# Simulation of Minority Game

**Group #4**

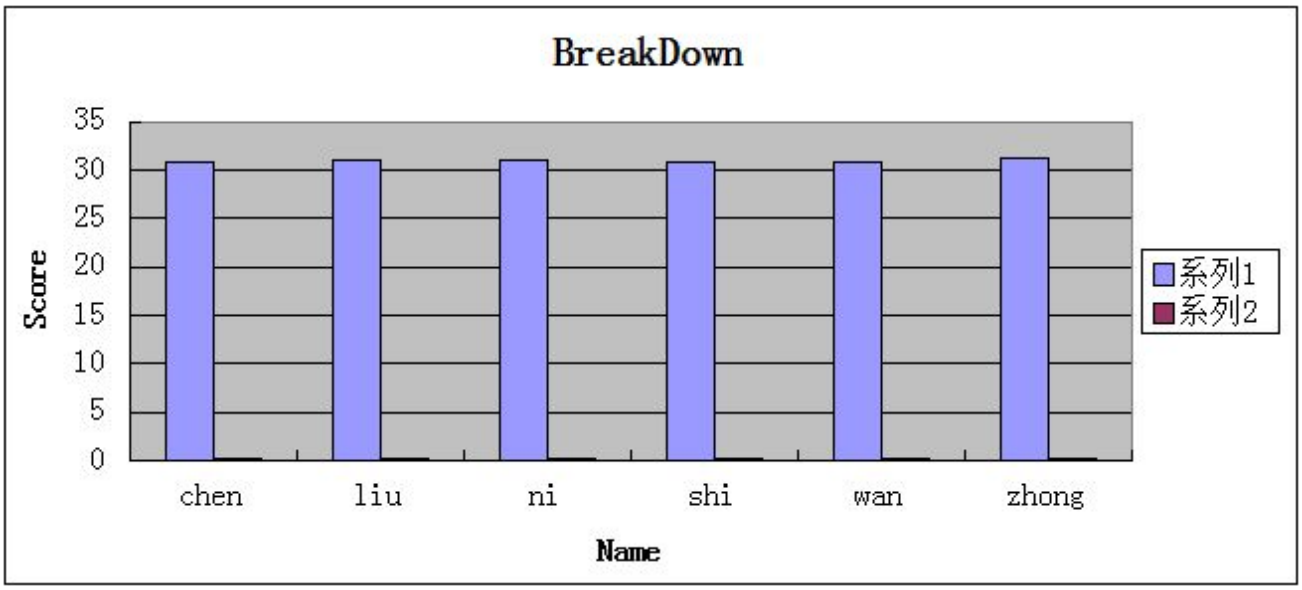

Group Members:

| | |
|---|---|
| Zhan Chen | johnalwaysyoung@gmail.com |
| Xiaoheng Liu | fsr0023120@gmail.com |
| Boyu Ni | niboyu@live.com |
| Pengcheng Wan | kevinwan1991@hotmail.com |
| Jinhe Shi | jhshi@hotmail.com |
| Zhengyang Zhong | timothy7784@gmail.com |


Final Report 2013/Dec/09

**Contribution Breakdown**

| | point | Team Member Name | | | | | |
|---|---|---|---|---|---|---|---|
| | | **Zhan Chen** | **Xiaoheng Liu** | **Boyu Ni** | **Pengcheng Wan** | **Jinhe Shi** | **Zhengyang Zhong** |
| Project management | 10 | | **20%** | **40%** | | | **40%** |
| CSR | 9 | **50%** | | | **50%** | | |
| System Requirements | 6 | | **16.7%** | **66%** | **16.7%** | | |
| Functional Requirements Specification | 30 | **40%** | **40%** | **20%** | | | |
| User Interface Specs | 15 | | | | | | **100%** |
| Domain Analysis | 25 | | | **10%** | **40%** | **50%** | |
| Plan of Work | 5 | | | **100%** | | | |
| Interaction Diagram | 30 | | **20%** | | **30%** | **40%** | **10%** |
| Class Dia & Interface Spec | 10 | | | **35%** | | **20%** | **45%** |
| Sys Arch & Design | 15 | | **33.3%** | **33.3%** | | | **33.3%** |
| Algs's & Data Structure | 4 | **25%** | **50%** | **%25** | | | |
| User Interface | 11 | **50%** | **30%** | | **20%** | | |
| Testing Design | 12 | **66%** | | | | **29%** | **%5** |




| Name | Score | Percentage |
|---|---|---|
| Chen: | 30.9 | (16.6%) |
| Liu: | 31.0 | (16.7%) |
| Ni: | 31.0 | (16.7%) |
| Shi: | 30. 8 | (16.6%) |
| Wan: | 30.9 | (16.6%) |
| Zhong: | 31.3 | (16.7%) |

# Table of Contents

# Summary of changes

In this edition, we want our software to be closer to our real life, especially in Stock market. Thus, we focused mainly on developing the new function named “herding” and “broadcasting”. What’s more, the non-function part of FURPS+ have tremendous changes to make our project more usable and reliable. The detail of changes are below.

* Herding Function：

Some of the Agents in the project can Automatically become several groups, group members can share their decision to others and help others make a better decision.

* Add new feature, Score of Agents:

Every Agents will have their own scores. Every time they win a game, score plus one. If lose, minus one.

* Add a New agent, named “Richest guy”:

The Agent with the highest score named the Richest Guy, this agent has a special function, he can broadcast to all the agents his decision.

* Add new function，Broadcast:

The function of the “Richest Guy”, he can give his decision to all the agents. He will not broadcast decisions every round, maybe only 20% ratio to broadcast.

* Add new function，Cheating :

It is a small function in of the Richest guy, when he is broadcasting his decision, there is about 20% probability that he will cheat other agents. It means that the decision he broadcast to other agent is in fact not his true decision.

* Optimizing the Algorithm when making decision

In former edition, Agents sometimes make wrong decisions and these agents will keep losing game. We optimize this bug.

* Beatify the GUI

Do more things, like using PhotoShop to edit the pictures and icons of our software, it make it more comfortable to use.

* Add new diagrams in the “diagram slide”

Can show the dead ratio and the highest score of all agents

* Remove the “number of bars” option in the first GUI

More bars can not increase the precision of our project, so we remove this feature.

* Remove diagrams of the diagram slide of the second GUI

Remove “number of bars that won”, and “Number of death”

* Optimizing the algorithm of “Speed change” button

In the former edition, speed change algorithm is not so good since when user changed very little speed the actual speed changed a lot.

# 1. Customer Statement of Requirements

## 1.1 Problem Statement

There has been a time that people are confused about how to make the right decision. Thanks to quick developing of the technology, now we have computers to help us simulate the situation of real world to solve our problems. Through simulation, it is not difficult to figure out what decision should be made in certain situations.

In order to make it convenient for us to make decisions in various circumstances, we need software which is capable of simulating the situations. Thus, we can easily deal with our problems just by adjusting some parameters. In other words, by clicking the mouse (adjusting the parameters) we can simulate the environment we are involved in, so we can get the result in advance.

The system should be mainly designed for finance field, but it should be used in other cases as well.

As for the us, by using this software, for example, we are actually shown a film or watching a documentary film about what's happening for a simulated market and experience various results that different parameters I choose bringing about.

In fact, this is a simulated market presented in front of us, and it is us, the customers to decide what the basic factors in the market are. A certain numbers of agents will come to buy or sell a stock. And the winners will certainly, be the minority. To simulate the real market, we need to set that the money lost by the majority will be redistributed to the winners as only the minority parcel social property. So, strategies for choosing to buy or sell at each time become of great significance, and this is the reason we need this software as well. Since we only see the choice and result of a single agent, we want someone to make the algorithms and useful methods.

Literally, we want the model to be even more like the real world, we want to see that every choice has its own impact to the next round. And we want information to be stored in each agent's memory. On one hand, we give some equal numbers of money to every agent, and drop the one who

lost all his/her money; on the other hand, we will do some research of the person who gains more money than others.

However, this is not the end, as the situation can be more complex in the real world. We will find out that agents are divided by their characteristics , so there must be some excellent ones who win frequently, while noise agents (individuals gaining profit by following the herd) also exist. This will result in more complex situation, to better reflect the real competition, the herding will be a common sense for almost every one, and experts will be favored for herding.

Since we have a good interest in social relations, we may further see some experts cheating the others so that they will get the largest share of the benefits. Through running again and again, the final results will be shown for us to analyze the situation we are involved in and help us get the conclusion.

We hope this software can be applied on market trading, though some details are not exactly as the real financial market, but it should be a fairly good reference as a matter of fact. In other words, it should show relatively reasonable strategy for deciding whether to buy or sell.

To better describe our needs, at least 5 main parameters should be involved: short-term memory, long-term memory, scores of agents, life duration and herding.

We need this software to calculate 2 types of score: Agent score and strategy score. Agent score is used to estimate the successful rate of each agent. Strategy score is used to estimate the quality of the strategy.

The program should allow us to customize the memory of each member consists of long-term memory and short-term memory.

Short-term memory: The agents should be able to memorize the game results of previous rounds and make decision according to these results.

Long-term memory: Each agent should have several strategies and scores them in its long term

memory. All strategies have a initial score and after every round, the strategies' score should add or subtract according to the result of the game. Thus, each agent can keep a running tally of each strategy's score in comparison to other strategies. The member will make decision depends on these "scores" to choose the strategy. The program should output the strategy with highest score and the agent who has the highest success rate. After several rounds, the strategy with bad performance should be excluded by the agent. In order to better simulate the real-world situation, agents should have different memory abilities.

The program should also include life duration (mortality) in the game, which will make members stop participating in the game when they die. Every agent will be given an age value randomly. The whole population of these dead people will be replaced by a whole younger generation. This is consistent with the law of nature and life. Furthermore, when they die, part of their scores, or capital, should be distributed to the entire system, the other will be inherited by younger generation.

Converging crowd causes group effect. As a consequence, different and complex situations come into being. Generally, there are two situations: one is that people have their own fixed group and the agents of that group do not change; the other is that the agents of the group change with time owing to the historical records of others in the same group: agents with bad performance will be dropped out of the group while the relatively competitive ones stay, nevertheless, while someone gets out, there should be new group member added, simply make the total numbers constant.

In the situation that the agents of the group sway, we need a role for agents called advancer, with a definition of people who win a lot in this sequence of the game. These roles sometimes cheat in their own group so that they can obtain benefits of their own.

Moreover, we need a role called giant who ranked among the top. This role is endowed two behaviors: cheating and broadcasting. Besides cheating, giants broadcast to influence other people. All the behaviors above are purposed for minimizing the income of the rest and gaining the most benefits for themselves (sometimes they may guide other agents to win the game in order to acquire their trusts, but sometimes they prefer to minimize the numbers of winners so as to maximize their own benefits). In such situation, agents should be able to keep track of other agents' credit. If a

certain agent cheats others for many times, its advice will be of less importance in other agents' decision making process.

### 1.2 Glossary of Terms

To better illustrate the contents, we list some important terms and our system graph below:

**Scoring rules -** It was introduced to help the customers to judge the gain and loss of the agents.

**Memory effect -** Every agent has its own memory length, like the situation in the real world that different people have different abilities of memorizing.

**Life duration -** Because in the real world, people have different life length, we introduce this parameter.

**Herding effect -** It is for simulating more complicated situations which is to simulate the behaviors of people who make their decision by referring other people's decision. This kind of behaviors especially exits in financial market.

**Cheating -** In order to gain the maximum profits of their own, the role advancer (including someone who ranks the top in this game) sometimes provide wrong information to their own group.

**Broadcasting -** The agents ranked among the top in this game are privileged to broadcast their advice to influence other agents' decision. This behavior is often related with cheating, since the ultimate goal of the top agents is to maximize their own benefits.

