on expectation of crowdedness.
8. **Minority Game** - A version of the El Farol Bar Problem where agents attempt to improve their performance at solving the problem.
9. **Fishing Minority Game** - An expanded version of the minority game featuring fishers attempting to maximize their fish haul and competing against each other.
10. **Home** - The default location for a fisher. Staying home means no fish caught.
11. **Report** - The Fishing Minority Game's option to summarize what happened in a given simulation. Allows for the creation of graphs explaining key outcomes.
12. **Location** - A spot where fishers can fish. Each location is separate from the others, and contains its own set of fish.
13. **Types of fish** - There are a variable number of fish species that can be caught. Changing the types of fish changes the number of species. Only one species can be caught at a time by a particular fisher.
14. **Fish population** - This is the number of fish available as a whole. Increasing or decreasing this affects how likely the fishers are to catch fish.
15. **Runtime** - Fishers go out to fish once per day. Changing the runtime changes the number of days. After each day, fishers will update their strategies based on the results of that day.

4. System Requirements

4.1 Enumerated Functional Requirements

Identifier	Priority Weight (Low1-High5)	Requirement Description
REQ-1	4	The simulation shall incorporate initial values for number of fishers, number of fishing locations, number of fish, number of types of fish, and runtime.
REQ-2	2	The simulation will account for weather conditions such as rain and snow.
REQ-3	2	The simulation will account for the amount of experience each fisher, increasing their success over time based on individual results.
REQ-4	1	The simulation will increase fishermen experience when they fish in unpopular areas or for unpopular types.
REQ-5	4	The simulation will track each fisher's decisions over time.
REQ-6	2	The simulation will account for reduction in fish due to overfishing by decreasing the amount of fishermen the area supports.
REQ-7	5	The simulation results will allow the user to generate a report with data from the simulation.
REQ-8	4	The simulation will have the fishers update their strategy based on the results of each day.
REQ-9	3	The simulation will provide different strategy options for fishermen to use when deciding when and where to fish.

Discussion:

The basic structure of these 9 requirements has remained intact throughout the project, though their exact nature has changed. For example, the simulation options changed in REQ-1 the functionality of report generation changed for REQ-7, etc. Notably, REQ-4, the lowest priority requirement, was not implemented.

4.2 Enumerated Nonfunctional Requirements

The FURPS model stands for functionality, usability, reliability, performance, and supportability. Each of these would be elaborated below for this particular software.

· Functionality - This software allows users to set parameters for different factors. User inputs parameters and the output would be graphs showing results of software simulation.

· Usability – The software should be easy to learn and navigate. The users would only need to input parameters and/or choose factors. User interface design should be well planned, organized and consistent. There are documentations and demos of how to use the program.

· Reliability – The probability of system failures should be low. Errors that are caused by user inputs should be addressed clearly in order to have it corrected immediately. Saved files should have backup copies for recoveries in case of saving/loading failure. The resulting data of simulations should be accurate.

· Performance – The software should have high but efficient performance. Since the data simulation would be running on real time, the speed should be slow enough to for the user to follow but fast enough so that the user would not have a long waiting period. Power, time, and space consumption should be minimal.

· Supportability – The software should be easy to understand by the user and the programmers. There would be clear documentations of all the product's required files. This would help product maintenance and repair. Technical support by email should also be available.

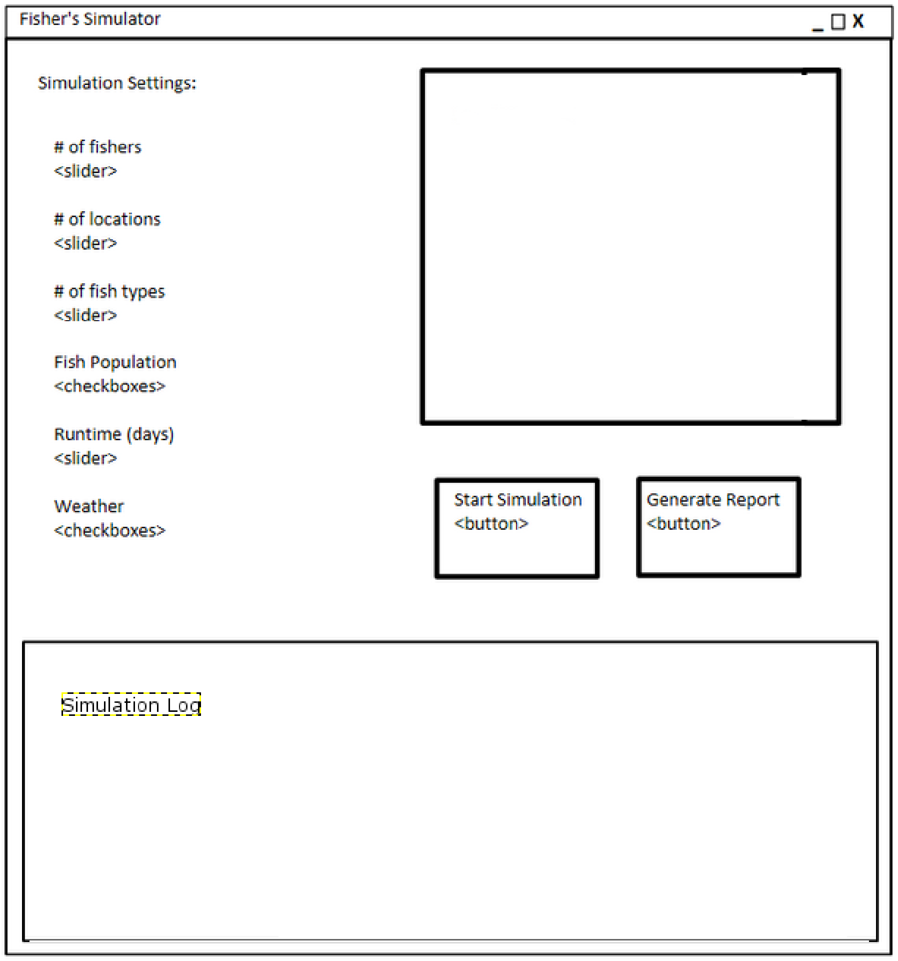
Identifier	Priority Weight (Low1-High5)	Requirement Description
REQ-10	5	Can choose which inputs to be included and not included in simulations.
REQ-11	1	Be able to create an account with username and email address.
REQ-12	4	The user interface should be easy to navigate and easy to learn.
REQ-13	3	Data results should be able to be saved externally for safety and distribution.
REQ-14	3	If an error is caused, tell user what it is and avoid crashing the program.
REQ-15	2	Software should be easy to install and run.
REQ-16	2	Be able to send email to creators for technical support.

Note: REQ-11 and REQ-16, the two lowest priority requirements, were not implemented. Others, such as REQ-13, were updated.

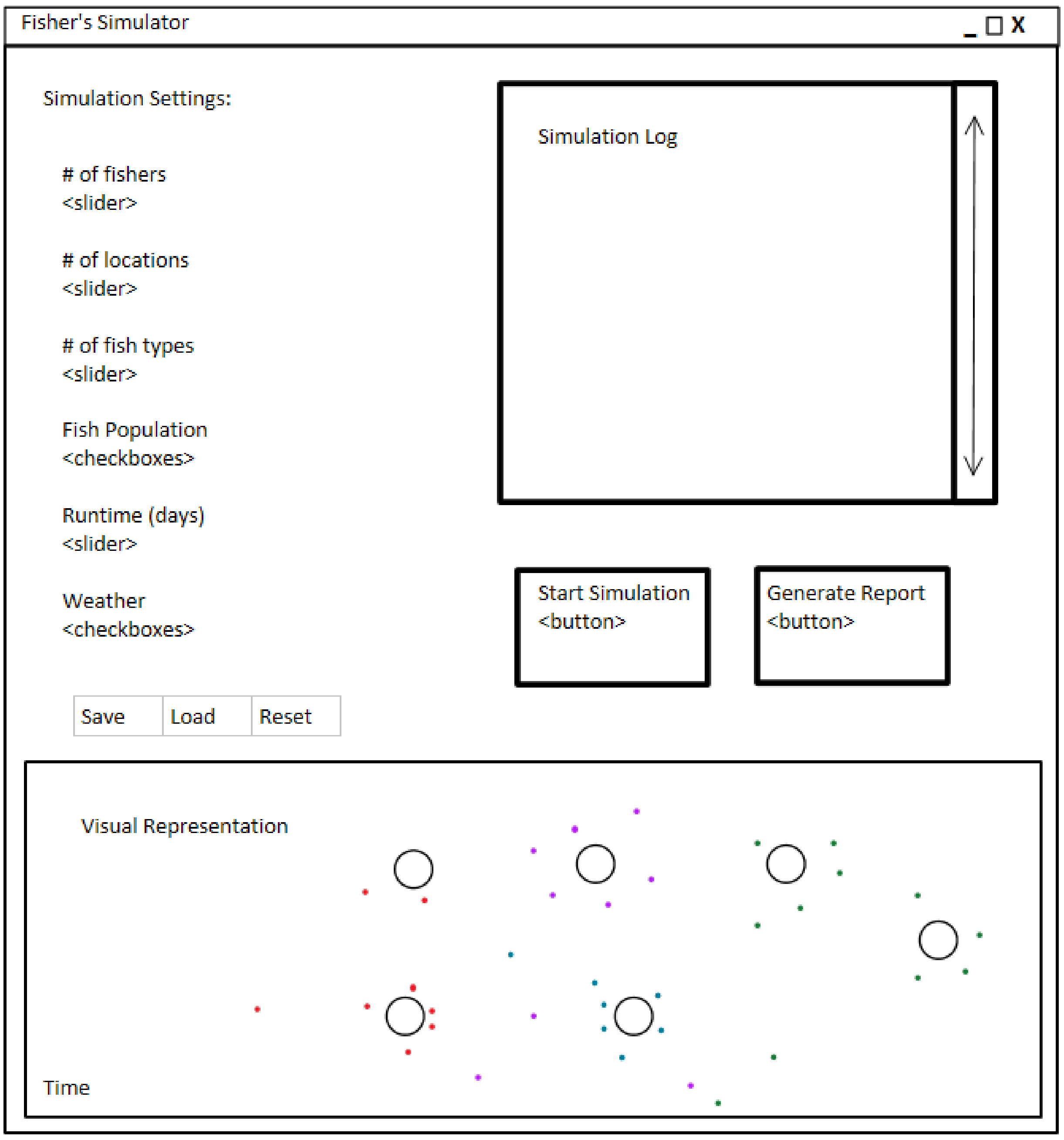
4.3 Onscreen Appearance Requirements

The Fisher's Simulator application has a simple design. Simulation settings are found on the upper left side of the window. They include options like the amount of fishers, number of fishing locations, number of fish types, fish population, simulation runtime, and weather. Checkboxes or sliders are used where appropriate. This method of data input provides a simpler, more guided user experience compared to fillable forms, which are too easily filled with invalid data by new users. The simulation visualization will dominate the right side of the application. It will show in realtime where all the fishers are, either at home or at one of the fishing locations. Below this will be large start and generate report buttons. The start button allows the user to start the simulation. It may turn into a pause button while the simulation is ongoing. The generate report button will allow the user to choose graphs generated from the simulation data and add them with text to a report, that then can be saved. On the bottom of the screen, there will be a text-based simulation report that keeps track of what's going on.

New Main Screen Mockup:



Original Main Screen Mockup:



5. Functional Requirements Specification

5.1 Stakeholders

This software is designed for fishermen who are looking to find the location that is least crowded and to maximize their chance of success of catching a certain species of fish. A fisherman would want to fish at the least crowded fishing spot and learn the least popular fish species in order to have a higher chance of enjoyment and success. Results are in graph form in order to make it easier to read and understand. The software can also be used to keep track of experience, strategies and changing factors in order to influence the fisherman's decision in a positive way.

5.2 Actors and Goals

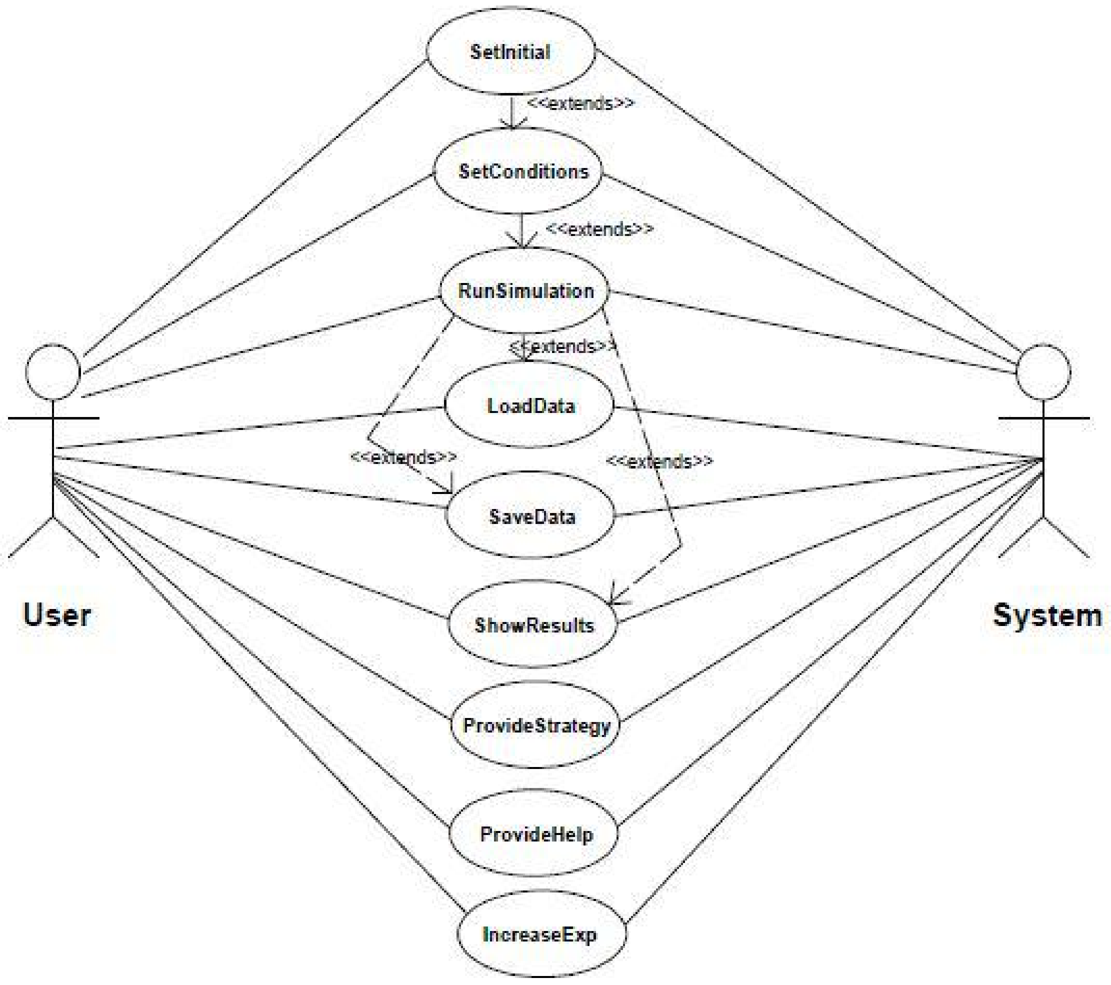
Actor	Actor's Goal
User	Set initial values of conditions
User	Choose conditions to include in simulation
User /System	Run simulation
User	Save results of simulation
User	Load results of simulation
System	Show graphs of simulation
System	Provide strategies to user
System	Increase experience points
System/User	Provide technical help

5.3 Use Cases

5.3.1 Casual Description

Identifier	Use Case Name	Casual Description	Related Requirements
UC-1	SetInitial	The user can set initial for the simulation.	REQ-1,REQ-2, REQ-3, REQ-6, REQ-10
UC-2	SetConditions	(Combined with UC-1)	
UC-3	RunSimulation	The user would tell the system to start the simulation. The system would respond by running the simulation that the user can watch.	REQ-7
UC-4	SaveData	The user can save current simulation data. Saved data will be backed up.	REQ-5,REQ-8,REQ-13
UC-5	LoadData	The user can load previous data. (not implemented)	REQ-8
UC-6	ShowResults	The system would show the finished graphs of the simulation.	REQ-7
UC-7	ProvideStrategy	The system can provide different strategy options for user.	REQ-9
UC-8	IncreaseExp	The system would take account current and past data to increase experience points of user when necessary.	REQ-4
UC-9	ProvideHelp	When there an error the system can provide help or user can email for help.	REQ-14,REQ-16

5.3.2 Use Case Diagram



5.3.3 Traceability Matrix

Req	**PW**	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
REQ-1	4	X								
REQ-2	1		X							
REQ-3	1		X							
REQ-4	2								X	
REQ-5	4				X					
REQ-6	2		X							
REQ-7	5			X			X			
REQ-8	4				X	X				
REQ-9	3							X		
REQ-10	5		X							
REQ-11	1									
REQ-12	4	X	X				X			
REQ-13	3				X					
REQ-14	3									X
REQ-15	2									
REQ-16	2									X
Max PW		4	5	5	4	4	5	3	2	3
Total PW		8	13	5	11	4	9	3	2	5

The Traceability Matrix above indicates how each of the use cases relate to the requirements originally proposed. From the above matrix it is clear that the most critical use cases are UC-2, UC-4, UC6, and UC1. These all deal with the users ability to setup the simulator and then receive graphs and other visualizations. Some of the less critical use cases include the ability to load previous sessions and the consideration of a fishers past experience in the simulation.

5.3.4 Fully Dressed Description

Use Case UC-2: SetConditions
Related Requirements: Req-2, Req-3, Req-6, Req-10
Initiating Actor: User
Actor’s Goal: Set up conditions for a simulation.
Participating Actors: System
Preconditions: None
Postconditions: Simulation is ready to run
Flow of Events for Main Success Scenario: 1. User opens menu for selecting conditions. 2. User selects options on menu. 3. User finishes selecting options. 4. System checks that all conditions are valid. 5. System enables user to run simulation.
Flow of Events for Extension: 2a. User does not finish selecting options and exits menu. 4a. Parameters are not all valid. 4b. System alerts user to which conditions are invalid.

Use Case UC-1: SetInitial
Related Requirements: Req-1
Initiating Actor: User
Actor’s Goal: Set values for conditions in simulation
Participating Actors: System
Preconditions: None
Postconditions: Simulation is ready to run
Flow of Events for Main Success Scenario: 1. User opens menu for selecting condition values.. 2. User selects values on menu. 3. User finishes selecting options. 4. System checks that all conditions are valid. 5. System enables user to run simulation.
Flow of Events for Extension: 2a. User does not finish selecting options and exits menu. 4a. Parameters are not all valid. 4b. System alerts user to which conditions are invalid.

Use Case UC-6: ShowResults
Related Requirements: Req-7
Initiating Actor: User
Actor’s Goal: Get results from simulation
Participating Actors: System
Preconditions: Parameters are all chosen
Postconditions: Simulation returns data.
Flow of Events for Main Success Scenario: 1. User clicks “Start simulation.” 2. Simulation runs on system. 3. System displays output in readable format.

Flow of Events for Extension: 2a. Simulation fails, error is reported.

Use Case UC-4: SaveData
Related Requirements: Req-5, Req-8, Req-13
Initiating Actor: User
Actor's Goal: Save data from simulation
Participating Actors: System
Preconditions: A simulation has successfully returned data
Postconditions: Data is saved
Flow of Events for Main Success Scenario: 1. User chooses save option. 2. User selects file name to save data into. 3. System saves data.
Flow of Events for Extension: 1a. No data is available because simulation has not been run, return error. 3a. System cannot save data, return error.

Note for Casual Descriptions:

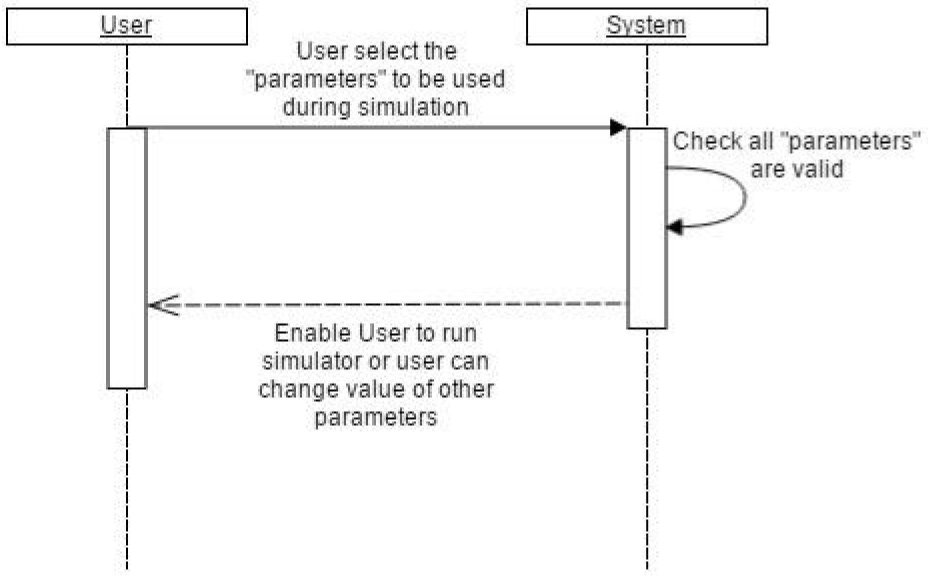
The following Use cases were not implemented and could be added at some future time.

UC-5: This use covered Requirement 8 and would have been initiated by the user. When initiated, this would allow the user to locate a previously saved session and load those settings into the simulator. The result of running this command would be a simulator that is in the same state as when the save was performed.

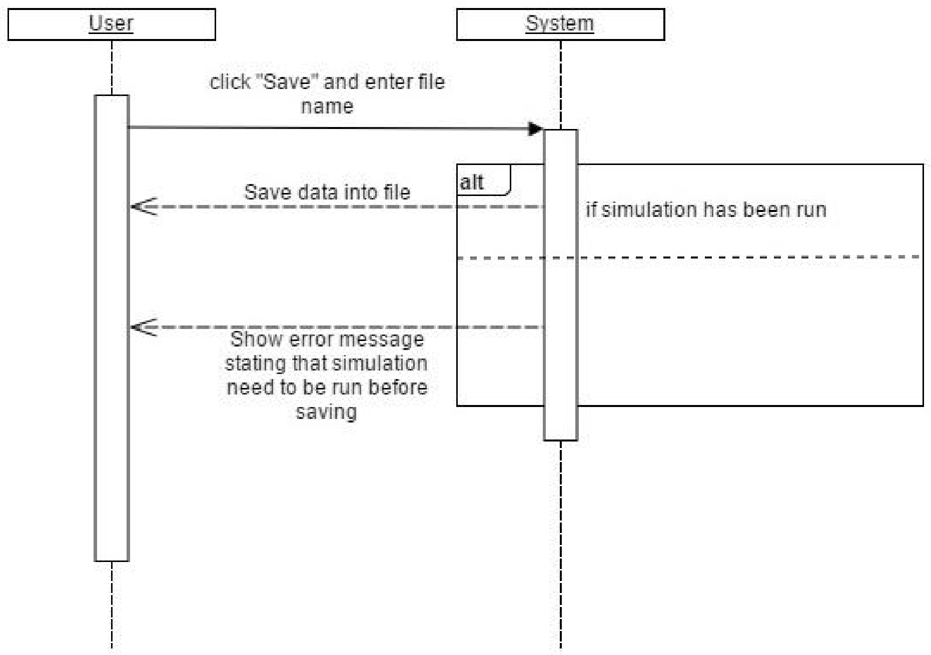
UC-9: This use case was to include a robust system to notify the users of system errors and attempt to provide ways of correcting those errors.

5.4 System Sequence Diagram

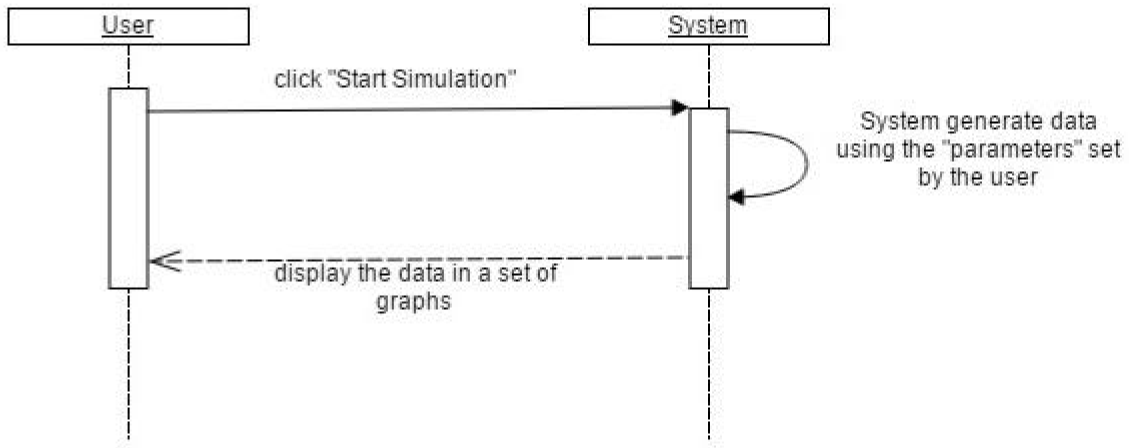
The system sequence will show the interaction between the system and the user. This said to be the boundary between the user and the system. By providing the system sequence diagram, every possible input from the user side will need to be predicted so that the system can manage the input. The most important is for the system to manage the invalid input so that the system will not crash. The more detailed descriptions of how system manage all the input will be provided in the design sequence diagram. Below are some of the system sequence diagram for the high priority use cases.


User
System
User select the "parameters" to be used during simulation
Check all "parameters" are valid
Enable User to run simulator or user can change value of other parameters

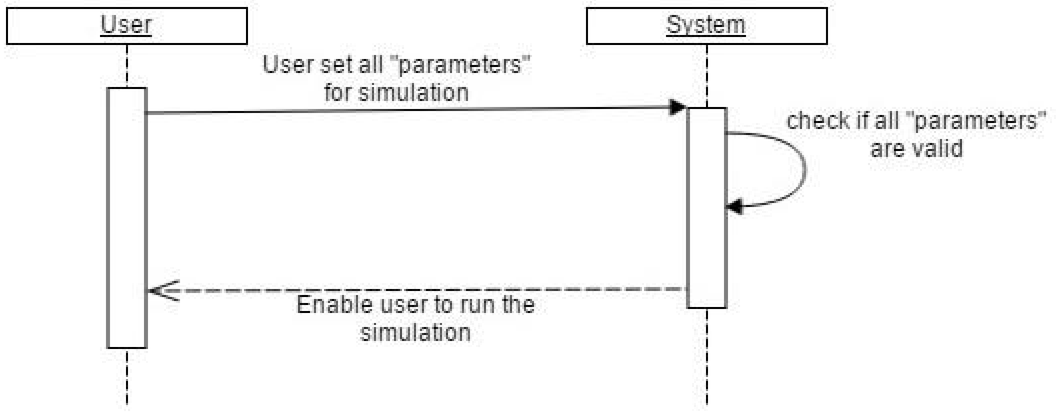
Use Case 2 (UC-2)


User
System
click "Save" and enter file name
alt
Save data into file
if simulation has been run
Show error message stating that simulation need to be run before saving

Use Case 4 (UC-4)


User
System
click "Start Simulation"
System generate data using the "parameters" set by the user
display the data in a set of graphs

Use Case 6 (UC-6)


User
System
User set all "parameters" for simulation
check if all "parameters" are valid
Enable user to run the simulation

Use Case 1 (UC-1)

6. Effort Estimation using Use Case Points

Unadjusted Use Case Points (UUCP) Calculation:

Unadjusted Actor Weight (UAW):
Used Table 4-1:Actor classification and associated weights in the *Software Engineering* textbook for complexity and weights below.

Actor Name	Description	Complexity	Weight
User	Interacts with system through GUI	Complex	3
System	Complex internal calculations and shows results through GUI	Complex	3
		Total	6

Unadjusted Use Case Weight (UUCW):
Used Table 4-3: Use case weights based on the number of transactions in the *Software Engineering* textbook for category and weights below.

Use Case	Description	Category	Weight
UC-1 (SetInitial)	Simple user interface, 2 participating actors (user and system), 5 steps for successful scenario	Average	10
UC-2 (SetConditions)	(combined with UC-1 above)	X	X
UC-3 (RunSimulation)	Complex processing, 2 participating actor (actor and system), 7 steps for successful scenario	Complex	15
UC-4 (SaveData)	Simple user interface, 2 participating actors, 4 steps to successful scenario	Simple	5
UC-5 (LoadData)	(not implemented)	X	X
UC-6 (ShowResults)	Simple user interface, 2 participation actors, 4 steps to successful scenario	Simple	5
UC-7 (ProvideStrategy)	Average processing, 2 participating actors, 3 steps to successful scenario	Average	10
UC-8 (IncreaseExp)	Complex processing, 1 actor	Average	10

	participating (system), 3 steps to successful scenario		
UC-9 (ProvideHelp)	(not implemented)	X	X
		Total	55

UUCP = UAW + UUCW
= 6 + 55
= 61

Technical Complexity Factor (TCF) Calculation:

Used Table 4-5: Technical complexity factors and their weights in the *Software Engineering* textbook for calculated factor below.

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor (Weight *Perceived Complexity)
T1	Only runs on one machine, not distributed.	2	0	0*0 = 0
T2	Users expect good performance, fast simulated results	1	4	1*4 = 4
T3	End-user expects high efficiency demand	1	4	1*4 = 4
T4	Complex interval processing, many calculations performed	1	5	1*5 = 5
T5	Reusable design for other specific purposes (this project is for fishing)	1	2	1*2 = 2
T6	Easy to install	0.5	2	0.5*2 = 1
T7	Easy to use, for users	0.5	2	0.5*2 = 1
T8	Portability not required	2	0	2*0 = 0

T9	Easy to change for improvements or other necessary changes	1	3	1*3 = 3
T10	Concurrent use not required	1	0	1*0 = 0
T11	No special security features	1	0	1*0 = 0
T12	No necessary direct access to third parties although a third party can use final results.	1	1	1*1 =1
T13	No special user training required (documentation available and easy to understand GUI presentation)	1	1	1*1 = 1
			Technical Factor Total	22

Constant 1 = 0.6 as defined in the book
Constant 2 = 0.01 as defined in the book
TCF = Constant 1 + Constant 2 *(Technical Factor Total)
= 0.6 + 0.01*22
= 0.82

Environmental Complexity Factor (ECF) Calculation:

Used Table 4-7: Environmental complexity factors and their weights in the *Software Engineering* textbook for calculated factors below.

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor (Weight *Perceived Impact)
E1	Beginner familiarity with development process	1.5	1	1.5*1 = 1.5
E2	Some application problem experience	0.5	2	0.5*2 = 1

E3	Some experience in objective-oriented approach	1	2	1*2 = 2
E4	Beginner analyst capability	0.5	1	0.5*1 = 0.5
E5	Good level of motivation	1	4	1*4 = 4
E6	Stable requirements	2	3	2*3 = 6
E7	No part time members	-1	0	-1*0 = 0
E8	Moderate difficulty programming language (C++)	-1	4	-1*4 = -4
			Environmental Factor Total	11

Constant-1 (*C*1) = 1.4
Constant-2 (*C*2) = 0.03
ECF = Constant-1 Constant-2 *(Environmental Factor Total)
= 1.4 + 0.03*11
= 1.73

Use Case Points (UCP) = UUCP * TCF * ECF

= 61*0.82*1.73

= 86.53

Duration = UCP x Producaticity Factor (PF)

= 86.53*28

= 2,422.84 person-hours

(PF = 28 hours per use case point assumed.)