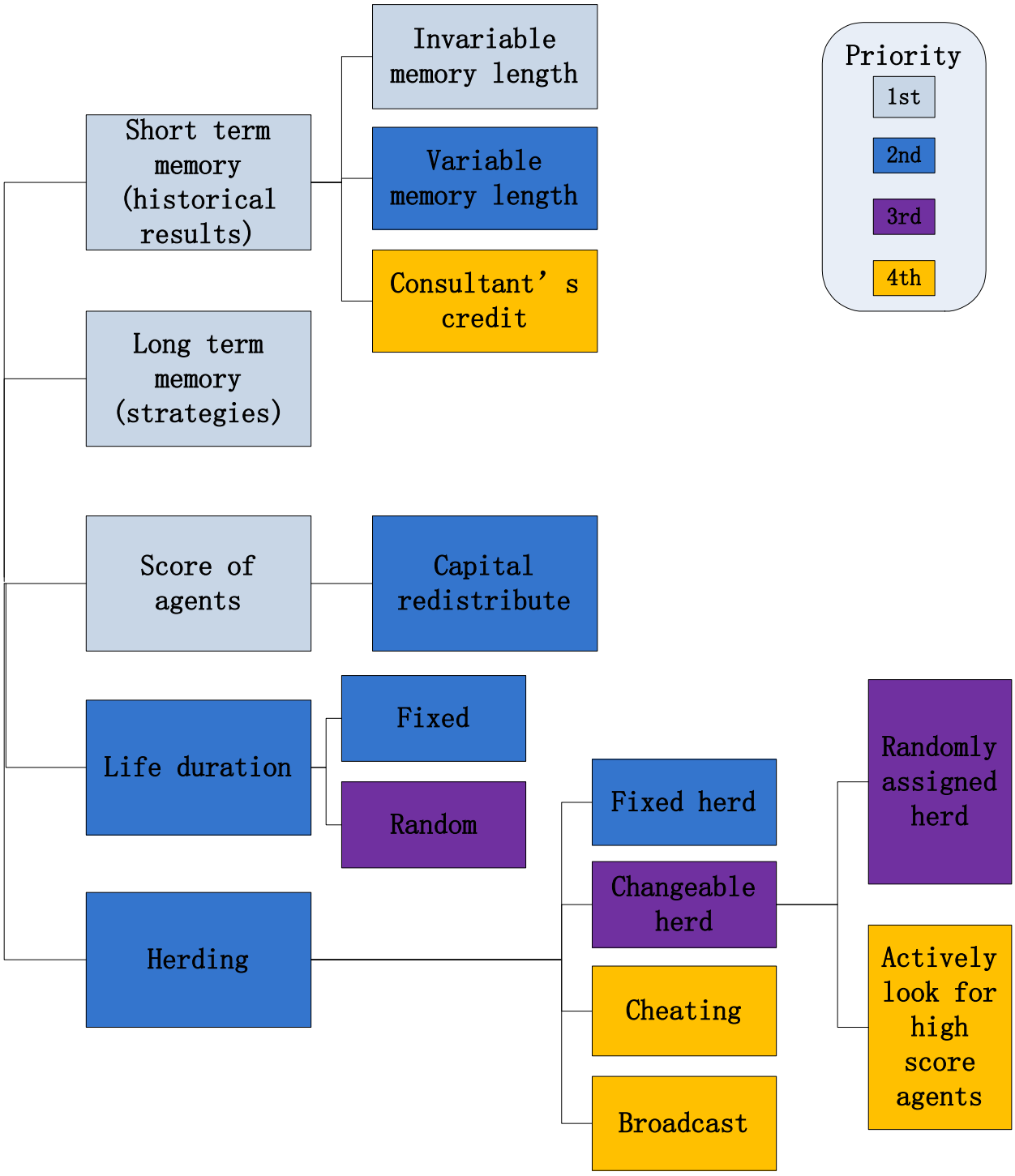



Fig 1 Flow sheet of parameters and priority

# 2. System Requirements

## 2.1 Enumerated Functional Requirements

| Identifier | Priority weight(Low1-5High) | Requirement Description |
|---|---|---|
| REQ-1 | 4 | Users should be able to choose whether or not to include mortality in this game |
| REQ-2 | 5 | Agents with the minority decision win the round. Those who choose the majority lose that round. |
| REQ-3 | 4 | Users should be able to choose whether or not to include memory in the game |
| REQ-4 | 3 | Agents should have multiply strategies to use when making decisions each round and they choose each strategy randomly from strategies space (long-term memory). |
| REQ-5 | 3 | High scoring strategies will be reused and low scoring strategies will be dropped |
| REQ-6 | 2 | When agents die, their scores will be distributed to the system and inherited by younger generation(<br><br>if mortality option chosen) |

| | | |
|---|---|---|
| REQ-7 | 5 | User should be able to set all initial conditions |
| REQ-8 | 3 | Strategy should keep score to estimate the quality of each strategy |
| REQ-9 | 3 | Agent should keep score to determine the successful rate of each agent. |
| REQ-10 | 3 | Agent also should be given a credit score to estimate the credit degree of each agent |
| REQ-11 | 2 | Agents should be able to retain memory of previous rounds' scores and make next decision depend on the previous scores (short-term memory). |
| REQ-12 | 4 | User can select whether or not to include herding in the game |
| REQ-13 | 3 | Agents should be able to choose to join a group to discuss and make decisions together(if herding option chosen) |
| REQ-14 | 3 | Low score agents will be dropped out of the group, and new agents will be added. (if herding option chosen) |
| REQ-15 | 2 | Agent who wins the most will be the advancer in the group and may cheat in their own group to obtain benefit (credit score). |
| REQ-16 | 1 | Users must be able to change speed while simulation is running. |

| | | |
|---|---|---|
| REQ-17 | 2 | The program can be used in various circumstances (trading market) |
| REQ-18 | 5 | Each agent will be given equal number of scores in the game. |
| REQ-19 | 1 | User must have access to data collected in the form of graphs. |
| REQ-20 | 3 | The length of short-term memory can be variable depend on different people. |
| REQ-21 | 5 | User can decide the numbers of agents and rounds. |
| REQ-22 | 3 | The user can decide life length of the agents(random or fixed) |
| REQ-23 | 3 | Agents whose scores are less than m will be dropped out of the game and new people will be added. |
| REQ-24 | 3 | User can choose whether the herding is fixed or changeable. |
| REQ-25 | 3 | Agents will look for those who have high scores to be a group. |
| REQ-26 | 1 | The program should keep running until the user feel they have enough data. |
| REQ-27 | 5 | The totally number of agents in the game is constant. |

| | | |
|---|---|---|
| REQ-28 | 2 | The giants' strategy will broadcast to influence others |

## 2.2 Enumerated non-functional requirement

A model called FURPS+ will be used here to qualify software attributes, which stand for functionality, usability, reliability, performance, supportability, and the + stands for other possible attributes needed. We will be focusing on the non-functional requirements which cover FURPS+.

**Functionality**: The functionality of our project if one of the most essential aspect. And our software will contain numerous functions and parameters which would help to solve problems in different situation. The logic level of the whole system, however, will be two as maximum. So the system will be characterized by multi-functionality and usability.

**Usability**: The logic layers of the system will be no more than two. Customers only need to select situation and add parameters to the simulation. We will do our best to minimize the mouse pointing time and maximize the function intuitionism.

**Reliability**: Frequency of failure should be very low. Customers only need to restart the software to recover and are able to choose whether to recover the last step they did. And software will be updated every month if not never.

**Performance**: The whole system also has high performance. Customers would wait no more than 5 minutes during the whole simulation procedures. And the running time, which depends on the parameters that chose by customers, will be few seconds as minimum and no more than 1 minute as maximum.

**Supportability**: The system is easy to understand by every user and programmer. There will be a user document to introduce how to use the program and an introduction demo will also be contained to the software. A technique support via E-mail will also be available and primarily to deal with the bug and imperfect part.

## 2.3 On-Screen Appearance Requirements

| Identifier | Priority weight(Low1-5High) | Requirement Description |
|---|---|---|
| REQ-29 | 2 | Help-button：A “help” button should on the top right corner next to the “close” button. A help document will appear after touching, every detail like the glossary, graphs and button use will be explained. |
| REQ-30 | 5 | Error checking：The user interface has error checking function for all inputs. It will indicate what the error is to customers and how to revise it. |
| REQ-31 | 3 | Range checking：not like error checking, a large range can still be operated, but will spend a lot of time sometimes even lead to computer crash. So we should warn the customer when the range is too big. |
| REQ-32 | 3 | User-friendly：The operation of the user interface should be as easy to operate, no specialized training is need for new customers. |
| REQ-33 | 1 | Aesthetic value：Nowadays, we live in the Market-Economy situation, a great application should not only pay attention to the function, but also to the aesthetic value. Thus our user may spend more time on the app. |
| REQ-34 | 1 | Flexible：the user interface should be convenient for customers so it should be removable and can be |

| | | |
|---|---|---|
| | | amplified and lessened. |

# 3. Functional requirements

## 3.1 Financial professionals and related personnel

This software is mainly designed for people participating in financial market. It can also be used for other purpose, considering we provide many parameters into the software.

In the financial market, people make decision according to previous data, so we made this character the basis of the software. Briefly, in this software, all agents make decision according to the past results of others and themselves. We introduce the concept “short-term memory” and “long-term memory”. Short-term memory refers to the memory of historical results. In this game, users can choose let the agents have variable or invariable short-term memory. Long-term memory relates to strategies memory. With long-term memory, agents make decisions according to the success rate of their past strategies.

In order to make this software more similar to real world situation, we introduce some other parameters. In the real world, people have life duration, so we introduce “mortality” in this software. Considering the variation of people’s life duration, we add this choice in this software. User can chose whether life duration varies among agents. What’s more, in the financial market, agents tend to communicate and exchange information before making decision. So we add the choice “herding”. User can select this item to simulate the situation in the market that some people in this market take others’ advices to make their own decisions.

Besides financial market, this software can be used in many other cases. In situations concerned with people and decision, this software is a useful way to help simulate the situation and analyze the decision accordingly. Airport, super mall and even the policy makings can use this software to simulate. Manufacturers can make decisions with this software to better participate in the market. Policy makers use this software to analyze whether the legislation can be passed. In fact, since there are so many parameters in this software, users can easily simulate many kinds of situations they need.

## 3.2 Actors and goals

| Actor | Actor goals | Use case name |
|---|---|---|
| **User** | Set initial condition for the simulation | SetInitialConditions (UC-1) |
| **User** | Run simulation | RunSimulation (UC-2) |
| **User** | Change graphs | ChangeGranph(UC-3) |
| **User** | Change speed | ChangeSpeed(UC-4) |
| **User** | Stop simulation | StopSimulation(UC-5) |
| **User** | Show graph using data log of past simulation | ShowPastData(UC-6) |

## 3.3 Use cases

| **UC-1** | **SetInitialConditions** |
|---|---|
| **Related requirements** | Req-1,Req-2,Req-7,req-8,req-11,req-12,req-16,req-17,req-18,req-19, |
| **Initiating actor** | User |
| **Actor's goal** | To set initial conditions for the simulation |
| **Participating actors** | System, User |

| | |
|---|---|
| **Preconditions** | Initial screen is showing |
| **Post conditions** | Initial conditions are set and Simulation is ready to start |
| **Flow of events for main success scenario** | |
| → | User (a) selects the menu item “long memory” (b) types in value |
| → | User (a) selects the menu item “short memory” (b) types in value |
| → | User (a) selects the menu item “score of agents” (b) types in value |
| → | User (a) chooses to include or not include “life duration” |
| → | User (a) chooses to include or not include “herding” |
| → | User chooses name of output file |
| ← | System verifies all values sets them in the Simulation |
| **Flow of Events for Alternate Scenarios:** | |
| → | 5.a.User chooses to include “mortality” |

| | |
|---|---|
| → | User (a) selects the menu item “life duration” (b) types in value |
| → | 6a. User chooses to include “herding” |
| → | User (a) selects the menu item “herding” (b) types in value |

| **UC-2** | **RunSimulation** |
|---|---|
| **Related requirements** | Req-1,Req-2,req-6,Req-7,req-8,req-11,req-16, req-19,req-24,req-25 |
| **Initiating actor** | User |
| **Actor’s goal** | To run a Simulation |
| **Participating actors** | System |
| **Preconditions** | Initial conditions have been set |
| **Post conditions** | User believes that they have gathered enough data |
| | Extends :: SetInitialConditions Includes :: SetSpeed, ChangeGraphs |
| **Flow of Events for Main Success Scenario:** | |

| | |
|---|---|
| → | 1.User chooses the button “Start Simulation” |
| → | 2.User chooses the initial graphs to be shown |
| ← | 3.System generates the specified data and shows user selected Graphs |
| → | 4.Graphs and speed may change based on user preference |
| → | 5.User hits the “Stop Simulation” button |

| **UC-3** | **ChangingGraphs** |
|---|---|
| **Related requirements** | REQ-3 REQ-9, REQ-17, REQ-24, |
| **Initiating actor** | User |
| **Actor’s goal** | Change the graph form while simulation is on. |
| **Participating actors** | System |
| **Preconditions** | 1.Simulation is on<br><br>2.User wants to change graphs on the screen. |

| | |
|---|---|
| **Post conditions** | 1.Show new graphs.<br><br>2.The simulation is still on. |
| **Flow of events** | |
| → | User chooses a new graph. |
| → | User chooses how many post rounds they want to show in the new graphs. |
| ← | System changes the current graphs into the chosen one. |

| **UC-4** | **SetSpeed** |
|---|---|
| **Related requirements** | REQ-15 |
| **Initiating actor** | User |
| **Actor's goal** | Change the speed of the system. |
| **Participating actors** | System |
| **Preconditions** | 1.Simulation is on.<br><br>2.User wants to change the speed of the system. |
| **Post conditions** | 1. Speed is changed |

| | |
|---|---|
| | 2. The simulation is still on. |
| **Flow of events** | |
| → | 1. User chooses a new operating speed. |
| ← | 2. System changes the currently speed into the chosen one. |

| **UC-5** | **StopSimulation** |
|---|---|
| **Related requirements** | REQ-24 |
| **Initiating actor** | User |
| **Actor's goal** | To stop the simulation and generate output file |
| **Participating actors** | System |
| **Preconditions** | Simulation is running and User wants to stop it |
| **Post conditions** | Simulation has been stopped, log file has been output and Simulation is ready to run again. |
| | Extends :: RunSimulation |

| **Flow of Events for Main Success Scenario:** | |
|---|---|
| → | 1. User presses button “Stop Simulation” |
| ← | 2. System finishes updating data to log file |
| ← | 4. System returns to Initial Screen for another run |

| **UC-6** | **ShowPastSimulation** |
|---|---|
| **Related requirements** | REQ-17, REQ-27 |
| **Initiating actor** | User |
| **Actor’s goal** | To see graphs from past Simulation |
| **Participating actors** | System |
| **Preconditions** | A past Simulation has finished and User wishes to review it |
| **Post conditions** | User has reviewed past Simulation |
| **Flow of Events for Main Success Scenario:** | |
| → | 1. User presses “read file” button |
| → | 2. User selects a log file |

| | |
|---|---|
| ← | 3. System retrieves data from the file |
| → | 4. User chooses graphs and number of turns that they care about |
| ← | 5. System Shows the requested graphs |

### 3.3.1 Use Case Diagram

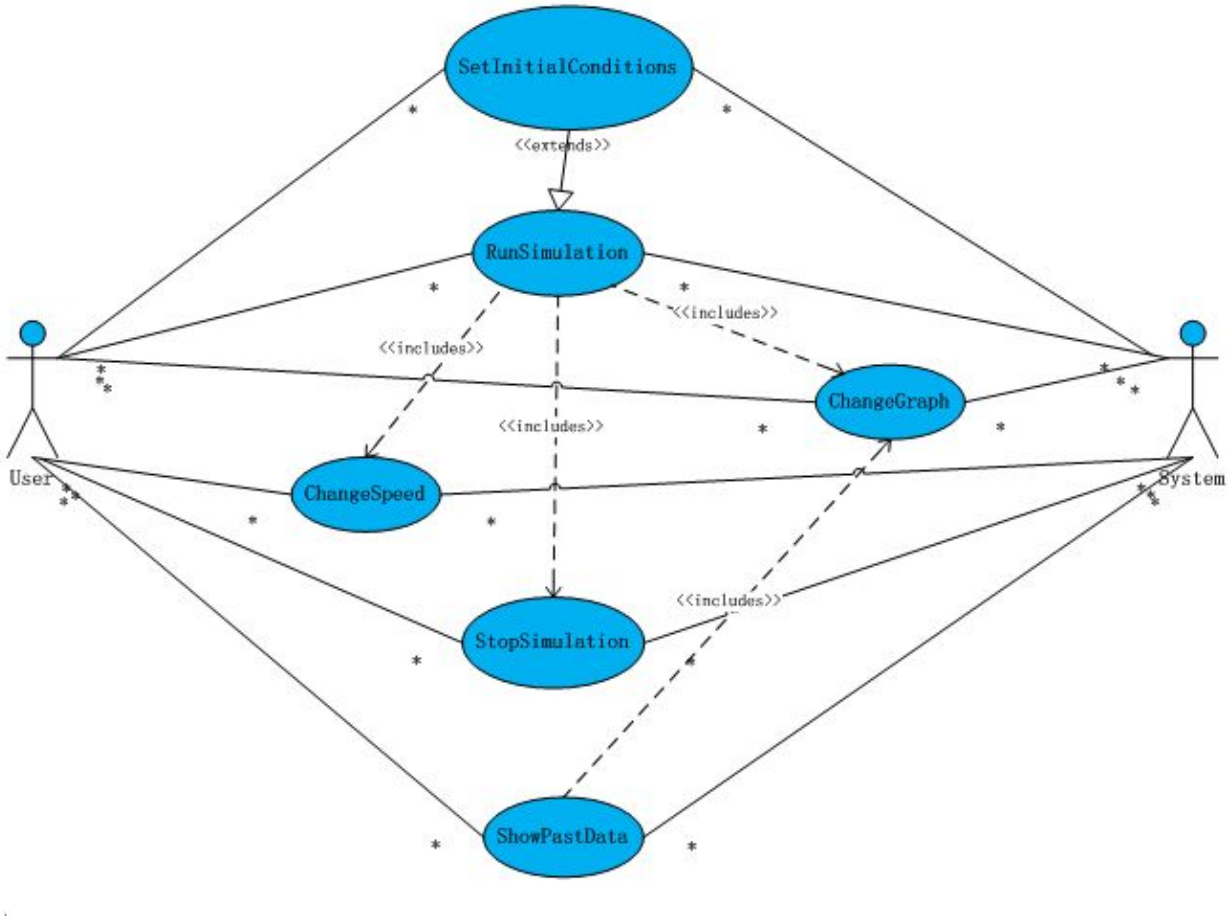



Fig 2 Use Case Diagram

### 3.3.2 Traceable Matrix

| Traceability VS Requirement | UC-1 | UC-2 | UC-3 | UC-4 | UC-5 | UC-6 |
|---|---|---|---|---|---|---|
| REQ-1 | × | × | | | | |
| REQ-2 | × | × | | | | |
| REQ-3 | | | × | | | |
| REQ-4 | | | | | | |
| REQ-5 | | | | | | |
| REQ-6 | | × | | | | |
| REQ-7 | × | × | | | | |
| REQ-8 | × | × | | | | |
| REQ-9 | | | × | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| REQ-10 | | | | | | |
| REQ-11 | × | × | | | | |
| REQ-12 | × | | | | | |
| REQ-13 | | | | | | |
| REQ-14 | | | | | | |
| REQ-15 | | | | × | | |
| REQ-16 | × | × | | | | |
| REQ-17 | × | | × | | | × |
| REQ-18 | × | | | | | |
| REQ-19 | × | × | | | | |
| REQ-20 | | | | | | |
| REQ-21 | | | | | | |
| REQ-22 | | | | | | |
| REQ-23 | | | | | | |
| REQ-24 | | × | × | | × | |
| REQ-25 | | × | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| REQ-26 | | | | | | |
| REQ-27 | | | | | | × |

## 3.4 System Sequence Diagrams

Uc-1:SetInitialConditions

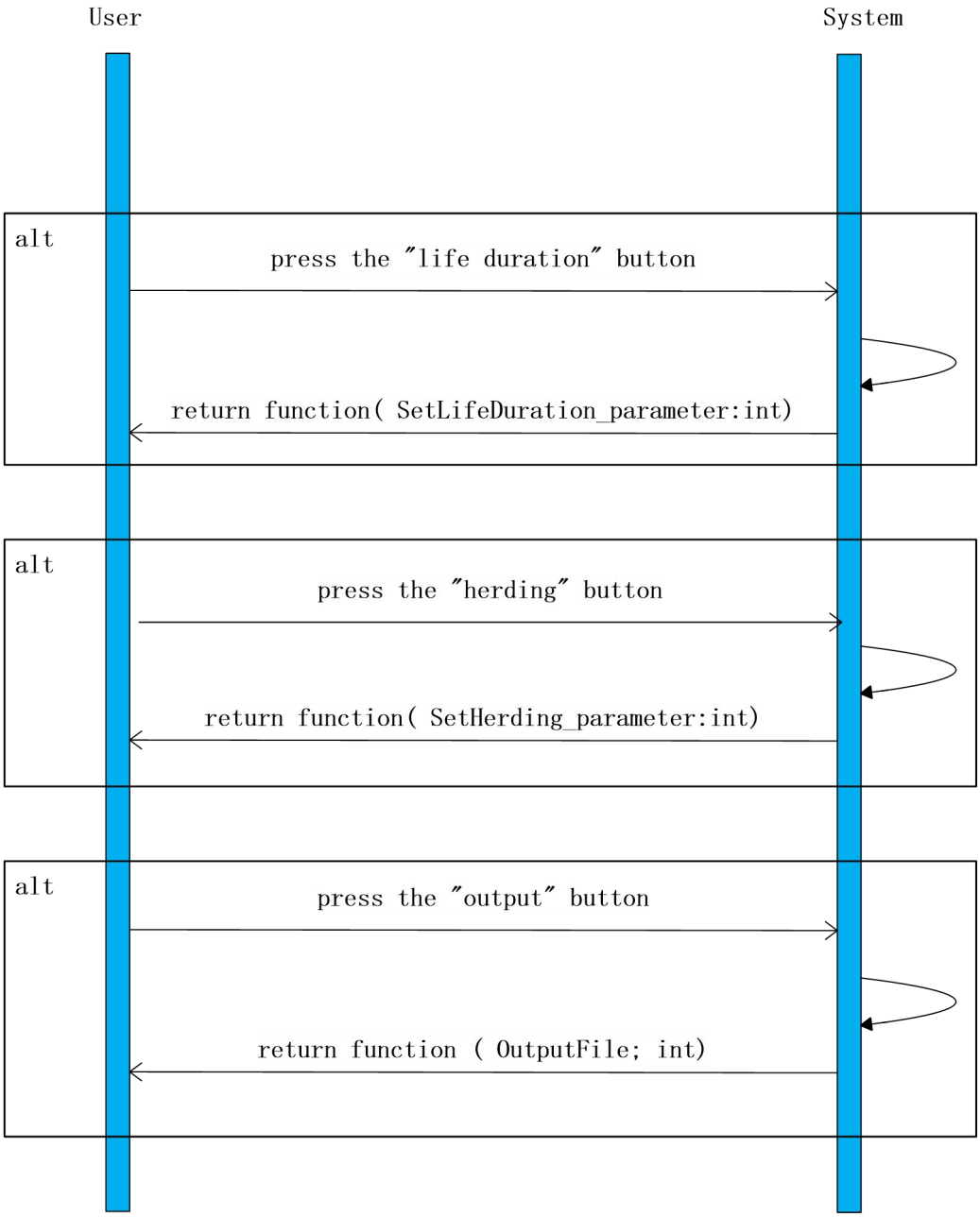


Fig3 SetInitialConditions

UC-2: RunSimulation

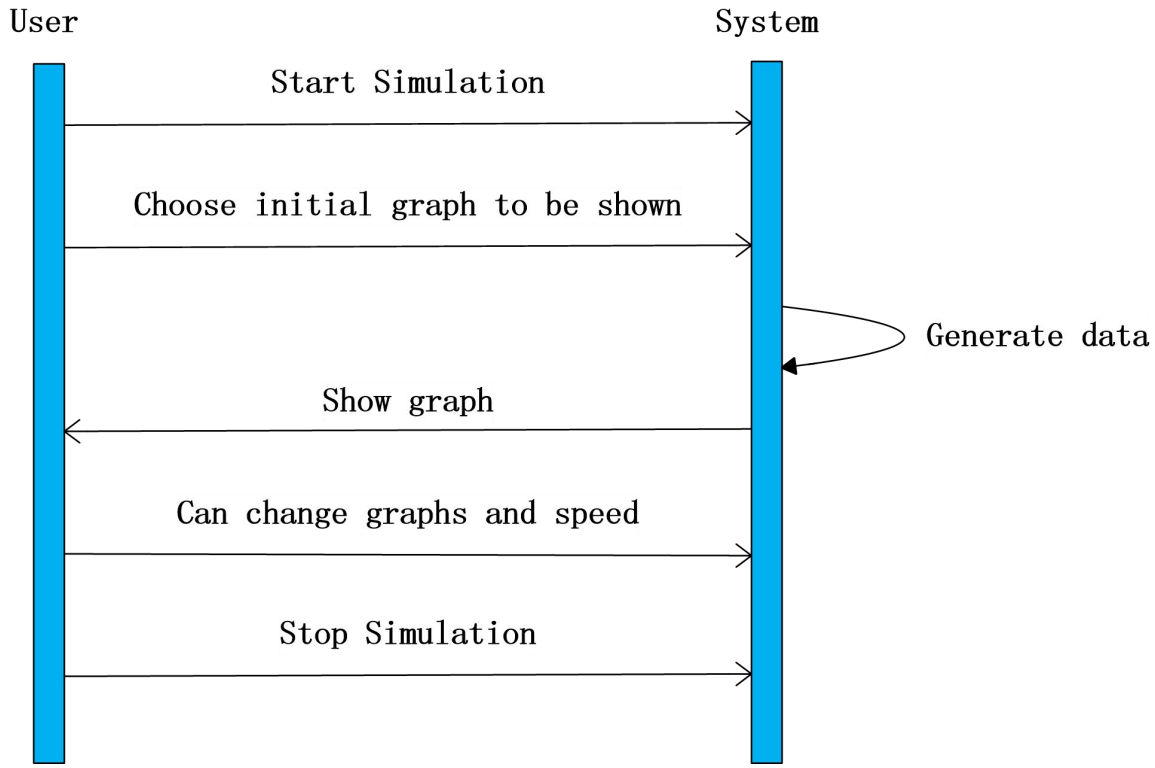



Fig 4 RunSimulation

# 4. User Interface Specification

**Effort Estimation Using Use Case Points**

**Standard Equations:**

Duration=UCP+PF

UCP= UUCP*TCF*ECF

UUCP=UAW+UUCW

**Definition**

Duration: the whole hours we need for a project

UCP: Use Case Point

PF: Productivity Factor

UUCP:Unadjusted UCP

TCF:Technical Complexity Factor

ECF: Environmental Complexity Factor

UAW:Unadjusted Actor Weight

UUCW: Unadjusted Use Case Weight

**1. UAW:**

| | |
|---|---|
| Simply | 1 |
| Average | 2 |
| Complex | 3 |

| | |
|---|---|
| User actor | 3 |
| System | 3 |

So our UAW = 6

**2. UUCW**

| | |
|---|---|
| Simple | 5 |
| Average | 10 |
| Complex | 15 |

| | |
|---|---|
| UC-1: Set Initial Condition | 5 |
| UC-2: Run Simulation | 15 |

| | |
|---|---|
| UC-3: Change Graphs | 15 |
| UC-4: Change Speed | 10 |
| UC-5: Stop Simulation | 5 |
| UC-6: Show past Simulation | 15 |