7. Domain Analysis

7.1 Domain Model

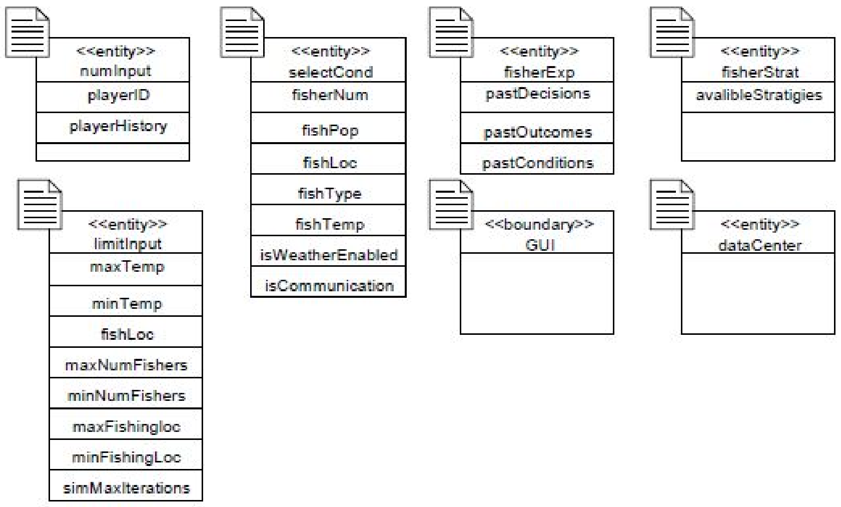
7.1.1 Concept Definitions

According to *Software Engineering* by Ivan Marsic, there are can be two types of responsibilities. There are type D and type K responsibilities which are about “doing” and “knowing”. A type D or “doing” responsibility requires making an action to get the specified results like data calculation. Meanwhile, type K or “knowing” responsibility requires to only remember a specific data like a data container. Sometimes it would be difficult to distinguish between the two types and the label would be blank.These are used to label all concept definitions below.

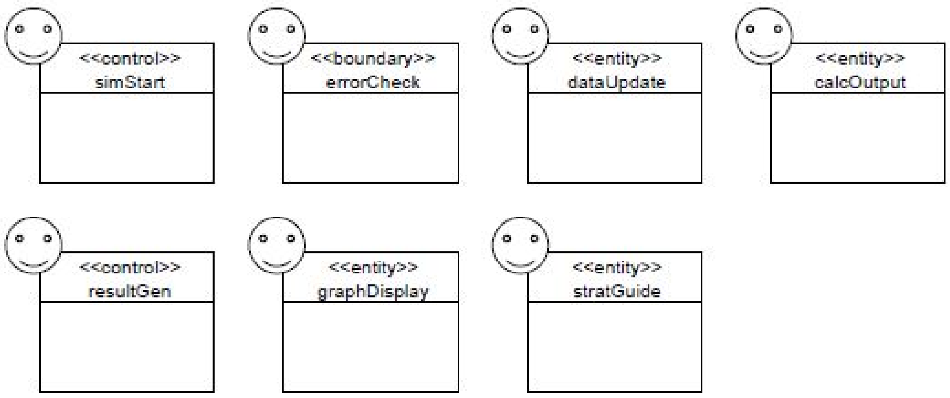
Responsibility Description	Type	Concept Name
The container that remembers all user’s inputs	K	numInput
The container of all selected conditions/factors	K	selectCond
Contains the number of fisherman	K	fisherNum
Contains the population of fish	K	fishPop
Contains the number of fishing locations	K	fishLoc
Contains the user’s experience	K	fisherExp
Contains the user’s strategies	K	fisherStrat
Contains the types of fish	K	fishType
Contains the weather temperature	K	fishTemp
Contains the limits of all input parameters	K	limitInput
Starts the simulation program	D	simStart
Checks if user inputs are valid and sends error message	D	errorCheck
Contains all the user’s saved results, experience and strategy.	K	dataCenter

Updates the user’s experience, strategies and selected conditions after simulations in the data center.	D	dataUpdate
Calculates output results using user inputs using algorithm/mathematical model.	D	calcOutput
Generates the results of the simulation in real time simulation.	D	resultGen
Shows the final graphs of the simulation.	D	graphDisplay
Give out strategies according to user data	D	stratGuide
Provides overall display of the software program for the users.	K	GUI

Type K diagram:



Type D diagram:

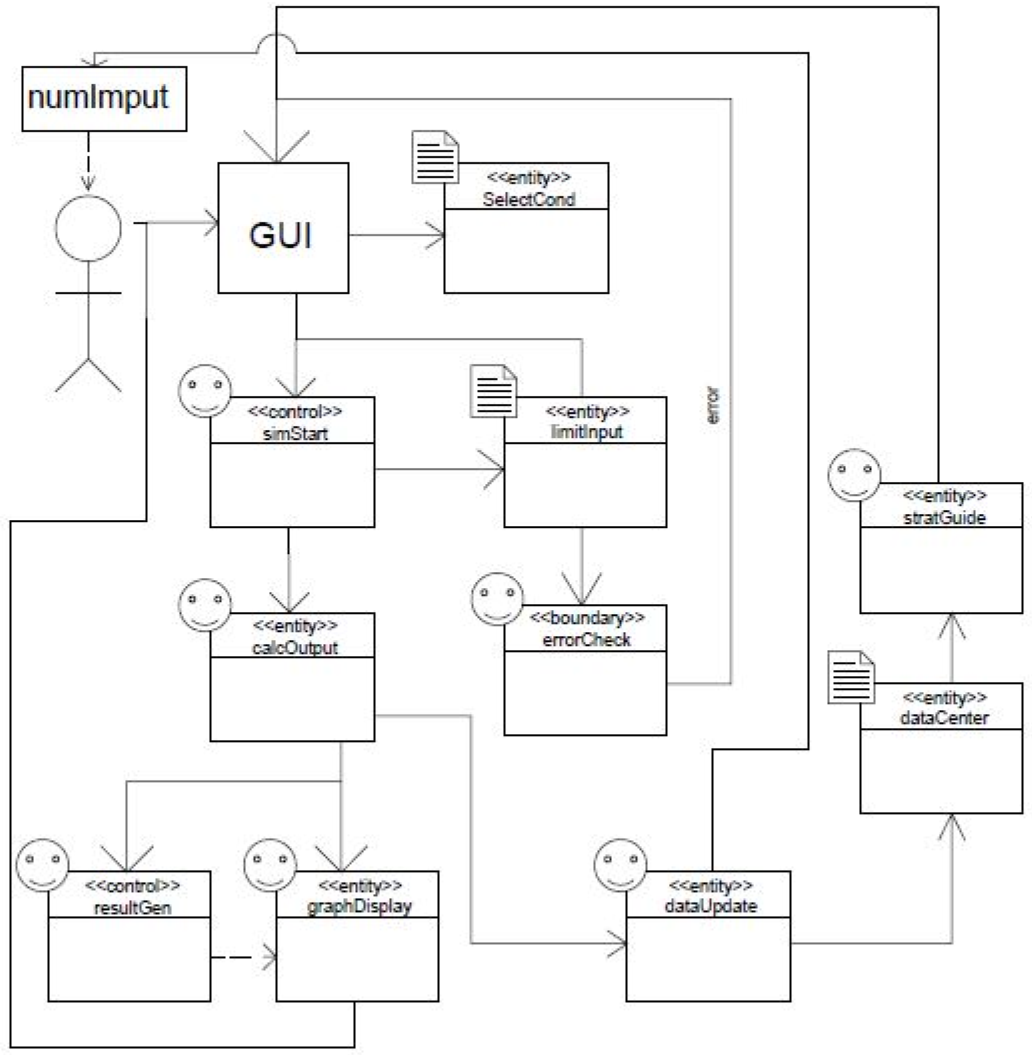


7.1.2 Association Definitions

The responsibilities above can be associated with each other to convey or update information. The arrows show that there is a relationship between two responsibilities.

Concept Pair	Association Description	Association Name
numInput ↔ GUI	The user inputs parameters via GUI.	Simulation input
selectCond ↔ GUI	User selects conditions on the GUI.	Simulation conditions
simStart ↔ GUI	User presses start to begin.	Press start
simStart ↔ limitInput	When user presses start the inputs would be compared with limits.	Compare limits
limitInput ↔ errorCheck	If the user inputs are not within limits the simulation stops, else it begins.	Simulation check
errorCheck ↔ GUI	If there is an error, a message is sent back to user on the GUI.	Error return
simsStart ↔ calcOutput	The software begins calculating for results when user presses start and there are no errors.	Simulation start
calcOutput ↔ resultGen	Output is calculated and shown on real time for users to watch.	Simulation output
calcOutput↔ graphDisplay	After all calculation, the graphs are displayed.	Simulation graphs
graphDisplay ↔ GUI	The final graphs are shown on the GUI.	Graph Output
numInput ↔ dataUpdate	User experience and strategies would be updated.	Save values
selectCond ↔ dataUpdate	User selected conditions would be updated.	Save conditions
graphDisplay↔dataUpdate	The final result would be saved and updated.	Save graphs
dataCenter ↔ stratGuide	A strategy guide would be created based on data.	Create strategy

stratGuide ↔ GUI	Send user new strategies on GUI.	Return strategy
dataUpdate ↔ dataCenter	All updated data would be in the data center.	Save data



7.1.3 Attribute Definitions

The following attribute definition seeks to identify the attributes that will be needed in each of the concepts identified. The selectCond concept deals with the settings that can be chosen by the user when setting up the simulation.

Concept	Attribute	Responsibility
numInput	playerID	saves the id of each player
	playerHistory	saves a history of each players past decisions
selectCond	fisherNum	contains the number of competing fishermen
	fishPop	contains the starting fish population
	fishLoc	Contains the number of possible fishing locations
	fishType	Contains the type of fish available. IE Bass, Catfish, etc.
	fishTemp	Contains the water temperature
	isWeatherEnabled	When true weather is considered in the simulation
	isCommunication	When true communication between fishermen is enabled in the simulation calculation.
fisherStrat	avalibleStratigies	A list of all of the strategies available for the fishermen to use in determining if they should fish and for what.
fisherExp	pastDecisions	Contains the past decisions that this fisher has made
	pastOutcomes	Contains the results of the fishers choices (win / loose)
	pastConditions	Contains the conditions (weather/ temp) when each past choice was made
limitInput	maxTemp	The maximum allowed water temperature.
	minTemp	The minimum allowed water temperature.
	maxNumFishers	The maximum number of fishermen allowed
	minNumFishers	The minimum allowed water temperature.
	maxFishingloc	The maximum number of fishing locations the simulator will handle.

	minFishingLoc	The minimum number of fishing locations. Default (1)
	simMaxIterations	The maximum number of iterations the simulation can run.
dataCenter		Contains all of the past data. This includes the past elements that could affect the simulation, fishermen choices, and the aggregated results.

7.1.4 Traceability Matrix

Each of the use cases have the following dependencies on concepts. Because UC-3 is running the simulation, it depends on almost all of the concepts. Also the Saving and restoring use cases (UC-4, UC-5) could depend on more concepts depending on how much data we save. Lastly UC-9 deals with providing help for the user and notifying users of errors. This use case encompeses each concept because they each could have errors that the user must be notified about.

Use Case	**PW (1-5)**	**numImput**	**selectCon**	**fisterStrat**	**fisherExp**	**LimitInput**	**dataCenter**
UC-1	4	X	X			X	
UC-2	4	X	X			X	
UC-3	5		X	X	X	X	X
UC-4	2				X		X
UC-5	2	X	X		X		X
UC-6	4				X		X
UC-7	3			X			
UC-8	2				X		X
UC-9	1	X	X	X	X	X	X

7.2 System Operation Contracts

Operation:	errorCheck
Cross Reference:	UC-1, UC-2
Preconditions:	User must have input settings
Postconditions:	Returns error if input settings are incorrect, allows simulation to be run otherwise

Operation:	dataUpdate
Cross Reference:	UC-4
Preconditions:	Simulation must have run successfully
Postconditions:	Stores data from simulation into data center

Operation:	calcOutput
Cross Reference:	UC-6
Preconditions:	Must have run a valid simulation
Postconditions:	Returns data on simulation to be stored or viewed

Operation:	simStart
Cross Reference:	UC-6
Preconditions:	User must have input settings that were checked and found to be valid
Postconditions:	Simulation begins running

Operation:	graphDisplay
Cross Reference:	UC-6
Preconditions:	Simulation must have successfully run, data must have been calculated
Postconditions:	Displays graphs built from the data calculated from the simulation

7.3 Mathematical Model

Minority games model can help an agent to make a decision in a competitive fishing environment. The basic fundamental for this mathematical model is an agent considered win if the fishing spot is least crowded because the agent will have a higher chance to get a fish. We will take an odd number of players N to choose one of two choices independently at each turn. Each of the N players takes an action deciding either to go to the bar (denoted as $a_i(t) = 1$) or stay at home (denoted as $a_i(t) = -1$).

$$A(t) = \sum_{j=1}^{N} a_j(t)$$

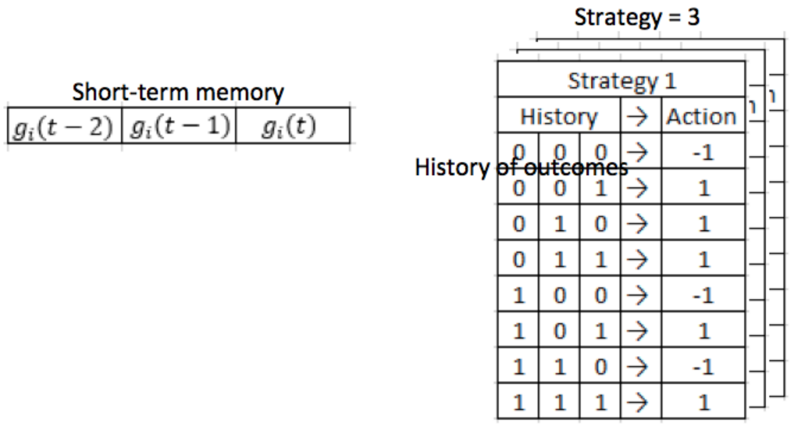
Since the number of players is odd, the summation formula will not go to zero. So, by using the above formula, we can conclude that if $A(t) < 0$, majority of the players stayed at home and the fishing spot is not crowded and a higher chance for the players to get a fish. On the other hand, if$A(t) > 0$, majority of players go to fishing and this will imply a lower chance for a player to get a fish.

$$g_i(t) =- a_i(t)A(t)$$

The function $g_i(t)$ will represent outcome of the current round for the player i and will make sure the agents with minority action are rewarded. Since we are only interested in winning (denoted by 1) or losing (denoted by 0) we will take the absolute value of $g_i(t)$.

$$g_i'(t) = \begin{cases} 0 & if\ g_i(t) < 0 \\ 1 & if\ g_i(t) > 0 \end{cases}$$

Players will have a short-term memory and limited to 3 previous outcomes. On top of that, each player will have sets of strategy which also limited to 3 strategies each player. Below is the illustration of the short-term memory and long-term memory.



Strategy 1				
History			→	Action
0	0	0	→	-1
0	0	1	→	1
0	1	0	→	1
0	1	1	→	1
1	0	0	→	-1
1	0	1	→	1
1	1	0	→	-1
1	1	1	→	1

When the simulation starts, each player will start with a random strategy. Each strategy will be able to keep score of its success rate. The higher the success rate of that certain strategy, the more likely for the player to use that certain strategy. Calculation of the success rate of that certain strategy as follows. If the S_{ij} suggested to stay at home ($a_{ij} = -1$), and the fishing spot turn out to be crowded ($A(t) > 0$), then S_{ij} is added with one point. Conversely, if the fishing spot turn out to be less people ($A(t) < 0$), the point will be lowered by one.

$$\sigma_{ij}(t) = \begin{cases} \sigma_{ij}(t-1) - 1 & if\ a_{ij}A(t) < 0 \\ \sigma_{ij}(t-1) + 1 & if\ a_{ij}A(t) > 0 \end{cases}$$

Since we are adding more features that will influence the decision making of the players, we will add one more step before calculation the success point of each strategy. After each players gets the suggested decision, a_{ij}, from the strategy S_{ij}, the decision can be change by the following factors.

· Skill and experience rank
· Frequency of communication
· Amount of each type of fish
· Fishing duration
· Weather pattern

We decided that each factors will contribute 20 percent increase of influence to change the current decision, a_{ij}, of the player. The higher percentage of the influence the more likely the player to change their decision. The threshold influence percentage to change the player decision is 70 percent. To calculate the influence percentage, p, we will use the formula below.

· Skill and experience rank

o This will be rank from 1 – 5, so to get the influence percentage for this factor we calculate it as below.

$$E = \frac{rank}{5} \times 20$$

· Frequency of communication

o Since this represent the percentage of communication between the players, this can be easily calculated as below.

$$C = \frac{Frequency\ of\ communication}{N, number\ of\ players} \times 20$$

· Amount of each type of fish

o The type of fish that players wanted to catch will be accumulated first. Then the percentage of players that interested in catching the fish will be calculated. The higher the percentage of interested players in catching a certain type of fish, the lower the influential rate of the fish. This is because, more players are interested in one type of fish, will cause a higher competitiveness. So it is more likely that a player to just stay at home. To get the influence percentage of this factor we will calculate it as below.

$$F = \left(\frac{Number\ of\ interested\ players\ in\ the\ type\ of\ fish}{Amount\ of\ type\ of\ fish} \times 20\right) - 20$$

· Fishing duration

o We assume that the player that spend a longer time in fishing will have a higher chance in getting a fish and since we go for a success rate of each player in a day, we will calculate the fishing duration influence rate as follow.

$$T = \frac{Fishing\ duration}{24\ hours} \times 20$$

· Weather pattern

o In general, the best temperature to fishing is around 55 – 80 Fahrenheit. The average temperature would be 67.5 Fahrenheit. The influence rate for weather pattern will be calculated as follow.

$$W = \left(-\left|\frac{Average\ Temperature - Today's\ Temperature}{Average\ Temperature}\right| \times 20\right) + 20$$

After all the influence rates of each factors has been calculated, we can get the overall influential percentage (denoted by p) and decide if the player should changes the decision. If $p < 70$, the decision a_{ij} that has been made by S_{ij} will not be change. If $p > 70$, any decision $a_{ij} = -1$, will be change to 1.

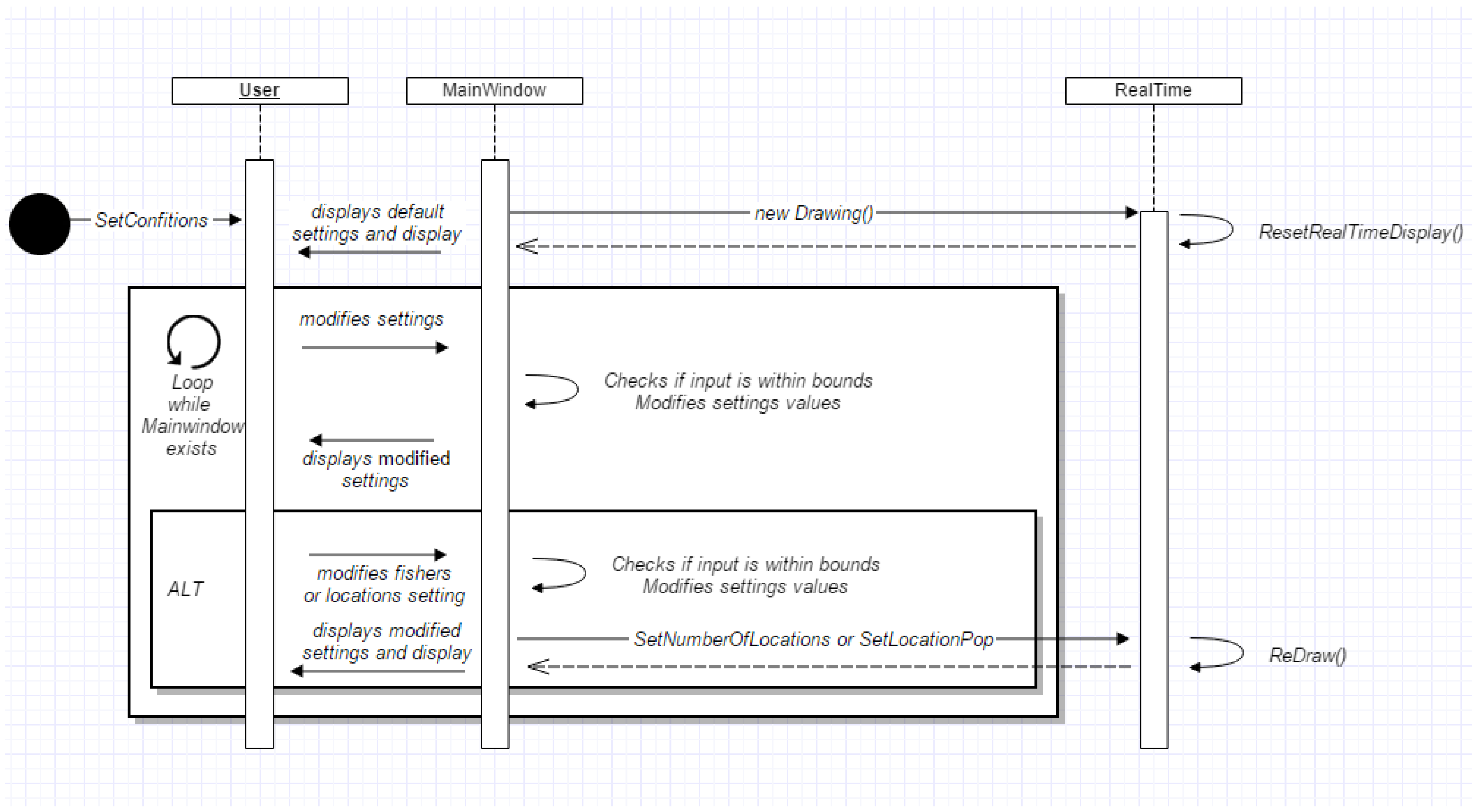
$$g_i'(t) = \begin{cases} 0 & if\ g_i(t) < 0 \\ 1 & if\ g_i(t) > 0 \end{cases}$$

8. Interaction Diagrams

UC-2: SetConditions:

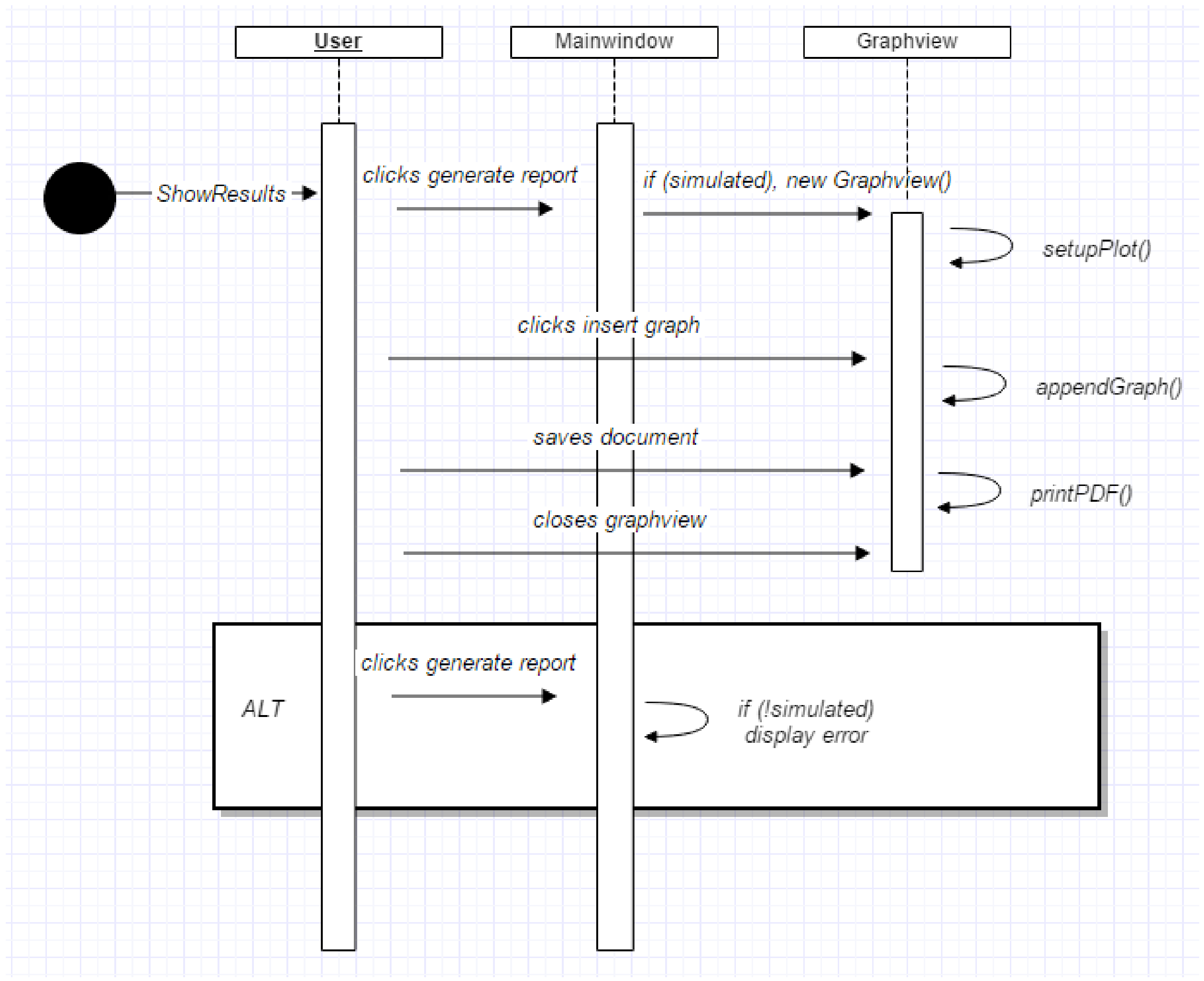
The SetConditions use case allows users to modify the default simulation settings. Mainwindow displays default settings and a default realtime display to the user on startup. The user is able to modify settings. For enabling poor weather, this is accomplished by clicking checkboxes. For the other settings, the user can manipulate a slider or input numerical values. Upon entering a numerical value for a setting, Mainwindow compares the input with the bounds for that setting. If the input value is outside our set bounds, the setting is still modified but only to the maximum or minimum allowed by the bounds.

When modifying the number of fishers or locations, the real time display is updated along with the settings.


User
MainWindow
RealTime
SetConfitions
displays default settings and display
new Drawing()
ResetRealTimeDisplay()
modifies settings
Loop while Mainwindow exists
Checks if input is within bounds Modifies settings values
displays modified settings
ALT
modifies fishers or locations setting
Checks if input is within bounds Modifies settings values
displays modified settings and display
SetNumberOfLocations or SetLocationPop
ReDraw()

UC-6: ShowResults:

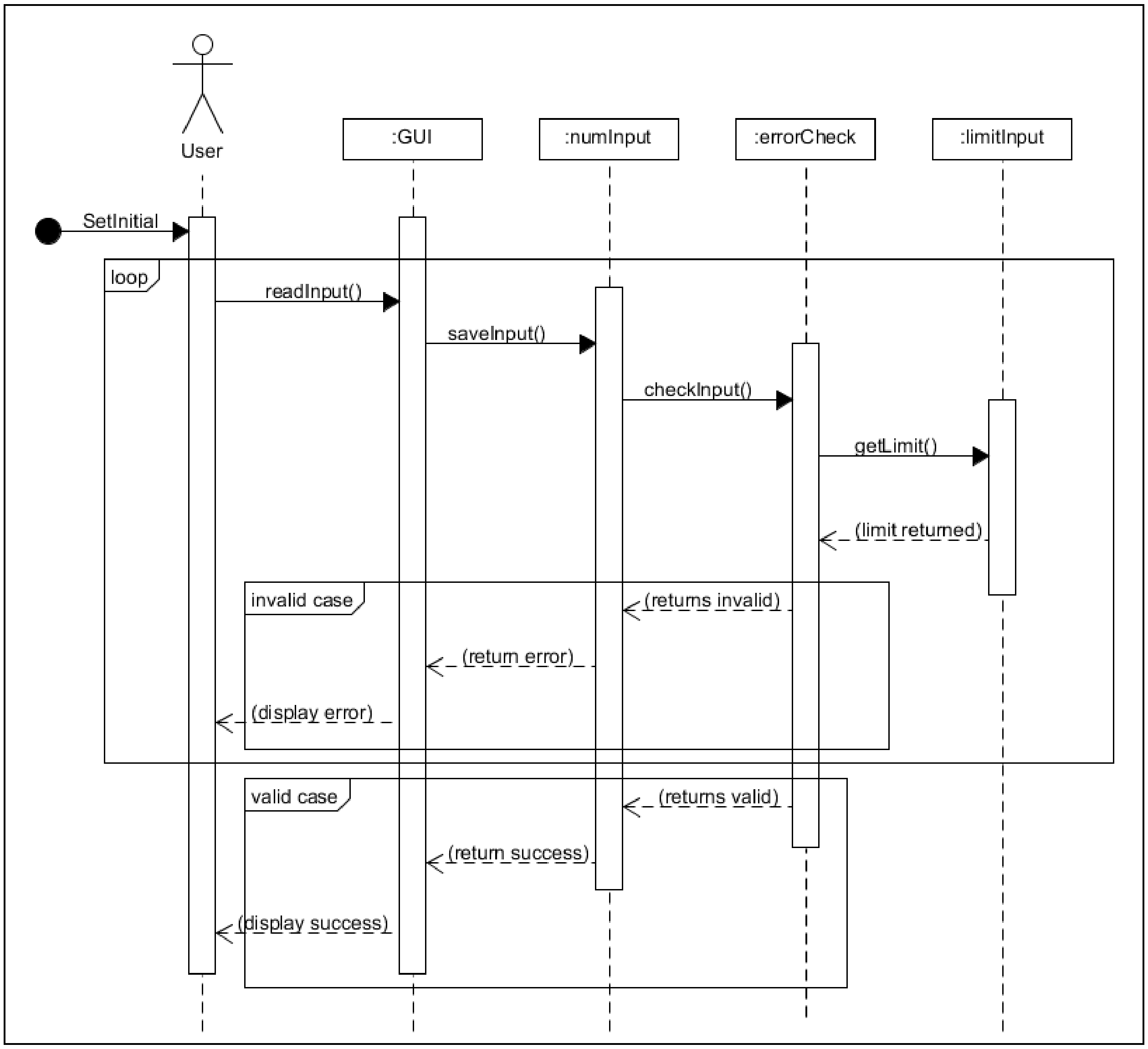
In ShowResults user gets results back from initial chosen parameters. The results would consist of real time simulation that the user can watch and the final graphs. The selection of conditions and input values are further elaborated on UC-1 and UC-2. The output would be calculated, calcOutput, with given conditions and initials. The calculation progress can be seen by the user. When the calculation is done and the user clicks generate report, the final graphs would be displayed.