So our UUCW = 65

**3. TCF**

TCF = constant1 + constant2*calculated factor

Constant1 = 0.6 (according to the textbook)

Constant2 = 0.01 (according to the text book)

| Technical Factor | Description | Weight | Perceived complexity | Calculated Factor |
|---|---|---|---|---|
| T1 | Distributed system | 2 | 0 | 0 |
| T2 | Performance objective | 2 | 5 | 10 |
| T3 | End user efficiency | 1 | 4 | 4 |
| T4 | Complex internal processing | 1 | 5 | 5 |

| | | | | |
|---|---|---|---|---|
| T5 | Reusable design or code | 1 | 2 | 2 |
| T6 | Easy to install | 0.5 | 1 | 0.5 |
| T7 | Easy to use | 0.5 | 4 | 2 |
| T8 | Portable | 2 | 0 | 0 |
| T9 | Easy to change | 1 | 3 | 3 |
| T10 | Concurrent use | 1 | 0 | 0 |
| T11 | Special Security Feature | 1 | 0 | 0 |
| T12 | Provide direct access to 3rd party | 1 | 0 | 0 |
| T13 | Special User Training | 1 | 0 | 0 |

So the calculated factor = 10+4+5+3+3+2+0.5 = 26.5

Then the **TCF** = 0.6 + 0.01 * 26.5 = 0.865

**4.ECF**

ECF = Constant1 + (-constant2 )*Calculated Factor  standard equation

Constant1 = 1.4  (according to the text book)

Constant2= -0.03  (according to the text book)

The explanation of the impact:

0: no impact

1: strong **negative** impact

3: average impact

5: strong **positive** impact

| Environmental Factor | Description | Weight | Perceived impact | Calculated factor |
|---|---|---|---|---|
| E1 | Familiar with the development process | 1.5 | 1 | 1.5 |
| E2 | Application problem experience | 0.5 | 1 | 0.5 |
| E3 | Paradigm Experience | 1 | 5 | 5 |
| E4 | Lead analyst capability | 0.5 | 3 | 1.5 |
| E5 | Motivation | 1 | 3 | 3 |
| E6 | Stable Requirement | 2 | 2 | 4 |

| | | | | |
|---|---|---|---|---|
| E7 | Part-time staff | -1 | 5 | -5 |
| E8 | Difficult programming language | -1 | 3 | -3 |

So the calculated factor = 1.5+0.5+5+1.5+3+4-5-3=7.5

Then the **ECF** = 1.4 - 7.5*0.03 = 1.175

**5.PF**

Although the best solution for estimating the Productivity Factor is to calculate our organization's own historical average from past project, our team is the first time to cooperate and the ability of team members are different. Considering that all of our team members have little experience of software design, we define the PF of us a much higher number : **29**

**6. Duration**

UUCP = UAW + UUCW = 6 +65 = 71

UCP = UUCP * TCF * ECF = 71 * 0.865 * 1.175 = 72.163 **approximately 72 use case point**

Duration = UCP * PF = 72 * 29 = 2088hours

# 5. Domain Analysis

## 5.1 Domain Model

To build a wholly detailed domain model we need to fully review all the use cases and requirement to find out the inner relations between different use cases and the responsibility holders to realize each use cases, that is, the so called concepts. And then we can get the corresponding attributes and associations later.

### 5.1.1 Concept definition:

Personally, we view the responsibility doer as a concept, that is, a section in program to realize a function that can, eventually, work coordinately to complete the whole use cases. Thus, a draft of how-to-work graph is made and then name the concepts one by one.

During making the graph, we firstly divide the actors into non-human and human actors and then according to what they act to draft the boundary concepts. As in this case, the only actor is the user, and he or she, just sets the initial parameters and let the software to simulate the model. For the initial parameters needed to set, we look through the use case and compose scenario, agent_num, round, herding_num, lifeopt……

1. Boundary concepts

| Responsibility Description | Type | Concept Name |
|---|---|---|
| Background or situation choice for users if he or she does not like to set the parameters or certain models user would like to see. | K | Scenario |

| | | |
|---|---|---|
| Container for user's choice of numbers of agents participating in the game | K | Agent_num |
| Verify whether the user type in valid numbers of agents number within certain limits | D | Agent_num_checker |
| Container for user's choice of how many rounds it would simulate | K | Round |
| Verify whether the user type in valid numbers of round times within certain limits | D | Round-checker |
| Container for how many groups the herding effect would cause if the user choose the option "herding" | K | Herding_num |
| Verify whether the user type in valid numbers of herding number within certain limits | D | Herding_num_checker |
| Container for how many agents a group would include if the user choose the option "herding" | K | Herding_scale |
| Verify whether the user type in valid numbers of herding scale within certain limits | D | Herding_scale_checker |
| Container for whether the user choose the option "Life duration" | K | Life_opt |
| Container for the initial score of every agent at the very beginning of the game | K | Score_init |
| Verify whether agents_num is equal or bigger than | D | Parameter_checker |

| | | |
|---|---|---|
| herding_scale*herding_num | | |
| Stop the simulation immediately | D | Sim_stopper |
| Switch to the past simulation graph | D | Past_sim_viewer |

The property of these concepts includes the types, namely, the K type or the D type as shown on the domain model graph later, the “smile” or the “document”symbol tagged on each concept. From the definition on the textbook, Professor Marsic compares K and D to things and workers: Workers get assigned mainly doing responsibilities, while things get assigned mainly knowing responsibilities. The following are the concept diagram divided by K or D while K symbolized by document and D symbolized by smile.

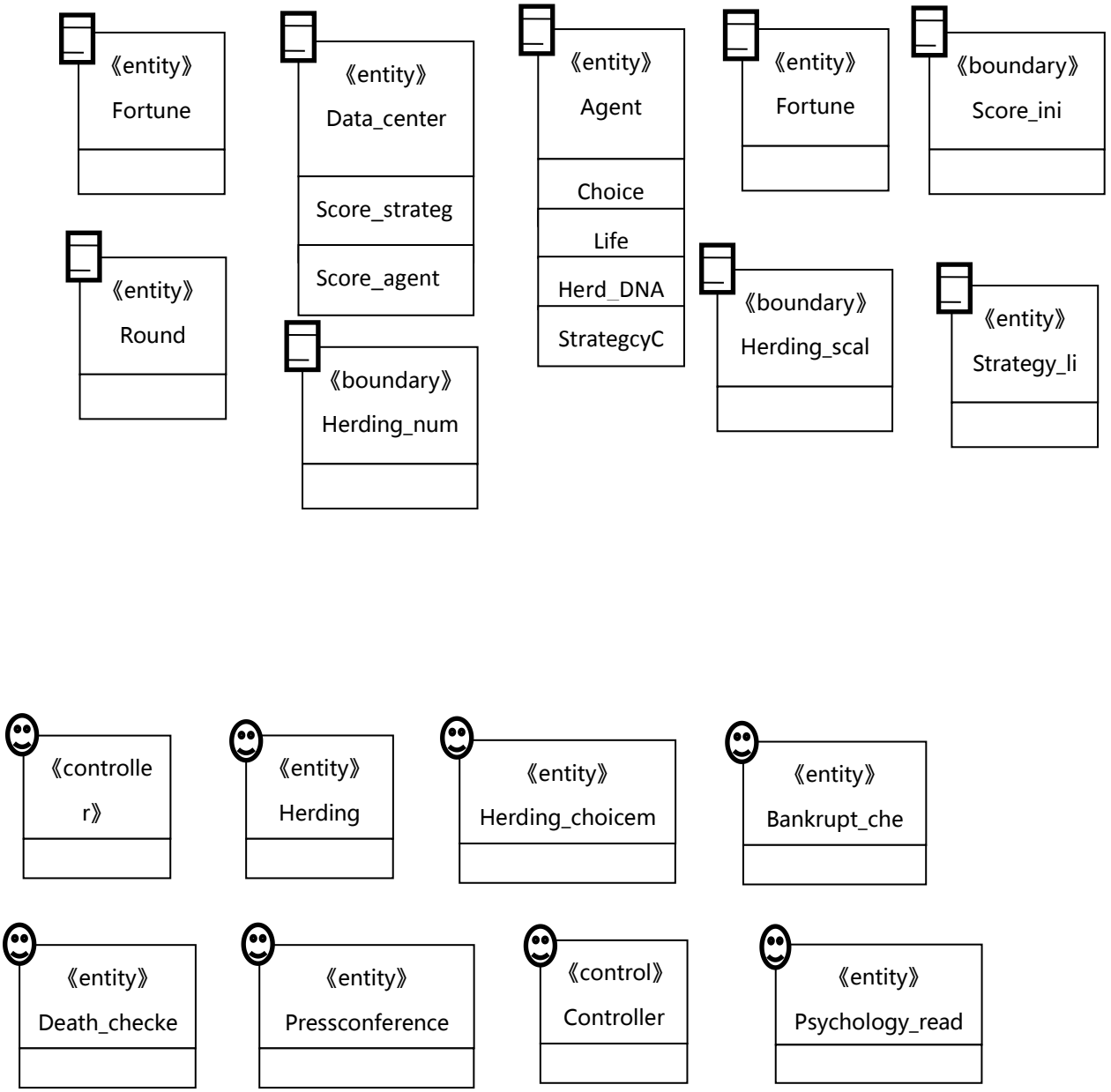



Fig 5 concept diagram

2. Internal concepts

As a matter of fact, the core part of our software mainly lies in our internal concepts; the

boundary concepts, mostly take the responsibility of accepting initial parameters and setting scenarios.

At meantime, the types of concepts are mainly D type as inside the software concepts need to communicate and coordinate with other concepts to fulfill the overall use cases eventually.

| Responsibility Description | Type | Concept Name |
|---|---|---|
| Container for total score of every single agent | K | Fortune |
| Verify whether the score of any agent be equal to 0 after each simulation | D | Bankrupt_checker |
| Container for the score of each strategy and each agent | K | Data_center |
| Update the score of each agent and strategy after every round of simulation | D | Data_updater |
| Reset the score and strategy of certain agents if they are checked bankrupted or checked death | D | Life_maker |
| Container for all strategies each agent may equip with for making decisions | K | Strategy_lib |
| Container for all the statistics of each agent, including choice,life_duration,strategy,herd_DNA | K | Agent |
| Making the choice of each agent and feed the choice back to Agent | D | Choice_maker |

| | | |
|---|---|---|
| Form the groups of agents willing to herd | D | Herding |
| Make a uniform choice of a herd | D | Herd_choicemaker |
| Check if round number has met life_duration of each agent | D | Death_checker |
| Find out the top 3 agents ranking in score and broadcast their strategy at that round | D | Pressconference |
| Figure out the probability of each strategy an agent would like to choose the next round | D | Psycology_reader |

There is one thing we need to notify that the concept "Agent"associates with some sub-concepts as life, choice, strategy and herd_DNA as mentioned in the attributes.