The other two fully dressed use cases, SetInitial and SaveData, have been deprecated and are no longer accurate. Their system sequence diagrams are reproduced below.

UC-1: SetInitial:

The SetInitial use case is for the general sequence of actions when a user wants to set a parameter for their simulation. Note that this is different from UC-2, where the user decides what conditions affect their simulation. In this use case, the value is saved and then checked against the stored limits for that specific value. If it is within these limits it passes the check, and the user will see that they successfully set the parameter. If it is not within the limits, the user will see an error and be prompted to input a new value.



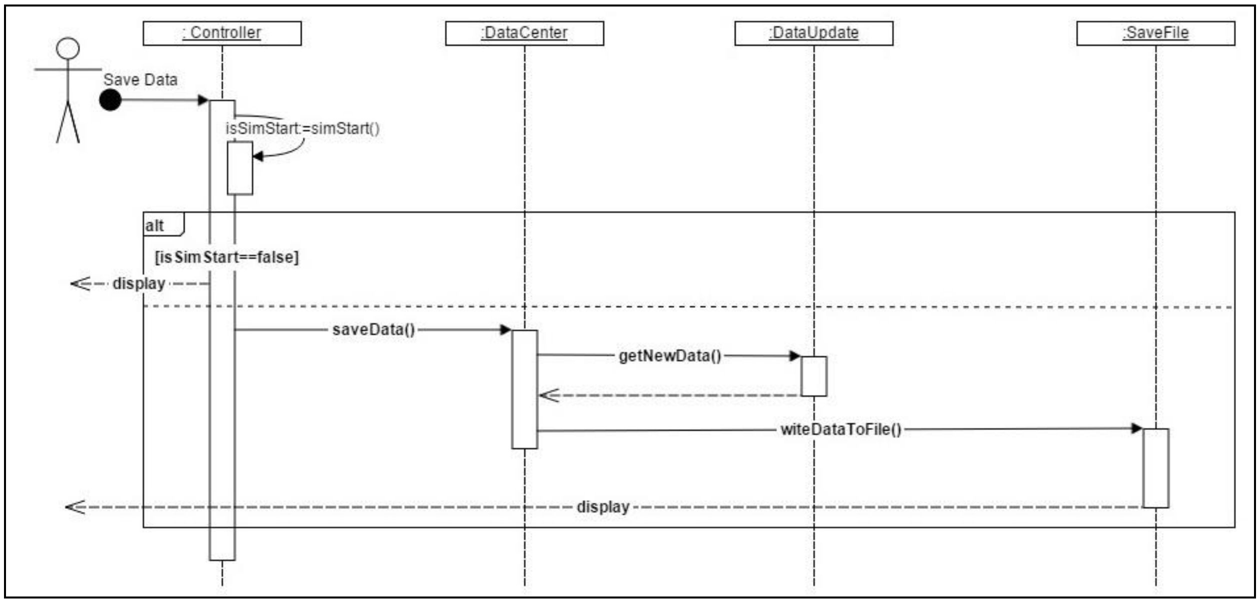
Design Sequence Diagram for Use Case 1: SetInitial.

UC-1: SetInitial Design Principles:

In UC-1, objects have high cohesion and are loosely coupled. LimitInput performs no computations and communicates with one other object. ErrorCheck performs comparisons to compute error and returns its results to NumInput. NumInput saves the ID of each agent and agent history. The GUI passes data on to NumInput and receives error checked data back so it can notify the user of any errors.
No objects follow the expert doer principle, as the first object to know passes on its knowledge leaving another object to act on it.

UC-4: SaveData:

When the user initiate save data process, the controller will check if the simulation has been started before. If the value in isSimStart is false, the controller will display an error message and request the user to run the simulation. When the precondition, simulation has been started, is fulfilled the controller will call saveData(). DataCenter will take over the process and request new data from DataUpdate by calling getNewData(). When the new data is available, it will be passed to SaveFile. SaveFile will arrange the data in appropriate format for easy retrieval in the future. User will be prompt to choose a destination and also set the name of the file to be saved. simStart() might not be the subroutine of the Controller. It may be a call to Simulation and then a Boolean value is returned to identify if the simulation has been started. The exclusion of Simulation is for simplicity to this UC-4 design sequence diagram.



Design Sequence Diagram for Use Case 4: Save Data. User is allowed by the system to save the simulation data for future use. This diagram only shows part of the system interaction.

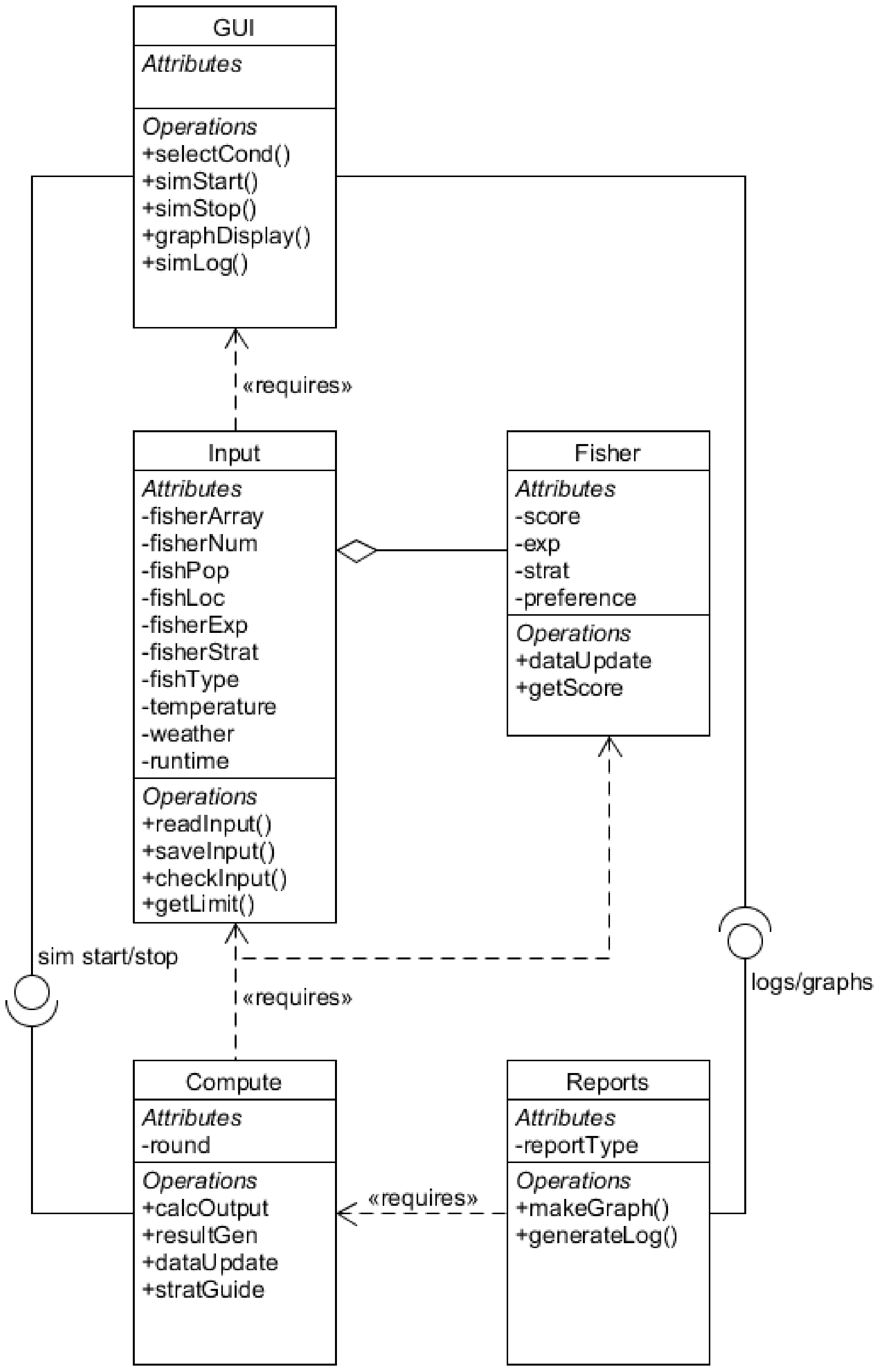
UC-4: SaveData Design Principles:

The objects involved in UC-4 have high cohesion and are loosely coupled. The computational tasks are spread across all objects. The Controller object performs a single comparison. DataCenter retrieves data from DataUpdate, which knows the simulation results. SaveFile writes this data to a file. Similarly, loose coupling is achieved by limiting each object's communication responsibilities. Each object sends, at most, one message.

The Controller object follows the expert doer principle. It is the first to know about the user's save request and takes action, checking to see if a simulation has been performed.

9. Class Diagram and Interface Specification

Class Diagram



Data Types and Operation Signatures

The classes shown above will have the following operations and attributes. Methods and attributes that are private to the class are indicated with a “-” sign and a “+” sign is used to indicate public items.

GUI: The GUI class is responsible for generating the live map and providing the needed controls for the user inputs.

```
+ selectCond()
+ simStart():
+ simStop():
+ graphDisplay():
+ simLog(Event):
```

Input: Contains information about the environment and the input from the user.

```
- fisherArray: Fisher
- fisherNum: int
- fishPop: int
- fishLoc: int
- fisherExp: int
- fishStrat: int
- fishType:int
- temperature: int
- weather: int
- runtime: int
```

Compute: Takes the environment factors and fisher inputs and computes the results of a round of the simulation.

```
- round: int
+ calcOutput()
+ resultGen()
+ dataUpdate()
+ stratGuide()
```

Fisher: Contains the details of each individual fisher such as their strategy and experience. This class also keeps track of each individual fisher's score.

- score: int
- exp: int
- stat: int
- preference: int
+ dataUpdate(): void
+ getScore(): int

Reports: Generates the final comprehensive report of the simulation. This includes any graphs of success of the fishers over time and the strategies used.

- reportType**:** int
+makeGraph(): void
+ generateLog(): void

Traceability Matrix

Domain concepts	Classes				
	GUI	Input	Compute	Fisher	Reports
numInput	X	X		X	
selectCond	X	X			
fisherNum		X			
fishPop		X			
fishLoc		X			
fisherExp				X	
fisherStrat				X	
fishType		X		X	
fishTemp		X			
linitInput	X				
simStart	X	X	X		
errorCheck	X				
dataCenter		X		X	
dataUpdate		X	X	X	
calcOutput			X		
resultGen	X		X		
graphDisplay					X
stratGuide	X				X
GUI	X				

The above Traceability Matrix shows that each class encapsulates several domain concepts. The domain concepts involving knowing something about a fisher will be implemented in the Fisher class and the items about knowing the environment will primarily be implemented in the Input class. Concepts of doing the simulation and corresponding calculations will be in the Compute class although the GUI class will signal that the users wishes to start the simulation and the Input class contains values needed in the simulation.

The dataCenter concept of having all of the results, fisher experience and the success of their strategies have been split between the Input class and Fisher Class. This was because items such as last rounds winner are the same irrespective of individual Fishers and other items such as the individual’s success are unique to the Fisher. The stratGuid concept of giving a guide for strategies will be implemented in both the GUI to allow for users to have easy access and also in the Report class to give additional ideas for strategies.

10. System Architecture and System Design

10.1 Architectural Styles

This software's architectural style can be described as event-driven. An event-driven architecture or EDA is a type or architecture where the results revolves around the production, detection, and consumption of events. It also take account the responses between these events. Events are any instances that are significant to the software. This type of architecture basically consists of event creators and event consumers. One advantage of an EDA is the possibility of creating large numbers of event creators and event consumers.

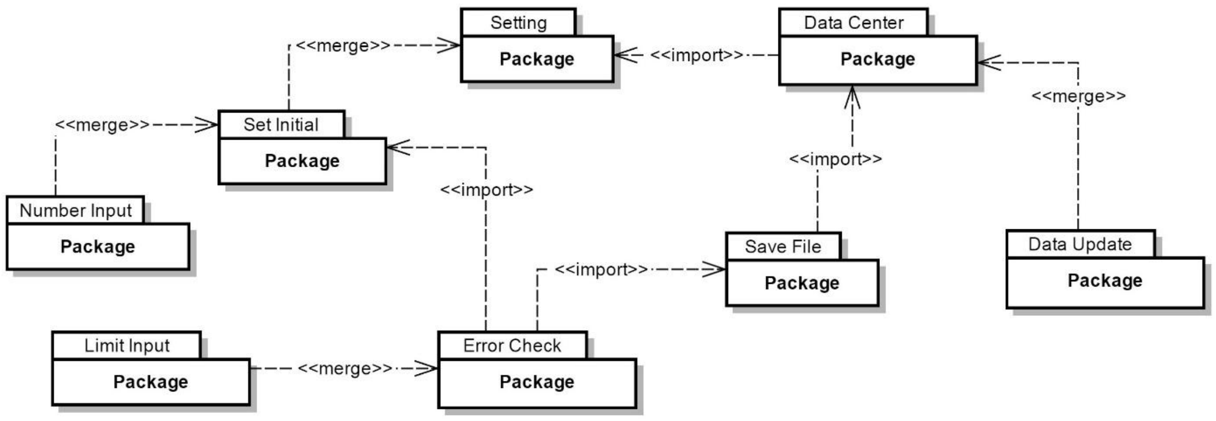
Many events are participating in the Fishing Minority Game software. Every simulation started by the user causes a production of visual results. The user would respond based on the given results, like whether the strategies or fishing spot should be changed. The user starts the production, detects, and consumes events. In this case, the user can be seen as both the event creator and consumer.

However, there are more interactions happening inside the software that produces, detects, and consumes events. For example, when the user press simulation start the software checks if the the inputs are valid. If the inputs are valid, the output calculations starts. The error check creates an event where the output calculation would respond by consuming input and producing results . In this example, the error checker can be seen as a consumer of inputs and the creator of error messages. The output calculator's response would depend on the result of the error check. In this perspective, the error check can be called an event creator while the output calculator can be called the event consumer.

Another architectural style to describe this software is the monolithic application. This software's user interface, data input, data output, data processing, and error handling are together in one application. This program enables the user to provide input and expect to receive output in the same platform. All the information are processed and stored in the same application. Also, there is an error checker built in the same application. In other words, the Fishing Minority Game software does not need other applications to function.

In addition, this software can also be seen using Model-view-controller architecture or MVC. This type of architecture have three interconnected parts: controller, model and view. The controller is where the input is given to be sent to either model or view. The model organizes the data, calculations and limitations of the program. The view is where the output of the program is shown. For the Minority Game, the error checker act as controller where it decides if inputs should be sent to the model or view. If there are no errors, input is sent to the data center, the program's model, where it will be calculated. Otherwise, an error message would be sent to the view. Results will be sent to the GUI, which is the software's view, presented as graphs.

10.2 Identifying Subsystems



UML packages interactions for use cases

UML package diagram provides the overview of the dependency between the packages. The diagram drawn is based on 4 use cases: SetInitial (UC-1), SetCondition (UC-2), SaveData (UC-4), and ShowResult (UC-6). By creating this interaction diagram we are able to identify most important packages for our program.

Error Check package may be extended to Set Initial and Save File. Rather than having different Error Check packages, we can integrate all the error checking in one package. This can simplify the implementation of the whole program. Besides, a programmer can easily focus on his task to implement the error checking package. Save File depends on Error Check in the sense that the simulation need to be run before a data can be save. By separating the error checking and the save file, it provides a clear boundary of the functions of every packages.

Data Center package helps to ease the flow of data within the program. Since we are working on a program that heavily depends on multiple inputs and outputs, we need to provide an interface for data exchange. By having a special package to deal with data, we are able to provide data consistency within the program. This data consistency is important for the simulator when it comes to dealing with previous results and a new input that provide by Data Update. As there are too many data the simulator is dealing with, having a Data Update package is compulsory because it keeps the new data away from the old data. This separation makes the operation on the data easier.

10.3 Persistent Data Storage

This system does include the need to save data, for Requirement 5. More specifically, for Use Case UC-4:

UC4 SaveData - The user can save current simulation data. Saved data will be backed up.

This involves the saving of a report generated from a specific simulation. The report in question will vary based on what graphs/text/etc. the user decides to put into the report.

The storage management strategy is simply to allow the user to save the generated report and its data to their own hard drive. This way, the data is not dependent on the program itself, but rather can be distributed by the user online or in e-mail to others who may be interested. The report generated is a PDF file, so the user can also later take out specific graphs or pieces of data for separate use, or even edit the report to include analysis or fix errors!

10.4 Global Control FLow

Execution Order:

This system will be procedure-driven, executing in a linear fashion. Every user has the same options available to them in the same order:

a. Setting variable values
b. Choosing options for simulation
c. Running the simulation
d. Observing/saving data from the simulation

This doesn't change based on the user. You can't run the simulation without choosing options, and you can observe the data without running the simulation, etc.

Time Dependency:

The system is, in general, an even-response type system. The events it responds to are the choosing the initial options for a simulation, and starting the simulation. The response is, of course, to run the simulation. The only catch is that the simulation itself runs on an internal timer – events in the simulation occur one after each other in simulated time. Essentially, the system is not a real-time system, but it does simulate one.
Concurrency:

The system does not have multiple threads. While it's conceivable that a user might want to, for example, run multiple simulations at once this is not a feature that we have decided is worth implementing. Simulations should not take significant amounts of time to run, and some of their value is watching them in real time. If simultaneous simulation running is truly required, the user can simply run multiple instances of the program.

10.5 Hardware Requirements

Preliminary hardware requirements:

- Screen display of adequate size
- Finite amount of memory for program, extra memory for saved data
- Processor power for simulation
- I/O devices for interacting with program
- RAM for running program

Screen display:
About a third of a regular 1920x1080 screen is required to fully display everything at once and see it all reasonably well. A reasonable minimum requirement would be 1280x720 pixels.

Memory:
The program has an installation size of 140MB. We can also assume that the data saved for a user-generated report will be up to 10mb. Assuming the average user would want to save at least 10 reports, total recommended space for memory is 150mb.

Processor power:
The program is expected to take very limited processor power. A 1GHz processor should be sufficient.

I/O devices:
The equivalent of a keyboard and mouse will be required. A keyboard for inputting values, and a mouse for selecting options.

RAM:
The program is not expected to be very resource intensive, so 512MB RAM should be sufficient.

Hardware	**Minimum Requirement**
Display Resolution	1280x720
Disk Space	At least 140MB, up to 300MB
CPU	1GHz
I/O	Keyboard and Mouse
RAM	512MB

11. Algorithms and Data Structure

11.1 Algorithms

Decision Making

The algorithm is made to compute a unique decision for every agent. The decision is either to go fishing (denoted as 1) or stay at home (denoted as -1). At first every decision of an agent is randomly chosen from a random strategy. Then, every decision may change by the percentage of influence threshold, ρ. The decision is determine using the logic below:

```
if $\rho < 20$
        decision that is made by the strategy is kept.
else if $\rho > 65$
        decision will be change to $1 -$ go to fishing.
else
        decision that is made by the strategy is kept.
```

The value of influence threshold depends on the factors below:

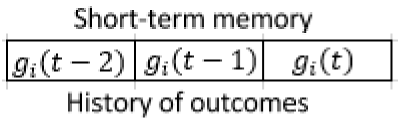
- Skill and experience rank
- Frequency of communication
- Amount of each type of fish
- Fishing duration
- Weather pattern

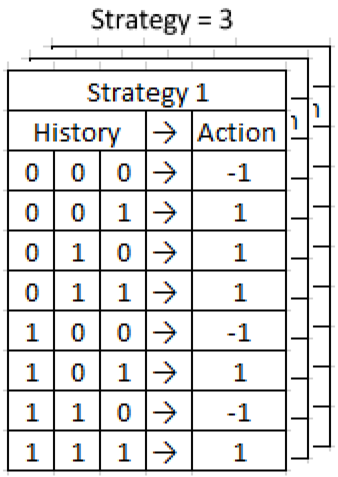
Since some of the factors above are unique for each agents, it will be able to preserve the uniqueness of every decision. Every factors will contribute 20% to the influence threshold.

Strategy

Every agent will have a short-term memory and a long-term memory. Short-term memory is limited to 3 previous outcomes of the agent winning and losing. Long-term memory is the strategy that is used by the agent to make the initial decision before taking into account of influence threshold.

Since there are 8 possible outcomes from the short-term memory, the strategy that can be generated from these outcome is 256. Every agent is allow to have 3 strategies, this will result in 2,763,520 different combinations of strategies. Every agent will get a random combination of 3 strategies and it will be likely that every combination is unique.




Strategy = 3

Strategy 1				
History			→	Action
0	0	0	→	-1
0	0	1	→	1
0	1	0	→	1
0	1	1	→	1
1	0	0	→	-1
1	0	1	→	1
1	1	0	→	-1
1	1	1	→	1

The process to make the early decision is shown below:

```
strategy = choose the strategy that has a higher score
for i = 1 →8
    if history == strategy history [i]
        early decision = strategy action [i]
return early decision
```

At the beginning of every simulation, all the strategies' score are zero. So, it can be conclude that the initial strategy of every agent is random. If the agent won the round the strategy score will increase by one point. Conversely, every losing round the strategy score is lowered by one. The early decision will be passed to the decision making where the influence threshold of the agent will be calculated and the early decision may be changed.

Overall Process

Below is the overall process of how every decision of an agent being made:

```
strategy = choose the strategy that has a higher score
for i = 1 →8
    if history == strategy history [i]
        early decision = strategy action [i]
```

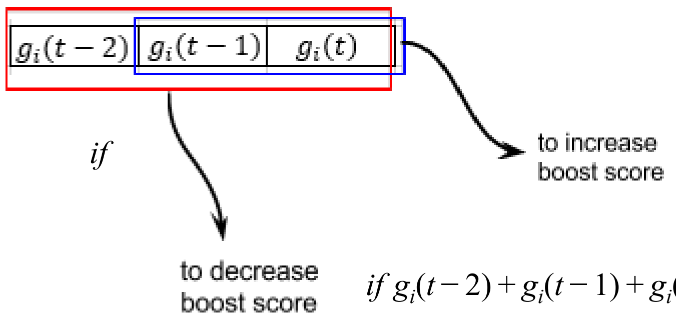
```
if ρ < 70
    decision = early decision
else if ρ > 70
    decision = 1
```

Strategy score will be calculated when all the decisions have been made. Plus for a strategy to earn the score the decision must not be changed by the influence threshold. The logic is shown below:

```
if ρ < 70
    if majority go to fishing and decision == −1
        strategy score increase by one point
    else strategy score lower by one point
    if majority stay at home and decision == 1
        strategy score increase by one point
    else strategy score lower by one point
```

Upgrade Skill and Communication

To make the agent behave more like human, we add the capability for an agent to upgrade his skill and communication, thus, result in higher threshold. To do so, we add a boost counter for every agent. This boost counter will increase add different rate because it depends the rate of winning and losing of the agent. In order to make this possible we use the agent history. To increase the boost score we use the two most recent decision of an agent. If the sum of $g_i(t-1)$ and $g_i(t)$ is equal to 2, which indicate the agent has gone to fishing for 2 consecutive days, the boost score will increase by 1. The boost score will decrease by 1 if the sum of all 3 past history is equal to -3, which indicate the agent has been staying at home for three consecutive days.



if $g_i(t-1)+g_i(t)$) = 2

boost score is increase by 1

if $g_i(t-2)+g_i(t-1)+g_i(t)) = -3$

```
        boost score is decrease by 1
```

Next, to link the boost score to the skill and communication upgrade, we assign one boost score for each one of the factor. The logic to increase the skill and communication is shown below:

```
if boostscore.skill > 8
    skill will increase by 1
    reset boostscore.skill to zero
if boostscore.communication > 5
    communication will increase by 1
    reset boostscore.communication ro zero
```

By doing applying the algorithm above, we allow the skill and communication increase at a different rate. Also, this algorithm provides a different upgrade rate for every agent. Thus, each agent will have their own unique characteristics.

Adapting New Strategy

We also provide the ability for the agent to adapt new strategy. Adapting new strategy is important in the sense that giving a better opportunity to an agent to perform better if the strategy that he is using perform badly. So, in order to do this, we track of the agent winning and losing frequency. The logic is as below:

```
if agent makes a minority decision
    winning score is increase by 1
else if agent makes a majority decision
    losing score is increase by 1
    winning score is reset to zero
if winning score = 3
    losing score is reset to zero
```

The logic above describe how the agent score is changed. Every minority decision has been made, winning score is increased and every majority decision is made, losing score is increased. Winning score will always be reset to zero for every majority decision is made. And when the winning score is equal to 3, we reset the losing score. By doing this we provide the ability to the agent to keep track of its strategy performance. We will only allow the agent to adapt new strategy when his losing score is greater than 5 and after that we reset all the scores back to zero in order to analyze the new set of strategies.

11.2 Data Structures

For the basic storage of simulation values and parameters, no special data structures are needed. The values can be stored in simple string or integer variables. This includes elements like fisherPop.

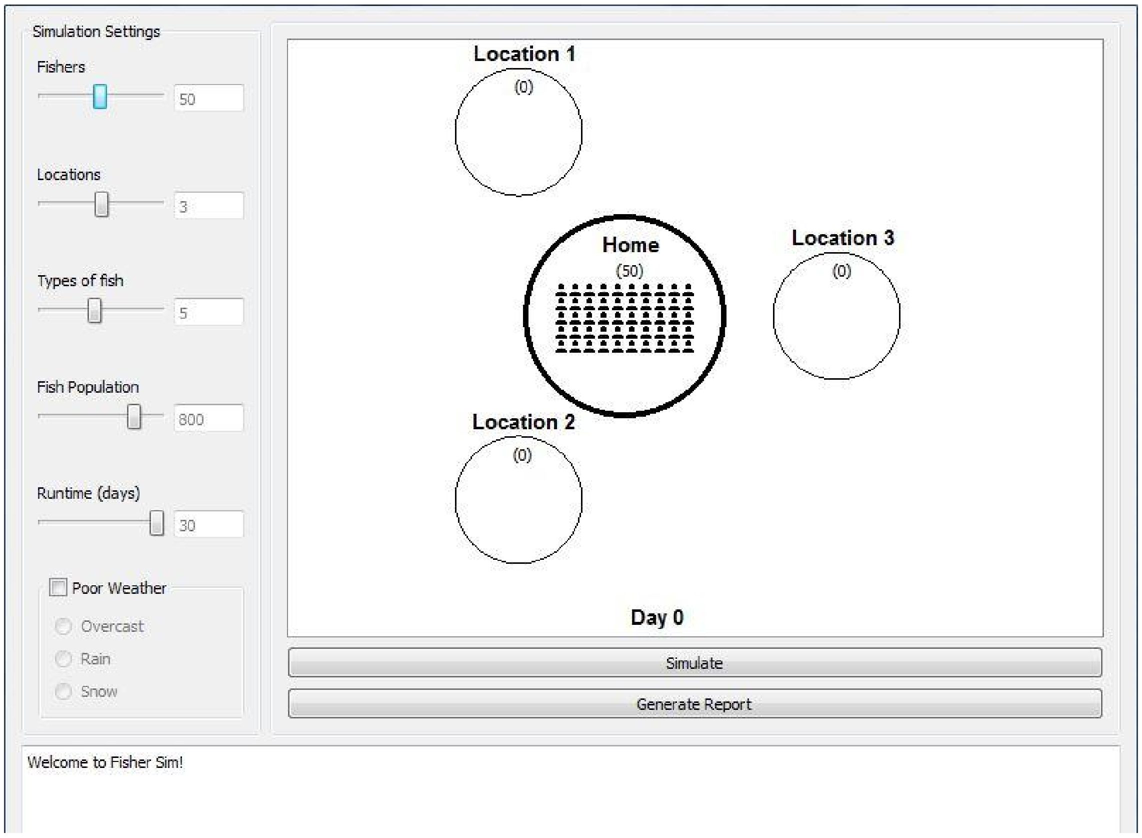
For handling the simulations themselves, more complex data structures are required.
First, the fishers themselves need to be organized. The fishers will be stored in an array, since they are all the same type and need to be accessed and updated individually. Because each fisher needs to be accessed quickly, and array makes the most sense for performance reasons.

Second, the fishers themselves will have linked lists containing their key characteristics, such as what they are currently fishing, their current strategy, etc. Because all of these characteristics will need to be updated after each round, it makes sense to keep them in a linked list so that the program can traverse each characteristic and update it one at a time. This ensures complete updates, and also allows for flexibility in adding or removing characteristics from the fisher’s linked lists.

12. User Interface Design and Implementation

When we developed demo 1, we had the program's core but we still needed to connect some of the graphs and also add graphic visualization to the program. In order to improve we create the real and final graphics and we connected parts of the program that needed to be finished, such as generate report's graphs.

Once the customer runs the program figure 11-1 is the first that we see. As we see it is pretty intuitive, having on the left side all the variables that we can modify, such as number of days that we want to simulate o the type of weather.



Welcome to Fisher Sim!

Figure 11-1 initial interface

First step:

The first variable that we can change on the left panel of our UI is the number of Fishers. As we increase the number of fishers, we can notice that in the graphical representation increase the number of people and once we hit number 72 a “plus” mark (sign) is inserted due to the fact that there is not enough space to represent all fishers.