**5.2 Association definition**

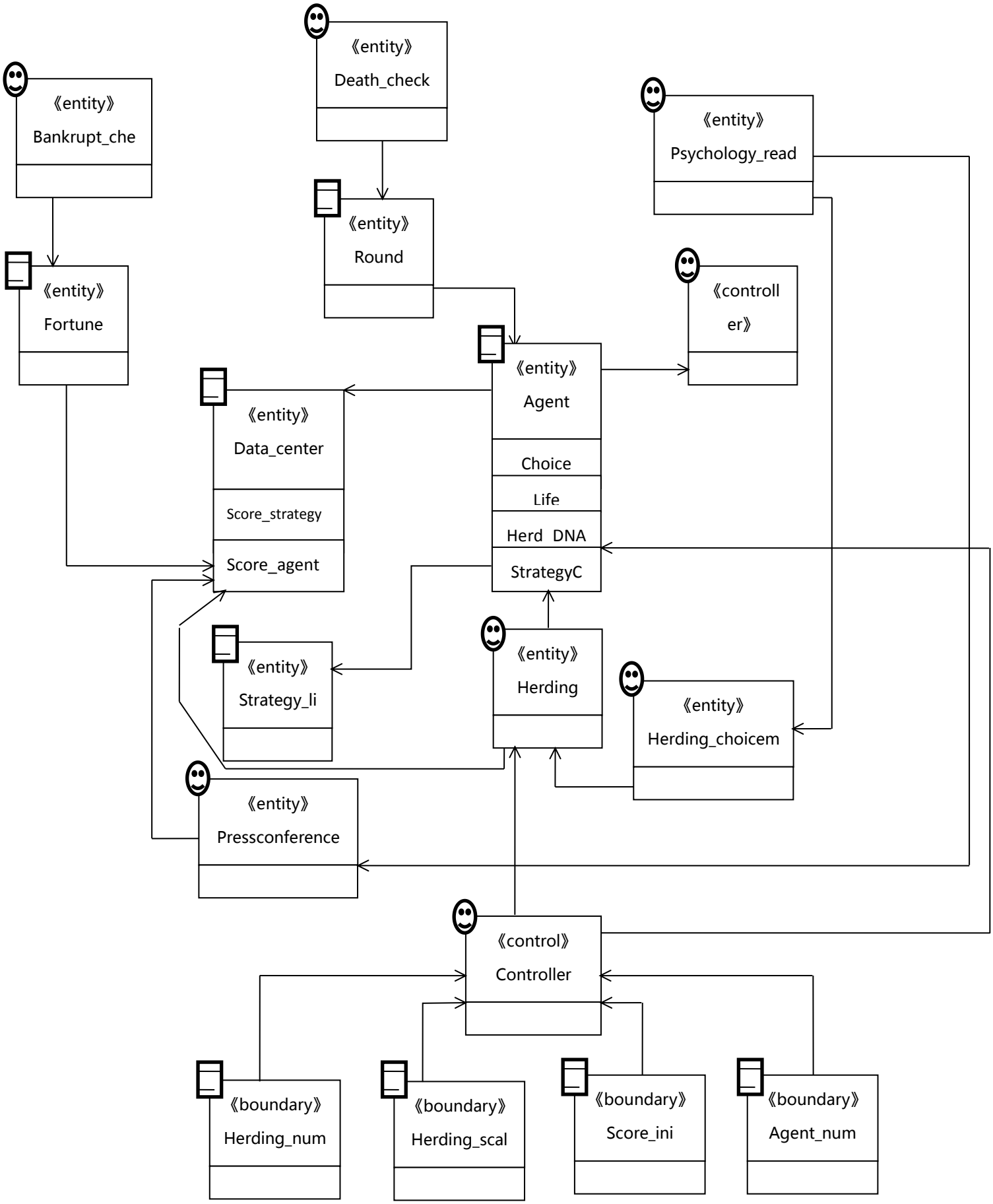
Associations with different concepts are mentioned below. These arrows indicate the relationship between each concept and mainly for conveying information and saving related information. As this software has a very obvious boundary for system and user, we mainly focus on how it operates in the inner side. And one thing is of great significance that there is strictly defined sequence of doing association especially for lots of associations on one concept. Let's take the example of concept Agents: firstly, after each round, Agent will acquire the updated information from Data_center and get the scores of strategies an agent owns. Then, for non_herding agents, they will draw the conclusion by the scores of strategies with the help of psychology_reader, so that's the association between psycology_reader. And for those have DNA of herding, they would first get the strategy by the advancer of the herding and the broadcasting of the top3 giants, so that's the association with herd_choicemaker and conference.

| Concept pair | Association description | Association name |
|---|---|---|

| | | |
|---|---|---|
| Fortune↔Death_checker | Death_checker passes requests to Fortune and receives back each agent's total score | Conveys requests |
| Fortune↔Data_center | Fortune passes requests to Data_center and receives back and save scores of each agent after a round is done | Requests save |
| Conferencepress↔Data_center | Conferencepress passes requests to Data_center and receives back overall scores of each strategy after each round | Conveys requests |
| Agent↔Data_center | Agent passes requests to Data_center and receives back and save scores of strategy each agent owns | Requests save |
| Herd_choicemaker↔ Data_center | Herd_choicemaker passes requests to Data_center and receives back the strategy with highest score in the very herding | Convey requests |
| Strategy_lib↔Agent | Agent passes requests to strategy_lib at first round and receives back N random strategies for every agent. | Requests save |
| Simulator↔Agent | Simulator passed requests to Agent and receives back choice and strategy at the beginning of every round | Convey requests |
| Data_center↔Simulator | Data_center passes requests to Simulator and receives and save scores of that round of each agent and the strategy used by that agent | Requests save |

| | | |
|---|---|---|
| Death_checker↔ Agent | Death_checker passes request to Agent and receives back and save lifeduration of each agent | Requests save |
| Death_checker↔ Round | Death_checker passes request to Agent and receives back how many rounds has processed | Convey request |
| Herding↔Agent | Herding passes request to Agent and receives back how many agents have DNA to herd | Convey requests |
| Data_center↔Agent | Agent passes request to Data_center and receives back and save each score of strategy he or she owns | Requests save |
| Herd_choicemaker↔Agent | Herd_choicemaker passes request to Agent and receives back the sterategy of that agent at the next coming round | Convey requests |
| Herd_choicemaker↔ Psycology_reader | Herd_choicemaker passes information to Phycology_reader of each agent in that herd the strategy of the advancer | Requests save |
| Psycology_reader ↔Agent | Phycology_reader passes request to each agent and receives back strategies and its scores of that agent | Convey requests |
| Choice_maker↔ Agent | Choice_maker passes final choice to each agent let it save | Requests save |
| Conferencepress ↔ | Conferencepress passes strategies of top3 agents to each Phycology_reader | Requests save |

| | | |
|---|---|---|
| Psycology_reader | | |
| Herding↔Herdign_choicemaker | Herding_choicemaker passes request to Herding and receives back herding information and save it | Request save |


《entity》
Death_check
《entity》
Bankrupt_che
《entity》
Psychology_read
《entity》
Round
《entity》
Fortune
《controll
er》
《entity》
Agent
《entity》
Data_center
Choice
Life
Herd DNA
StrategyC
Score_strategy
Score_agent
《entity》
Strategy_li
《entity》
Herding
《entity》
Herding_choicem
《entity》
Pressconference
《control》
Controller
《boundary》
Herding_num
《boundary》
Herding_scal
《boundary》
Score_ini
《boundary》
Agent_num

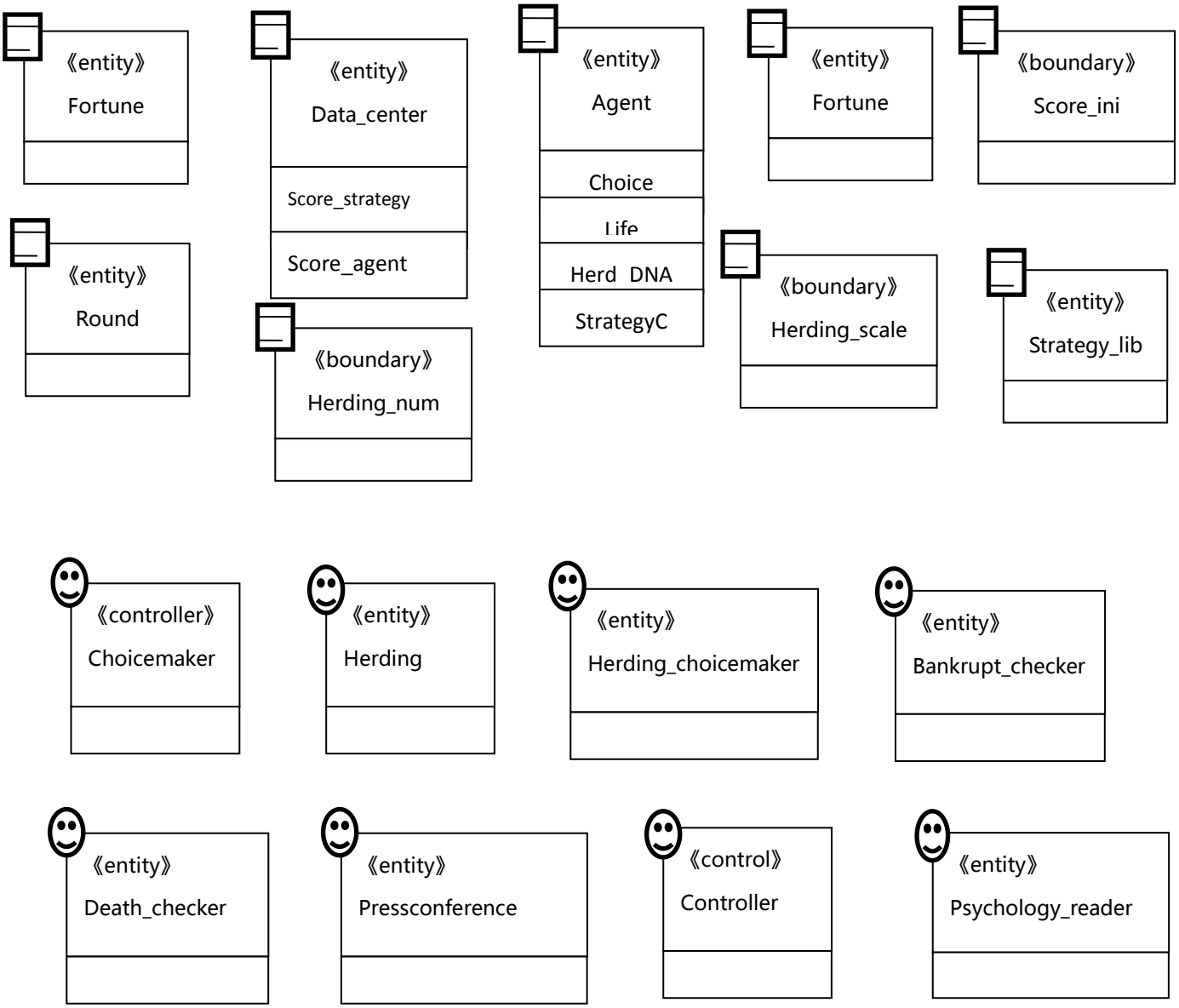



Fig 6 domain model diagram

### 5.1.3 Attribute definitions

The Minority Game has 7 concept attributs: Agent, poorcheck, container, herdconstitution, herdchoice, database and input variables

Each agent will choose their own strategy depend on their memory or the group advancer's suggestion. Furthermore, agents who have Herd DNA will set up groups and share the strategies. However, they are also constrained by life duration, which means how long they can exist in this game system. Thus, the concept of Agent has attributes of Choice, Life Duration, Herd DNA and Strategy_C.

After several rounds, agents whose scores are less than 0, they will excluded the system. So, Death_checker has attribute of Exclude

System needs a container to record and contain the scores of each strategy and agent. Poorcheck and herdchoice will call this data. So NumS_a and NumS_s are attributes for Container.

When we select herding button, we need to determine the number of groups and group scale. Furthermore, we also need to determine which strategy each group will choose. In this case, Herding will have attributes of Herding_scale , Herding_num, and Advcancer.

The system also needs a database to store some initial data, such as strategies, life model and herding model. User should input initial information such as numbers of agents and rounds. So Database and InputVariables also have some attributes.

Attribute Definition shows below.

| Concept | Attribute | Attribute Description |
|---|---|---|
| Agent | Choice(memory) | The strategy a agent will choose each round |

| | | |
|---|---|---|
| | Life Duration | How long they can exist in this game |
| | Herd DNA | Agents who will hold together with others |
| | Strategy_C | The number of Strategies contained in agent's memory |
| Death_checker | Exclude | Agents whose scores are less than 0 |
| DataCenter | NumS_a | Record and contain the score of each agent |
| | NumS_s | Record and contain the score of each strategy |
| Herding | Herding_scale | The total number of groups |
| | Herding_num | The number of agents each group have |
| | Advancer | Agents who have the highest score in a group |
| Database | Strategies | The initial strategies storage |
| | MortalityType | contains enumerated type of mortality model being using |
| Inputs Variables | Num_a | The number of agents participating in the game. |
| | Num_r | The number of rounds played before the end of the game. |

### 5.1.4Traceability matrix

The traceability matrix for is shown in Figure-7. It shows how the system use cases map to the domain concepts

| UC | PW (1-5) | Scenario | Agent num | Agent num checker | Round | Round-checker | Herding num | Past sim viewer | Herding scale | Sim stopper | Life opt | Score init | Fortune | Data center | Life maker | Strategy lib | Agent | Choice maker | Herding | Herd choicemaker | Death checker | Pressconference | Phycology_reader |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UC1 | 5 | X | X | X | X | X | X | | X | | X | X | | | | | | | | | | | |
| UC2 | 5 | | | | | | | | | | | | X | X | X | X | X | X | X | X | X | X | X |
| UC3 | 4 | X | | | | | | | | | | | | | | | | | | | | | |
| UC4 | 3 | | X | X | X | X | X | | X | | | X | | | | | | | | | | | |
| UC5 | 1 | | | | | | | | | X | | | | | | | | | | | | | |
| UC6 | 2 | | | | | | | X | | | | | | | | | | | | | | | |

Fig 7 Traceability matrix

## 5.2 System Operation Contracts

### 1. What are the Sections of a Contract

| **Operation:** | **Name of operation and parameter** |
|---|---|
| **Cross reference** | Use cases this operation can occur within |
| **Preconditions** | Note worthy assumptions about the state of the system or objects in the Domain Model before execution of the operation. These are non-trivial assumptions the reader should be told. |
| **Postconditions** | This is the most important section. The state of objects in the Domain Model after completion of the operation. |

### 2, Our System Operation Contracts

| **Operation:** | **Agent_num_checker** |
|---|---|
| **Cross reference** | UC-1 |
| **Preconditions** | User inputs the number of agents |
| **Postconditions** | Data valid, system runs |

| Operation: | Bankrupt_checker |
|---|---|
| **Cross reference** | UC-2 |
| **Preconditions** | 1,Data-center contain each agent's score successfully<br><br>2,Succeed in receiving data from Data-center |
| **Postconditions** | 1, Agents exclude from system<br><br>2, New agents being added in |

| Operation: | Herd_choicemaker |
|---|---|
| **Cross reference** | UC-2 |
| **Preconditions** | 1, Agents hold together with others<br><br>2, Data-center contain valid data |
| **Postconditions** | A strategy is chosen for the whole group |

| **Operation:** | **Past_sim_viewer** |
|---|---|
| **Cross reference** | UC-6 |
| **Preconditions** | 1, System runs successfully<br><br>2, All data is true |
| **Postconditions** | Analysis graph and data |

| **Operation:** | **Sim_stopper** |
|---|---|
| **Cross reference** | UC-5 |
| **Preconditions** | system is running |
| **Postconditions** | The game is finished |

| **Operation:** | **Agent_maker** |
|---|---|
| **Cross reference** | UC-3 |
| **Preconditions** | All data valid<br><br>Change initial conditions, such as Agent_Num, Life_duration |
| **Postconditions** | System runs with another speed |

# 6. Interaction diagram

In general, there are three parts which form this system: Score-judge, agent, and data center. The score-judge computes the data of every agent and their strategies to the data center. In the same time, it records the winning and losing situation of the strategies of every agent and then pass the data to every agent. While agents receive the data, they update their inner memory. Before the agents make their own decision, they refer to the Advancer's strategy of their own group and the broadcasting from the giants.

This is the general point of how every choice of the agents be made. While the game continues, this part loops. The following will show you a general idea of the use case for the whole system and then give a micro view of each part explaining the details for the software system.

## 6.1 Use Case General Section:

Figure7.1 is the interaction diagram of our whole running game section.

Associated Responsibilities mainly are the same as mentioned above. Agent is responsible for sending choices to Socre-judege, then DataCenter records all information during each round includes agent score, each strategy score of a agent and total strategy score.

During designing running game section, we follow Expert Doer and High cohesion Principle: in our system, almost every object just takes 1-2 responsibilities and do their own tasks, such as herd, Gaint and DeathChecker. []The following are the detailed descriprion of them.

In this system, we add herding and Gaint to make it realistic. Herd means some agents may form groups and share their strategies. We define agents who has herd_DNA will join a group, the others won't share their strategies. The number of groups and the number of agents in each group is constant forever. After herding is formed, then they get each memebrs' socre from DataCenter, and choose the strategy of whom has best agent socre.

Then, on the part of Gaint, Gaint receives 3 agents' informaion from DataCenter, those has the top 3 agent score. These 3 agents make their decisions first, then broadcast their strategies to other whole angets, which will impact their choices. Simultaneously, the strategies they broadcasted may not theirs, may cheat others and broadcast strategies not their own, the probability is 50%.

At last, our DeathChecker will compute each agent's socre and life. If a agent's socre is less than 0 or life has been maximum, our DeathChecker will remove them and add new agents into system(update system).

So, as mentioned above, in our system, a agent's choice wil depend on three parts. The first is his own memory, his memory will record the best strategy. Then is herding choice, members in the same group will select the same strategy. Finally, a agent's choice is influenced by broadcasting.

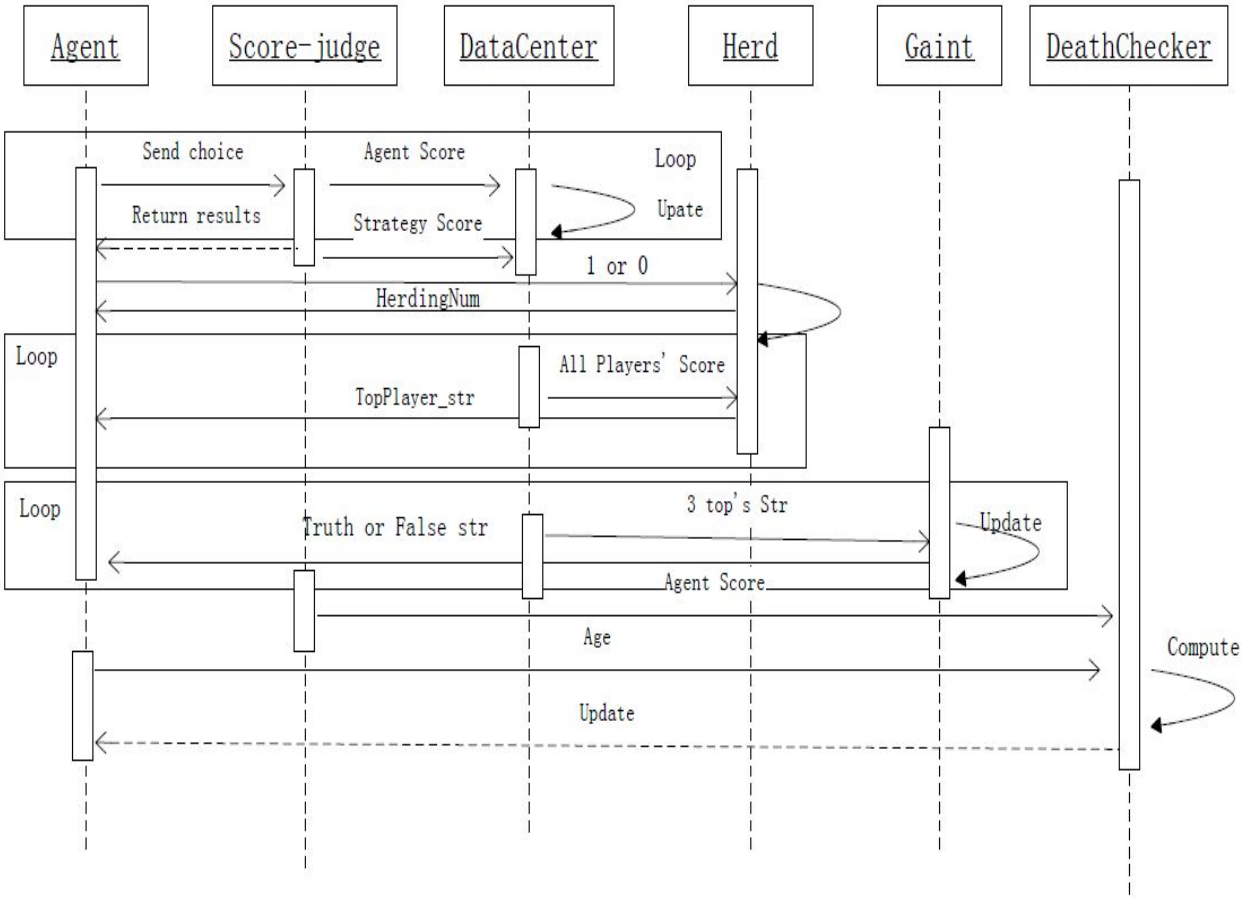



Figure7.1

## 6.2 Set_initial section use cases:

Comparing our software with the other, there is one thing needed to clarify: our system has very little combination with the outside actors after the initial parameters have been set. As the simulation starts, what the users will do is just watching the screen and analyzing the data shown. Therefore, if we just ignore what is inside the system and just show the sequence diagram for use cases, that is not meaningful as our core part remain in the inner side and especially the algorithms. As a result, we will show the sequence diagrams using actors and main entities as well. Here is the part of set_initial section:

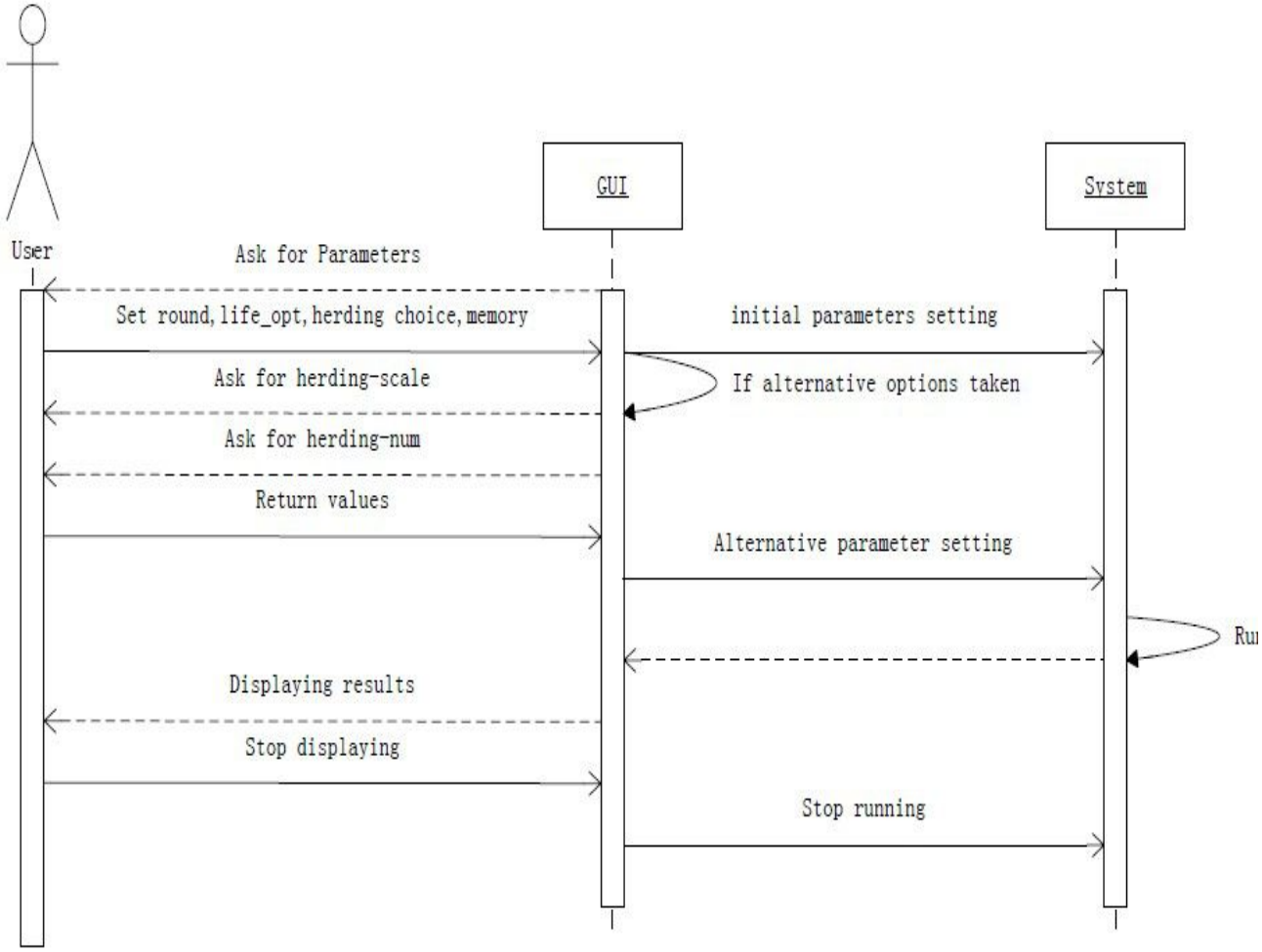



Figure 7.2

Using designing principle of expert doer, this section get every part do what it knows (as we can see that the system just get charge of running and GUI only gets parameters from the user), which is somewhat similar to what normal software will do: that's the main part of user interface and get the general parameters to start the running. So, the GUI will actively ask the user for the initial values of different parameters and see if alternative functions are needed to simulate. The core part is the choice of using herding, a top function in our system. If the user feeds back that herding function is necessary the system will then ask for what the scale for a herding group and how many herding groups are needed. And then, cooperating with the length of memory, the system will get the starting values and eager to start.

While running the part, the screen will show the sequence all the time as every round past all the values updated. So, there is a big loop for the total software displaying the result every moment.

## 6.3 Herding section use cases:

As we mentioned above, herding is a really exciting part of our software, which reflects the real world situation of make choice best for financial market. There is no doubt that everyone lives in a real world social circle, which provides the main information we will receive especially when we want to make a significant choice. We want to add this factor into our minority game model, which will be much more realistic.

We follow the design principle of both high cohesion and expert doer: as a complex part of the use case, we certainly get every section do what it knows: agents to get information and make choice and herd to ask for advancer's strategy; and absolutely, we minimize the doing responsibilities as much as possible as making a reasonable structure.