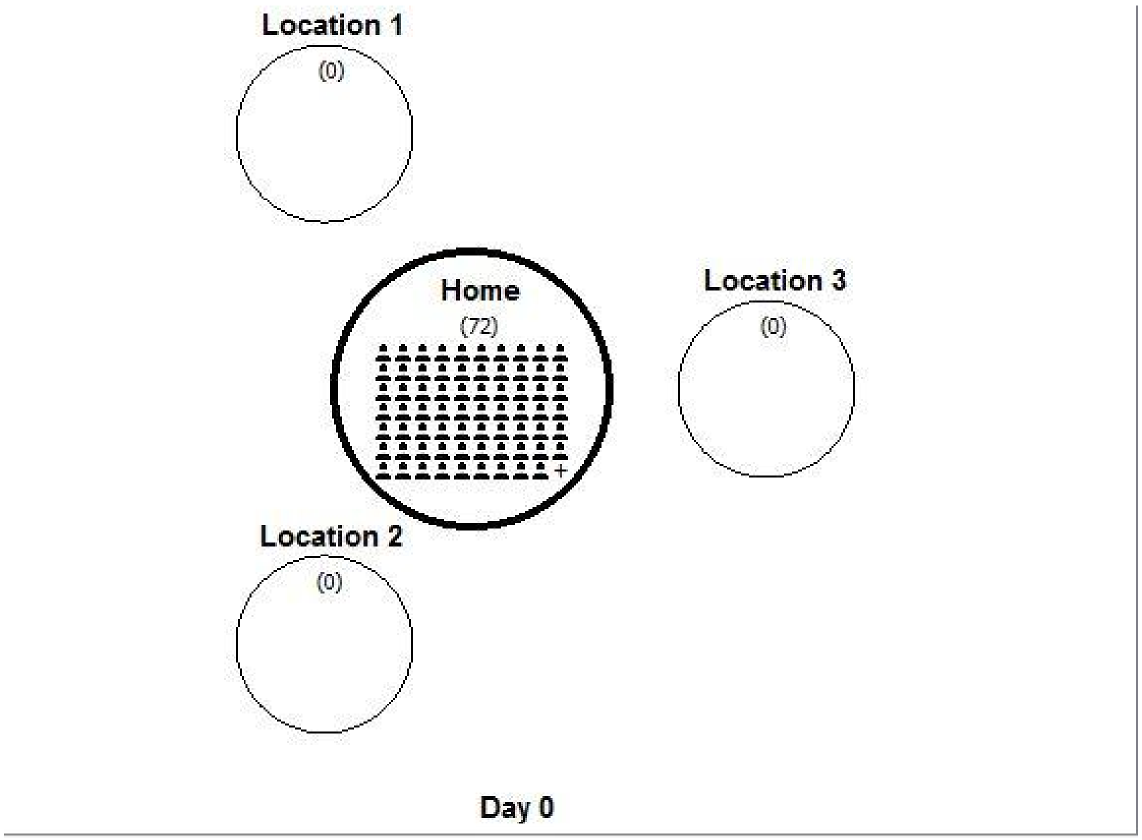
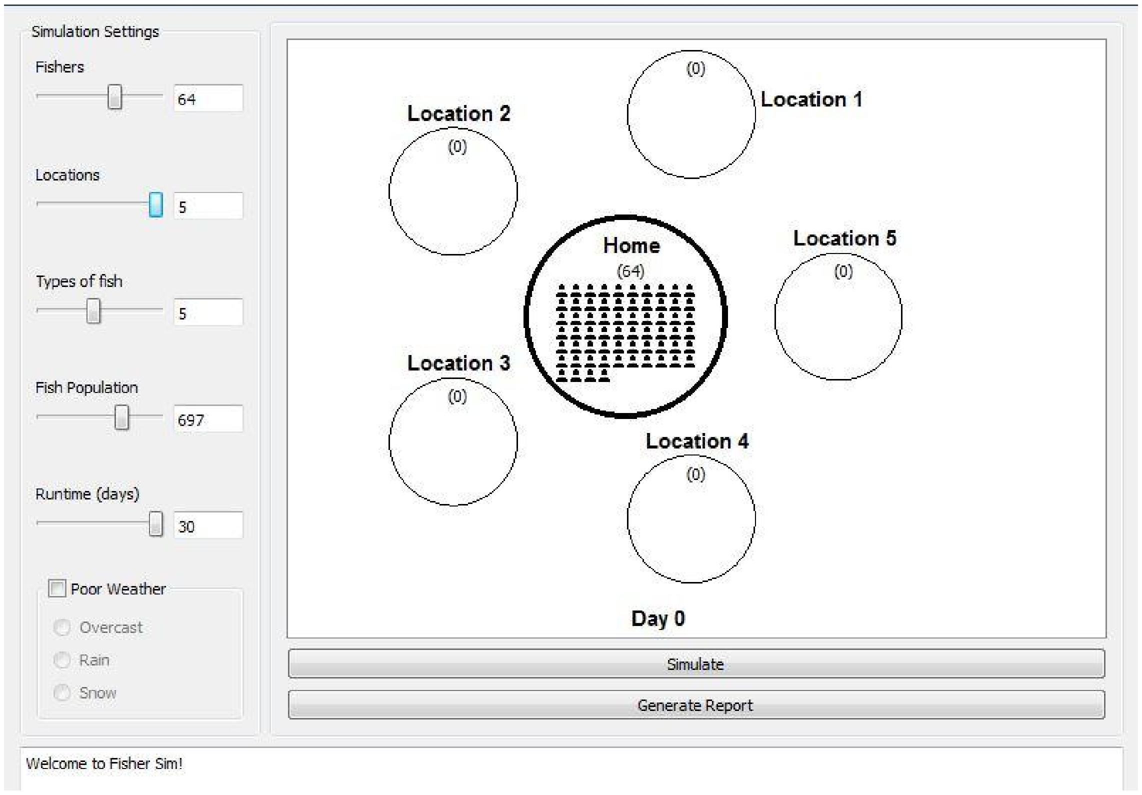


Figure 11-2 fishers

Second step:

The next variable that we can change is number of locations. And as in the previous case, if we increase it, more circles will appear at the screen, as the location representation, however the number of fishers in these different locations will be 0 until we press the “simulate” button. The maximum number of locations that we can have is 5, however we could change that top if we need, by just changing the code.


Simulation Settings

Fishers: 64

Locations: 5

Types of fish: 5

Fish Population: 697

Runtime (days): 30

Poor Weather

Overcast

Rain

Snow

Welcome to Fisher Sim!

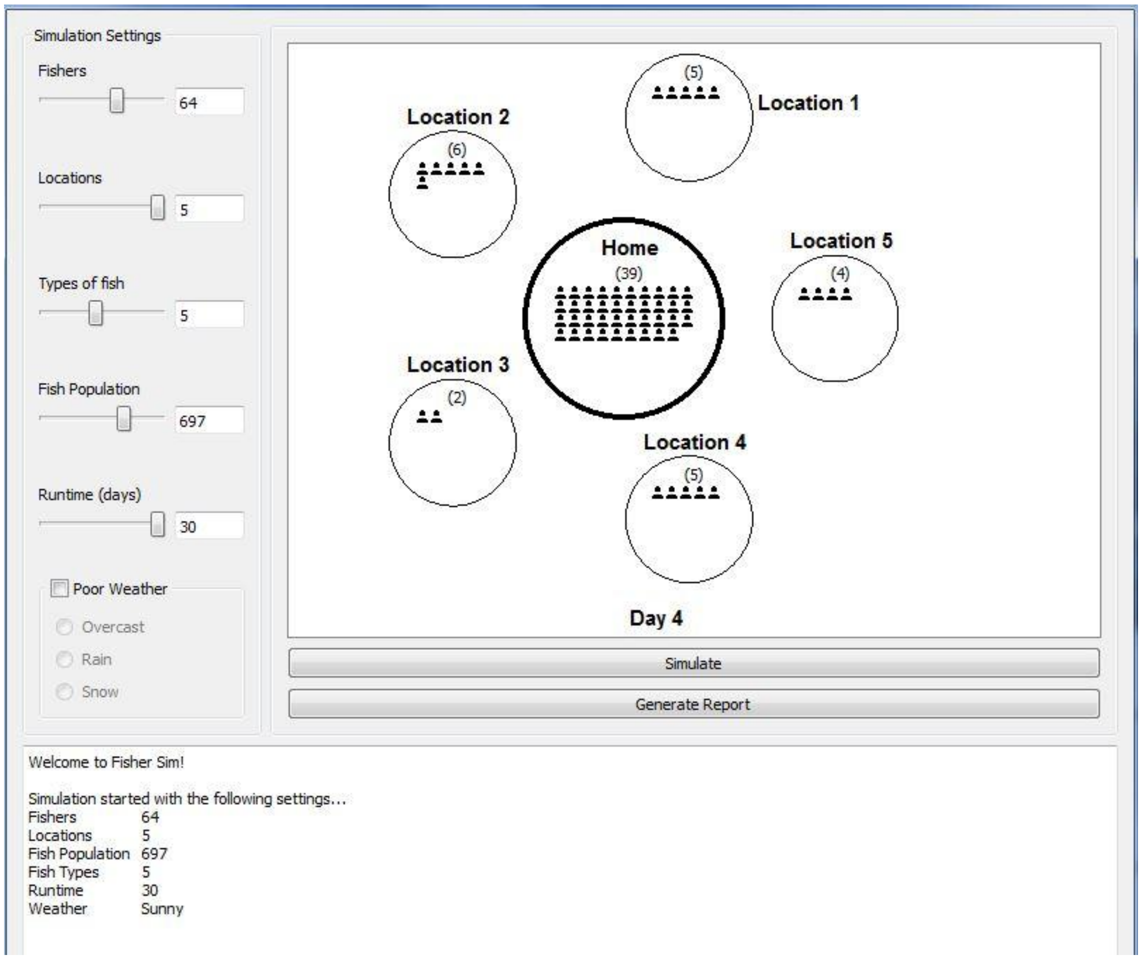
Figure 11-3 locations

Third step:

Once we have changed and select all the variables that we have on the left side of the UI, we can start the simulation and the program start to distribute all the fishers in the different spot fishing locations and at home.

Another change that we can notice is that on the bottom we don't have any more the phrase "Welcome to Fisher Sim!", in fact now we have the parameters that we have selected previously.

Finally the last change that we see is that the number of days passes by as the simulation lasts.



Fishers 64

Locations 5

Types of fish 5

Fish Population 697

Runtime (days) 30

```
Welcome to Fisher Sim!

Simulation started with the following settings...
Fishers          64
Locations        5
Fish Population  697
Fish Types       5
Runtime          30
Weather          Sunny
```

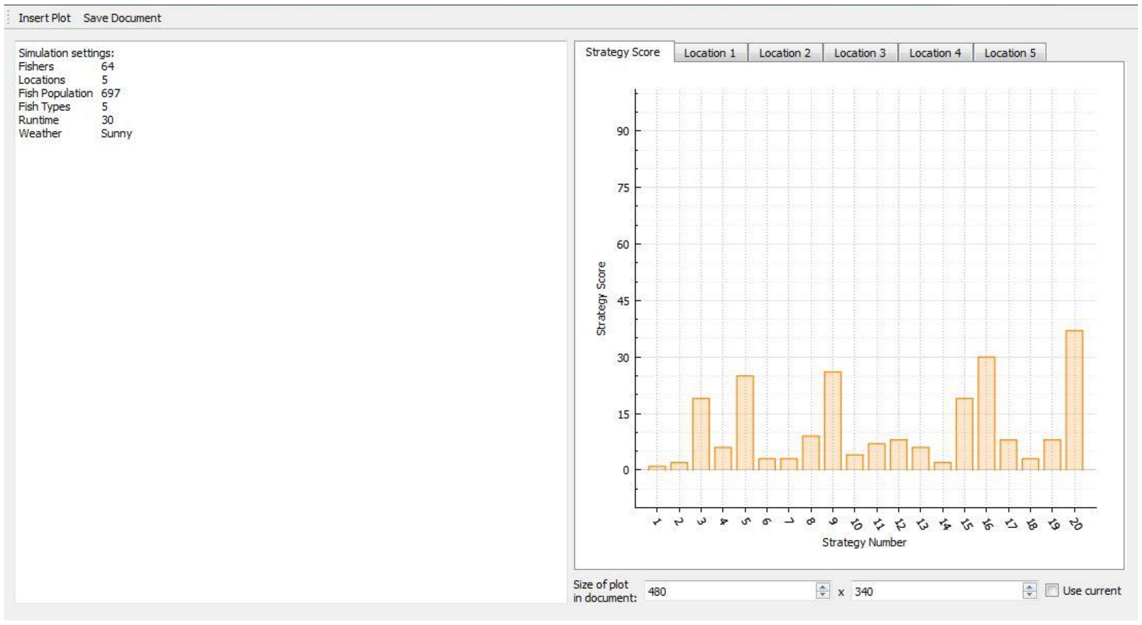
Figure 11-4 simulation

Generate Report:

Once the simulation has ended we can click the “Generate Report” button and then we can have our report shown in a different window on another UI.

Now we have on the left side of the screen the values of the selected variables and on the right side of the screen we are going to be able to see the different graphs that we have obtained. In particular, we have 6 possible graphs. The first Graph is the “Strategy Score”, where we can see the scores that we have for the 20 possible strategies.

The rest tabs that we have are the locations’ graphs.


Insert Plot Save Document

Simulation settings:

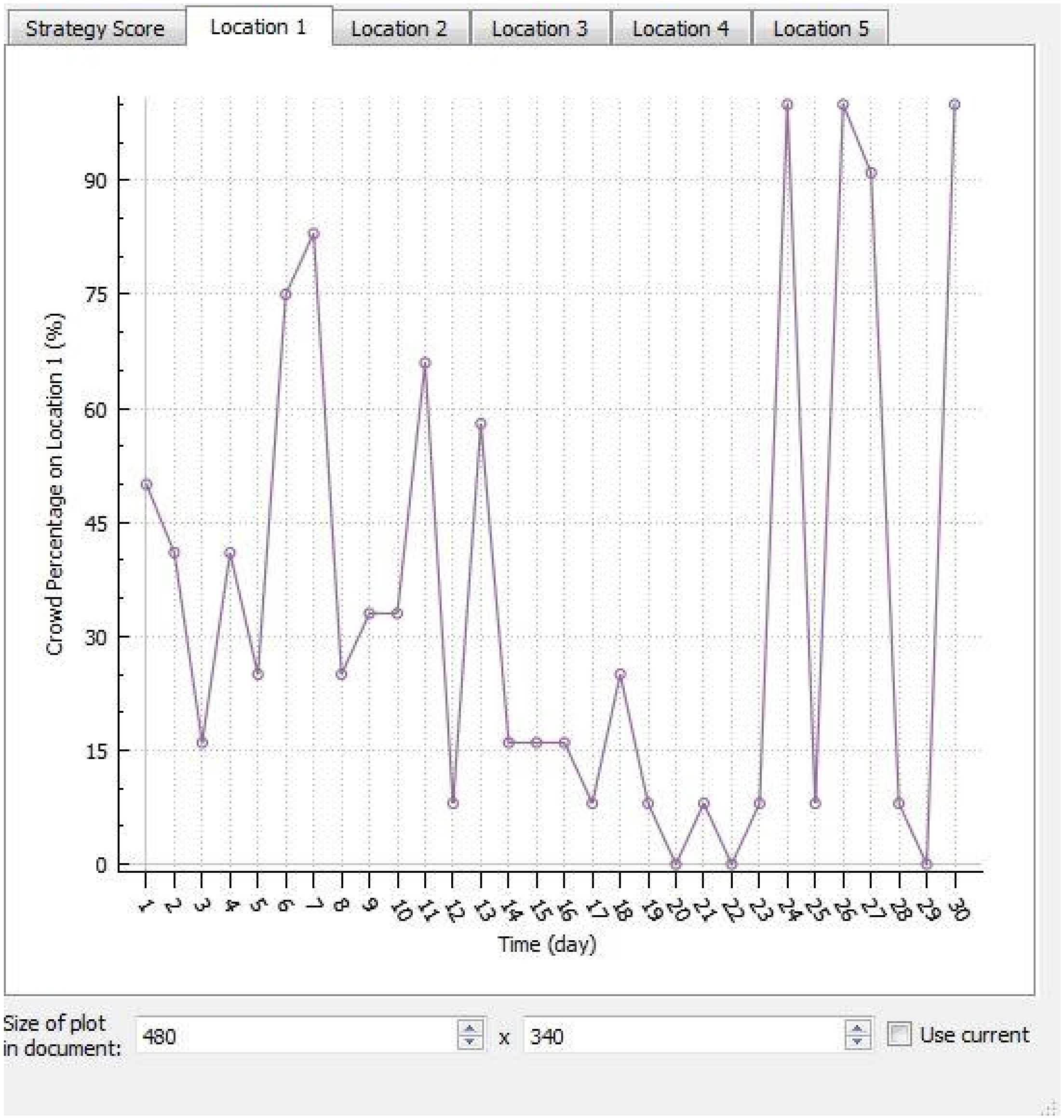
Fishers	64
Locations	5
Fish Population	697
Fish Types	5
Runtime	30
Weather	Sunny

Figure 11-4 strategy score

Location's graph:

If we change to the other tabs we can find a graph for each selected location. We can see a percentage of the people vs the days. We can have with this the percentage of people in each location and the day that this happened. With this we can "predict" what we could do to win the game.

If we have the total percentage of people that have gone to that specific location and also we have the number of people in the rest of the locations then the guessing task is simpler.