For the initial part of herding, we need to divide the agents into different herding groups. As these agents has no difference with each other besides its names. So we just put the agents into groups one by one if the agent is willing to herd as we set randomly, as the figure 7.3 shows.

Then, for making everyone's choice, herding will show advantages from the wisdom of the group. We set the group will find the best player in their social circle and take a deep consideration of the

advancer's strategy. To demonstrate that in the algorithm, we set the strategies of every agent's brain a probability to be chosen. If a strategy used leads to a win, then the probability rises. On the other hand, if the herding group releases the advancer's strategy as a bonus of herding, the agent will, as well, get the probability of strategy mentioned risen. That is the micro view of herding.

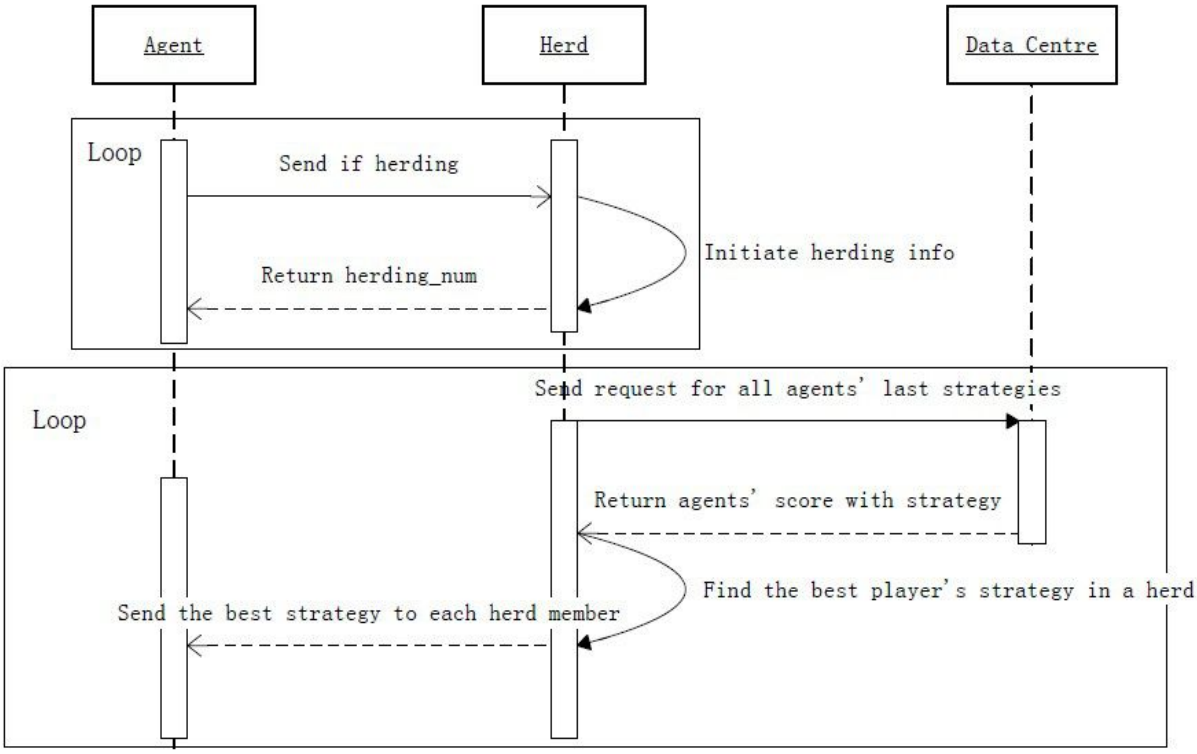



Figure 7.3

## 6.4 Judge_score section use case:

As a matter of fact, this can be view as a top level functional part for the whole system and it will loop all the time until user commands to stop. As we can see from figure 7.4, each agent takes deep consideration of the information from advancer and giants and then gets his final idea to choose a strategy he trusts. In that way, a choice is made for every agent and we need to summarize the overall choice and find the minority victor.

This section follows the low coupling principle, we tried hard to minimize the communication times for the use case and just use three messages to decide the strategy of each user as below.

However, that is not the end yet, to make every end of simulation meaningful, we shall take records for every strategy and agent for analyzing. After each round, the score_judge will send feedback to Data_centre to update the scores for each agent and strategy. And the agents will get feedback as well for update each strategy's probability in his minds.

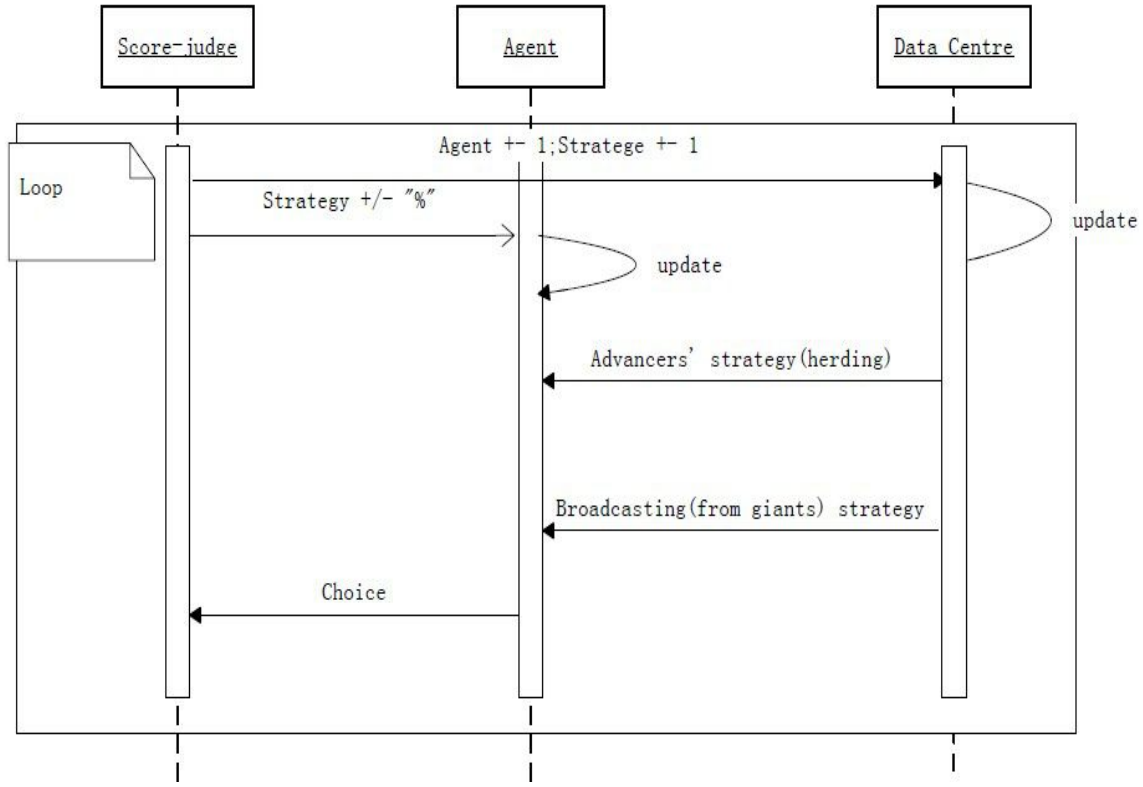



Figure 7.4

## 6.5 Gaint Section Use Case:

Agent is the actor of the whole system, mainly responsible for sending choices to Score-judge. Data_Center records all information of the system, include agent score, each strategy score of an agent and the total strategy scores. Giant has the best agent scores in system and will broadcast their strategies to others and also will cheat others.

For the design principle of this use case, we follow Expert Doer Principle, each parameter has its own responsibility and knows do which task. Data_Center sends the top 3 agents' information to Giant, and these 3 agents in Giant make their decisions first, then broadcast their strategies to all agents and will impact their decisions. Furthermore, they may cheat others, we define that they have 50% probability to broadcast other strategies (not theirs) to agents and 50% probability to broadcast their own strategies to agents.

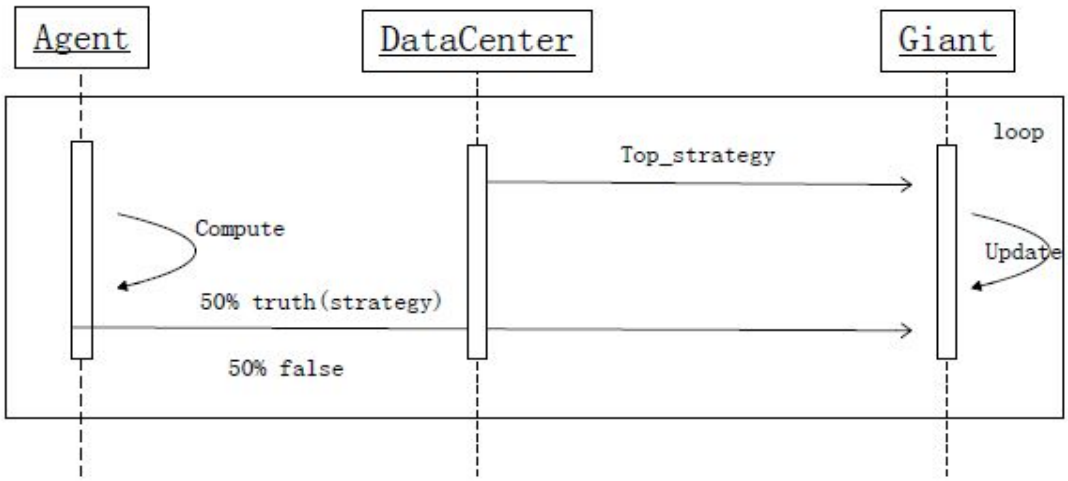
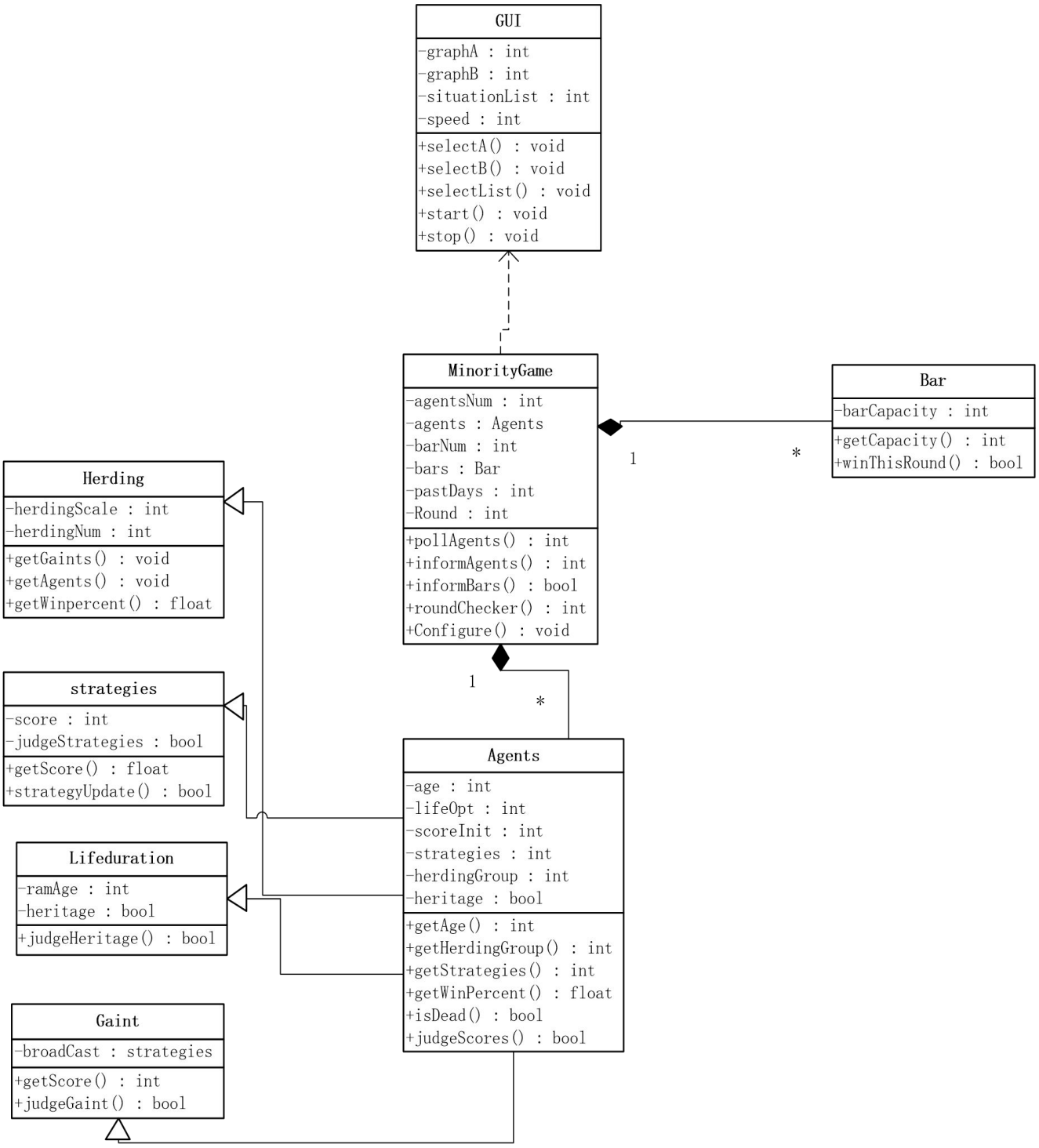



Figure 7.5

# 7. Class Diagram and Interface Specification

## 7.1 Class Diagram



## 7.2 Data type and operation signature

**Note**: - private, + public, # protect

**GUI**: The GUI class is in charge of managing the graphs for the user and the input variables.

- graphA: int

- graphB: int

-situationList: int

-speed: int

+selectA(): void

+selectB(): void

+selectList(): void

+start(): void

+stop():void

**MinorityGame**: The MinorityGame collects data and acts as intermediaries for the other classes. It returns data concerning the round to the GUI.