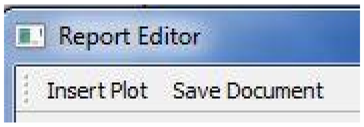
Size of plot in document: 480 x 340

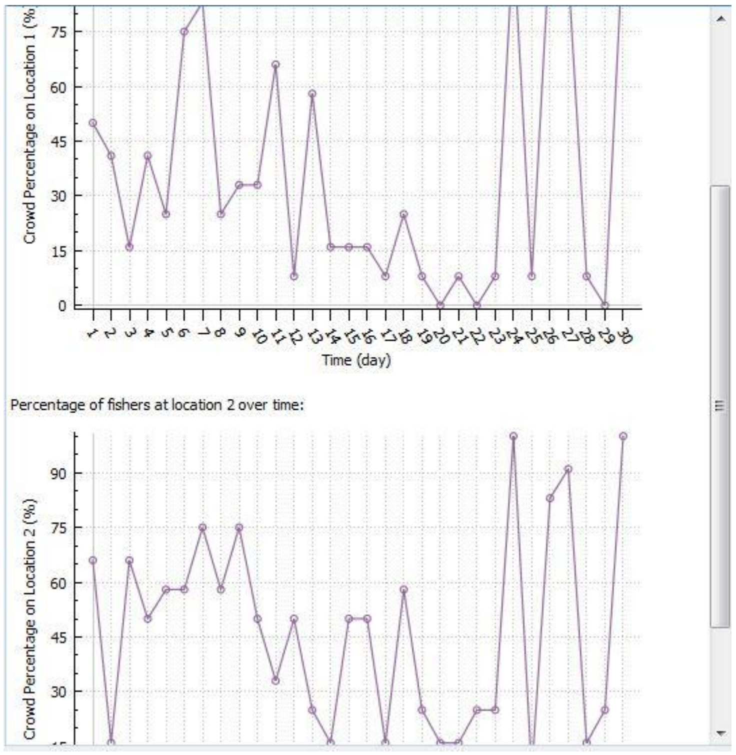
Figure 11-5 location analysis

Plots Comparison:

If we analyze the interface in more detail, we can observe that on the top-left corner we have two buttons (figure11.6). Insert plot and save document, the first button is going to let us do an easier and fast comparison of the plots by place them in the left side of our interface. And each time we change the location graph and we click insert plot, we are adding a new graph to the document on the left and with this we can compare them more comfortably.

Finally if we want to save the document, we just have to click “save document”, and a document with the variables followed by the graphs will be saved with the name and the location that we desire to stablish in a .PDF format (figure 11-8).

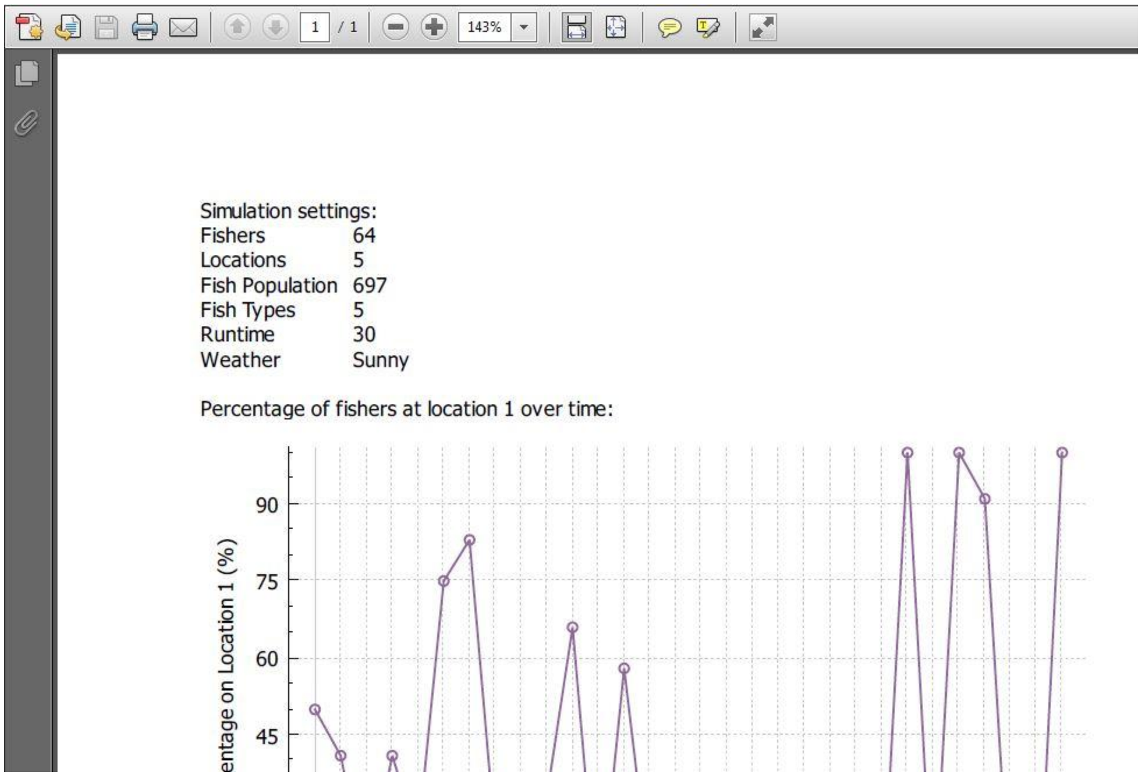




Crowd Percentage on Location 1 (%)
Time (day)
Percentage of fishers at location 2 over time:
Crowd Percentage on Location 2 (%)

Figure 11-6 buttons

Figure 11-7 location comparison


Simulation settings:

Fishers	64
Locations	5
Fish Population	697
Fish Types	5
Runtime	30
Weather	Sunny

Percentage of fishers at location 1 over time:

Figure 11-8 PDF document

13. Design of Tests

For the integration testing strategy, the "bottoms-up" strategy would be likely used. The "bottoms-up" integration starts from the lowest level of units to the highest level of units. In this case, testing the GUI functions would be important to do first. Making sure all the GUI buttons and sliders work and fully functional would provide a good foundation for the "bottoms-up" strategy. Next, another unit like the output units would be tested next. This include the simulation graphics and plots data. This can be tested again through GUI. Finally, the unit involving data retention could be tested next. It is important to check all saved data can be accessed again.

Testing the GUI is important since it is what the users would interact with. All the codes and algorithms inside the program, which are also important, will also be tested separately shown in detail in the documentation.

Testing on GUI:

Test Case ID: Sliders
Unit to Test: GUI input sliders
Assumptions: Program is running ready for inputs.
Test data: Chosen input data
Steps to be Executed:
a. Decide on specific inputs
b. Move the sliders to the chosen specific inputs
c. Watch as the values on the numerical box is changing according to inputs.
d. The values should stop if minimum or maximum limits are reached.
Expected Results: The inputs are changing according to slider's movements for each sliders and stops at minimum or maximum limits.
Pass or Fail: This test passes if the values changes as sliders move stopping at maximum and minimum limits. It fails if nothing happens to input value numbers or the numbers goes over the minimum or maximum limits when sliding.
Comments: Using slider eliminates the possibilities of giving invalid inputs.

Test Case ID: Simulate button
Unit to Test: GUI simulate button
Assumptions: Valid input data are chosen
Test data: Clicking simulate button
Steps to be Executed:
a. Press the simulate button.
b. Watch the output results via graphical real time simulation.

Expected Results: The program successfully give out results in the graphical real time simulation
Pass or Fail: The test pass if the program runs and fails if it does not start running.
Comments: This test makes sure that input data would start to process for desired outputs.

Test Case ID: Generate Report button
Unit to Test: GUI generate report button
Assumptions: Input data is already simulated via the simulate button.
Test data: Valid input data and simulated results.
Steps to be Executed:
a. Click the generate report button
b. Wait for another window to open
c New window contains different plots
Expected Results: Graphic plot representation appears on separate window from main GUI window.
Pass or Fail: The test passes when a window for graphic plots appears after generate report button is clicked. It fails if there are no appearance of the graph's window
Comments: This is different from the Plot display test below since this only specifically test the generate report button not the output results.

Test Case ID: Save button
Unit to Test: GUI save document button
Assumptions: Program is finished running and generate report button is already clicked
Test data: Save document button
Steps to be Executed:
- a. Press the save document button
- b. Enter the filename of newly saved data, then press okay
- c. (Optional) Do Test Case: Saved Data to check if data is saved.

Expected Results: The data is saved
Pass or Fail: This test passes if the data saved using the save button and fails if the nothing happens.

Test Case ID: Insert plot button
Unit to Test: GUI insert plot button
Assumptions: Program is finished running and generate report button is already clicked
Test data: Insert plot button
Steps to be Executed:
- d. Press the insert plot button.
- e. Watch as the plot is inserted in the report space.
- f. (Optional) Click another tab for another plot then press insert plot button.

Expected Results: Plot is inserted in the report space

Pass or Fail: This test passes if the plot is inserted in the report space when the button is clicked and fails if the nothing happens.

Test Case ID: Real Time Graphics
Unit to Test: GUI real time graphic output
Assumptions: Program is ready for inputs
Test data: Valid inputs in the sliders
Steps to be Executed:
a. Move the Locations slider
b. Watch as the number of graphic location circles appear with the same number on slider
c. Move the Fisher slider
d. Watch as the number of graphical people changes with the sliders
Additional:
e. Decide on specific input values
f. Click Simulate button and watch the number of people change as time goes by on different locations.
Expected Results: The graphical simulation reacts on the given inputs and simulates results that user can watch.
Pass or Fail: The test pass if the number of location and people graphics changes with input and if the values of people changes when simulation starts. It fails if the graphic visuals does not respond to changes and does not start running when Simulation button is clicked.
Comments: This test makes sure that graphical visuals responds to inputs and simulate the desired outputs that user can watch..

Test Case ID: Plot display
Unit to Test: GUI Output plots
Assumptions: The simulation button has been clicked.
Test data: Simulated result given by input data.
Steps to be Executed:

a. Click generate report button
b. Watch another window open with the plots
c. Access and check different plots via tabs on top of plot.
d. Check if plots have reasonable data for chosen input (there should be the same number of location as the input number of locations and one plot for strategy scores)

Expected Results: Graphic representation of the output as plots.
Pass or Fail: The test passes when there are graphic plots with the correct number of plots and the plot shows the right data. It fails if the number of location plots does not match number of input location.
Comments: It can also be checked if the output graphs are labeled correctly or the numerical data are in the right places (x or y axis).

Test on output results data:

Test Case ID: Output data
Unit to Test: Strategy Guide, Output Graphs
Assumptions: Valid input.
Test data: Input data.
Steps to be Executed:
a. Put random valid input through all strategy and agent functions to get decisions
b. Use decisions, strategy scores and agent data to get decisions of all agents
c. Calculate crowd percentage per location
d. Print out all calculated values
Expected Results: Strategy scores for all strategies and crowd percentage per spot.
Pass or Fail: It passes when strategy scores and crowd percentage are reasonable. It fails if strategy score and crowd calculations does not make sense (crowd percentage the same for all locations or all strategy scores are the same).
Comments: This test is connected to Plot display tests.

Test on saved data:

Test Case ID: Saved data
Unit to Test: Save document
Assumptions: PDF file saved.
Test data: PDF file.
Steps to be Executed:
a. Go to directory for saved document
b. Find the saved PDF file through given file name
c. Open up PDF and see if information are right
Expected Results: Saved PDF file will be opened.
Pass or Fail: Passed if saved PDF file is found and has the right contents and fails if there is no saved PDF file.
Comments: This is for saved reports in PDF form.

14. History of Work, Current Status, Future Work

When we started the Project we didn't know exactly how we wanted to focus the idea behind our actual program. However after some meetings we decided to create something different to our antecessors. For this reason we ended up choosing the idea of Fishing. And we start to develop the complete idea for our proposal using google drive to add more features that we thought could be helpful in the program.

Finally when we finished of collecting all the ideas we sent the proposal and we got our project accepted. From this point we started to work.

In the First report we didn't have to many experience about dividing the project. However, we decided to create another file with the different parts that must be covered for the end of the week and then each person of the group choose what he/she wanted to develop.

This probably was the first milestone of the group and the project, learning how to divide and contribute to the project.

In the first report we had to create the core of the project including the use cases that we were going to implement and even the interface that we were going to have. The creation of the first report was the second and probably last milestone that we achieve, because once this report was finished we had in front of us the look that our program was going to have and in as a first approximation the main use cases, variables…

Once we finished the first report, we got a complete synchronization with the deadlines and with the rest of the members in the group. Another tool that we used to increase the efficiency of the group in communication, it was the creation of a SMS group in order to ask for help, set time meetings or know how we were going to integrate and finish the work to submit it.

For the second report we updated some of the parts that we were going to develop and got a better approximation to the final version. And in order to complete this task we updated the first report.

Due to the fact that each week we had to finish a part of the project, we didn't have to change too many sections. For each report we had 3 parts (first report, second report and final report.) and due to this the general work wasn't as hard as if we had only one report without dividing it in 3 parts.

In the second report we went deeper in the software, we started to describe in detail each use case and create the diagrams. Other sections that we had to cover were the architecture, data types or hardware that we needed. With this we can see that we specified in more detail the project and concrete how it was going to be(data, hardware, features…)

In the first Demo, we showed how was the program working and three things that the professor and TA mentioned were that some of name of the use cases that we had weren't too clear because the name was a little confusing. The second thing that they mentioned was respect to the weather, because the type of weather wasn't clear and we needed to specify that if we don't change the weather in the GUI, it took sunny by default. The last thing was respect to the graphs, even though that we didn't finish that part we needed to make sure that when we finish it we had labels in each graph.

For the second Demo we made all the changes that the professor and the TAs mentioned and we finish the graphs that we didn't have in the first demo, the inputs visualization and some algorithmic improvements.

Current status

UC-1 SetInitial

UC-2 SetConditions

UC-3 RunSimulation

UC-4 SaveData

UC-6 ShowResults

UC-7 ProvideStrategy

UC-9 ProvideHelp

- Implemented GUI
- Implemented the functions that run the simulation, and generate the report.
- Implemented the option of adding plots and saving the document.
- Improved algorithms to calculate the strategy score (we used to have negative numbers in the graph, now it has been modified to have positive numbers and be more straightforward to understand) and to calculate each location.

Future work:

Right now or program is using random values to obtain the final data, for this reason a modification that we would like to add, is do an analysis of a real park to see if the data that we had matches with the data that we obtain in the research and make our calculation more accurate in order to have more real results. Another change that we would like to add is make it more global, in other words, have in the beginning of the program an option to choose if you are a business user or a personal user, in the case of the business user, we would keep the plots and everything that we have right now in the program and in case of being a personal user we would make it fast for the person, instead of show graphs or data, we would just give him/her what they want to know, what they should do.

15. References

Full Report:

1. Marsic, Ivan. "Object-Oriented Software Engineering." Software Engineering. 2012. 110-113. Print.

2. Marsic, Ivan. "Software Measurement and Estimation." Software Engineering. 2012. 222-230. Print.

Report 1:

1. "El Farol Bar Problem." Wikipedia. Wikimedia Foundation, n.d. Web. 08 Feb. 2015.
2. "The Minority Game: An Introductory Guide." Implicit None. N.p., 31 Aug. 2004. Web. 08 Feb. 2015.
3. "Concepts: Requirements" UPEDU. Polytechnique Montreal, 2014. Web. 07 February 2015.

Report 2:

1. "Sequence Diagram." *Wikipedia*. Wikimedia Foundation, n.d. Web. 01 Mar. 2015.

2. "Architectural pattern." *Wikipedia.* Wikimedia Foundation,n.d. Web. 07 Mar. 2015.

3. Chou, David. "Using Events in Highly Distributed Architectures." *Architecture Journal.* Microsoft. 2008. Web. 07 Mar. 2015.

4. "Pattern: Monolithic Architecture." *Microservice architecture.* Chris Richardson. 2014. Web. 07 Mar. 2015.

5. Qt Application Framework - http://qt-project.org/