-agentNum: int

-agent: Agent[]

-barNum: int

-bars: Bar[]

-pastDays: int

-round: int

+pollAgents(): int

+informAgent(): int

+informBars(): bool

+roundChecker(): bool

+configure(): int

**Agent**: The Agent contains the basic score and memory decisions.

-age: int

-lifeOpt: int

-socreInit: int

-strategies: int

-herdingGroup: int

-heritage: bool

+getAge(): int

+getHerdingGroup(): int

+getStrategies(): int

+getWinPercent(): float

+isDead(): bool

+judgeScores: bool

**Strategies**: The Strategy is a logical unit for holding an Agents bank of strategies and their success for a particular Agent.

-score: float

-judgeStrategies: bool

+getScore(): float

+stragegyUpdate(): bool

**Herding**: The herding contains herding group construction and the giant of the group judgment.

-herdingNum: int

-herdingScale: int

+getGiants(): void

+getAgents(): void

+getWinpercent(): float

**LifeDuration**: The LifeDuration contains the definition of age of each agent and judge that whether a agent would get a heritage.

-ramAge: int

-heritage: bool

+judgeHeritage(): bool

**Giant**: The giant contains the element of judge if an agent is a giant and if a giant use broadcast function.

-broadcast: strategies[]

+getScore(): int

+judgeGiant(): bool

**Bar**: The bar knows its capacity and if the people at the bar won.

-barCapacity: int

+getCapacity(): int

+wonThisRound(): bool

## 7.3 Traceability Matrix:

| Domain concepts | Classes | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Agent | Minoritygame | Strategies | Herding | Lifeduration | Giant | Bar | GUI |
| Choice(memory) | X | | X | | | | | |
| LifeDuration | X | | | | X | | | |
| HerdDNA | X | | | X | | | | |
| StrategyC | X | | X | | | | | |
| NumSA | X | X | | | | | | |
| NumSS | | X | X | | | | | |
| HerdingScale | | | | X | | | | |
| HerdingNum | | | | X | | | | |
| Advancer | | | | X | X | X | | |
| Strategies | | X | X | | | | | |
| MortalityType | X | X | | | X | | X | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| NumA | | X | | | | | X | |
| NumR | | X | | | | | X | |
| Start | | X | | | | | | |
| ErrorDetector | | | | | | | | X |
| GraphDisplay | | X | | | | | | X |
| GUI | | | | | | | | X |

**Justification:**

Our classes and methods relied heavily on our domain concepts, and they accurately reflect the system architecture. For example, all of the related data center will be grouped together for relatable processes.

For the agent part, choice reflects the strategy an agent will choose each round. Life duration defines how long an agent can exist in this game. Herd DNA expose which agent would willing to hold together with others. Strategy C implies the number of strategies contained in agent's memory.

For the herding part, herding scale reflect the total number of groups. Herding num contains the number of agents each group has. And advancer means the agents who have the highest score in a group.

Similarly the GUI concept has directly influenced the creation of our GUI class.As previously explained in the Game Simulator Concept the minority game class will act as the controller unit. In addition, the minority game also has its own set of duties including setting up and monitoring the

town itself as the simulation progresses.

# 8. System Architecture and System Design

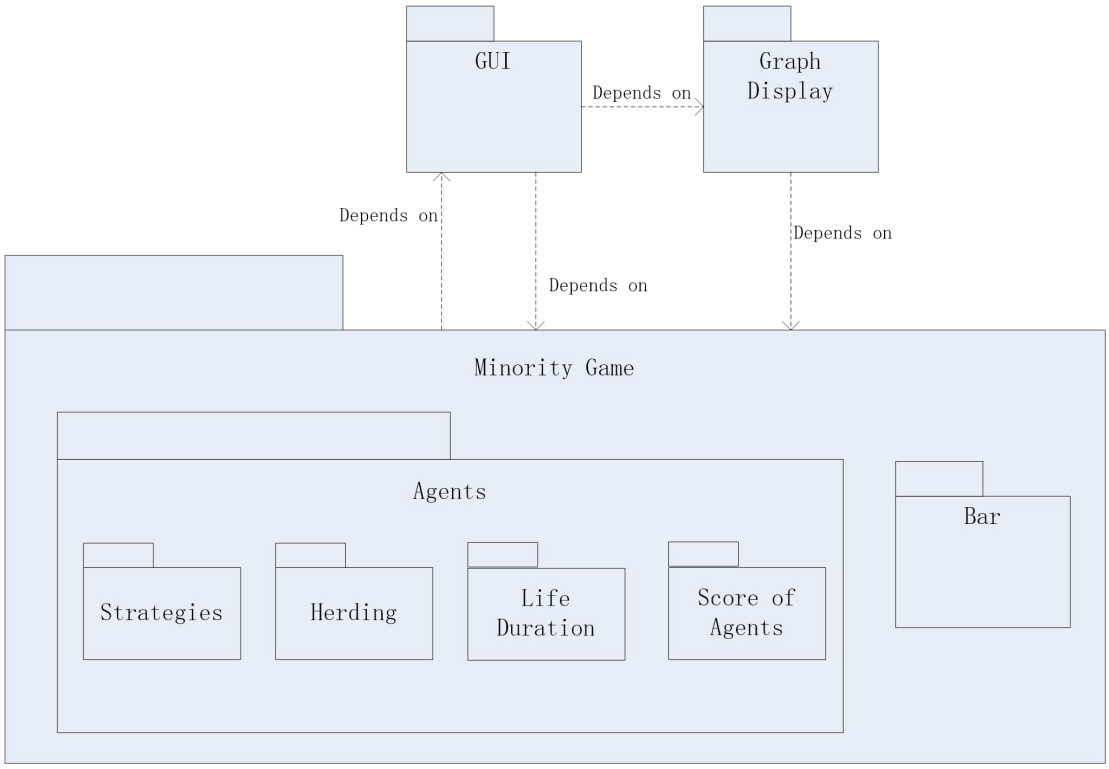
## 8.1 Architecture styles

The architecture style we used in our system is event-driven. Event-driven architecture is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. An event can be defined as "a significant change in state". In our system, an event can be producing the result of a round or replacing an agent.

An event-driven system typically consists of event emitters (or agents) and event consumers (or sinks). Sinks have the responsibility of applying a reaction as soon as an event is presented. The reaction might or might not be completely provided by the sink itself. When the events mentioned above occur, all the agents in the game should respond by changing the capital and scores of strategies.

From another point of view, our system also has the Monolithic application style. Because the user interface and data access code of our simulating system are combined into a single program from a single platform. Our application is self-contained and independent from other computing applications.

## 8.2 Identifying subsystems




There are three main subsystems in our application: the GUI, Graph Display, and the Minority Game. The GUI subsystem is in charge of getting input data from the user (death, number of bars, etc); it is the sole communicator from user to Minority Game. It also displays data that the Minority Game computes in the format of graph that the Graph Display generates. Therefore, the GUI subsystem is dependent on the Minority Game and Graph Display subsystems. Graph Display depends on Minority Game to receive data. The data is then outputted to graphs that are displayed in the GUI.

Minority Game is the main subsystem. It has subsystems within itself: Agents and Bar. Bar knows its capacity and notifies the Minority Game whether the people at the bar won. Each agent has

his own set of strategies, so the Strategy subsystem is contained within Agent. Also along with each of their strategies, each Agent has a Life Duration which is given to them upon creation. Another subsystem of Agents is Herding. Herding allows for agents to form groups to make decisions. Score of Agents keeps track of the Agents' score (capical).

## 8.3 Persistent Data Storage

Our application will generate a "txt" file each time it runs. The text file contains the execution log of the simulation, which can be used by the application to regenerate the graphs. Since "txt" files are easy to use, the simulation log can be used by other applications to analyze the data.

## 8.4 Global Control Flow

### 8.4.1 Execution orderness

Our simulation application is procedure driven. The user needs to input their preferred settings for the game and press "Simulate" to start the simulation. Once the game is running, the rounds of game and the strategies are executed in an iterative process and the GUI is updated automatically. The speed of this linear procedure can be controlled by using the slider bar at the bottom of the GUI screen. When the user is satisfied with the data generated and recorded so far or would like to enter another set of options for the simulation, they may pause and stop the game.

### 8.4.2 Time dependency

The system is of the event-response type. Each round happens very quickly and could be considered an instantaneous event. The application should run as fast as possible in order to satisfy user's demand quickly. The only time relevant aspect of our application concerns the slider while the simulation is under way. The purpose of the slider bar is to set the speed of how fast the round progresses. The time that the graphs are displayed for the current round in the GUI is delayed by a certain amount depending on the slider bar. This also delays the calculation for the next round in the background.

## 8.5 Hardware Requirements

| Hardware and Software Requirements | Minimum | Recommended |
|---|---|---|
| Display Resolution | 800 | 1024 |
| CPU | 1GHz | 2GHz |
| Size on Disk | 10MB | 10MB |
| RAM | 512MB | 1GB |
| .NET | 3.5 | 4.0 |
| Visual C++ | 2010 | 2010 |
| Operating System | Windows | Windows |

# 9. Algorithms and Data Structure

## 9.1 Algorithms

### 1) Town Creation

Town is the overarching class that sends data to where it should be. Upon creation it creates a list of Groups and then populates them.

### 2) Bar Creation

Bar is a simple object that only keeps track of its own size.

### 3) Group Creation

Groups are collections of agents. Upon creation it is populated with a list of Agents that are to be in its group. Agents within the same group can exchange their intended choice of the round and make decision based on the information.

### 4) Agent Creation

Agents are the players of the game who win or lose each round based on the actions of other agents. Upon creation they are given some strategies that are to be used later to make decisions, a death day based on a Gaussian distribution centered at avgAge , and an age initially set to 0 (see Mortality)

### 5) Strategy Creation

Strategies tell the agents whether they are going to the bar in each round. They contain an Integer array of length 2048 (this is chosen due to memory constraints) and a score that is initially set to 100.

### 6) Agent Strategy Choosing

In any given round an Agent must choose a strategy to use based on the scores of its strategies. Since

score of all strategies are updated every round, the agent can just use the strategy with highest score.

**7) Group Decision Making**

Group members first ask each others' intended choice of this round, and then use this value as a "vote". The result of the vote is then returned to the agent and the agent will make its new decision based on this result. In some case the agent would make a different choice with its groupmates. (See Cheating)

**8) Choosing Winners**

After each group has chosen which bar it is going to this data is collated by Town and relevant data is then sent to the bar that needs it. That bar returns how under cap it was (a number >1 indicates it was over cap) the bar that was the most under cap is then declared the "winner" of the turn the information about which bars won is then sent to each agent.

**9) Updating/Dropping Scores**

After an agent is informed of which bars have won on this turn they update the scores of its strategies. They go through each of their strategies and see if the suggested choice of the strategy is the same as the final result. If they are the same, the strategy increases its score. Independent of whether the Strategy won or not it is then multiplied by .95. This is to ensure that more recent actions count for more than older actions. Each strategy is then checked to see if it below a threshold value (set by user). If the score of a strategy is below the threshold, it would be replaced by a new strategy.

**10) Short Term Memory**

Short Term Memory is the agents' memory of the results of the latest rounds. The agents use the short term memory as the input of the strategies to make decisions.

**11) Mortality**

After a round is over Town goes through each agent and increases its age by 1. If the age is now

greater than or equal to its deathday the Agent “dies”. If all agents in a group die then that group is removed. The agents that died are then replaced in the simulation by adding them to the most recently added group. When this group is filled up a new one is created and

**12) Cheating & Broadcasting**

The advanced agents are allowed to use the broadcast function, that is, telling all other agents in the game their suggested choice. The advanced agents may cheat other agents in order to maximize their own benefit.

## 9.2 Data Structure

1**) Array**

The most used data structure in our code is an array used to store values in an ordered fashion and to get a specific entry in constant time.

**2) List**

This is used to store Groups in the Town class. This data structure make it easier to remove and add "nodes" as needed as this is the only group that changes length as the program progresses.

**3) graphPtr**

This is the self made structure that we use to pass data from the backend to the GUI.

# 10. User interface design and implementation

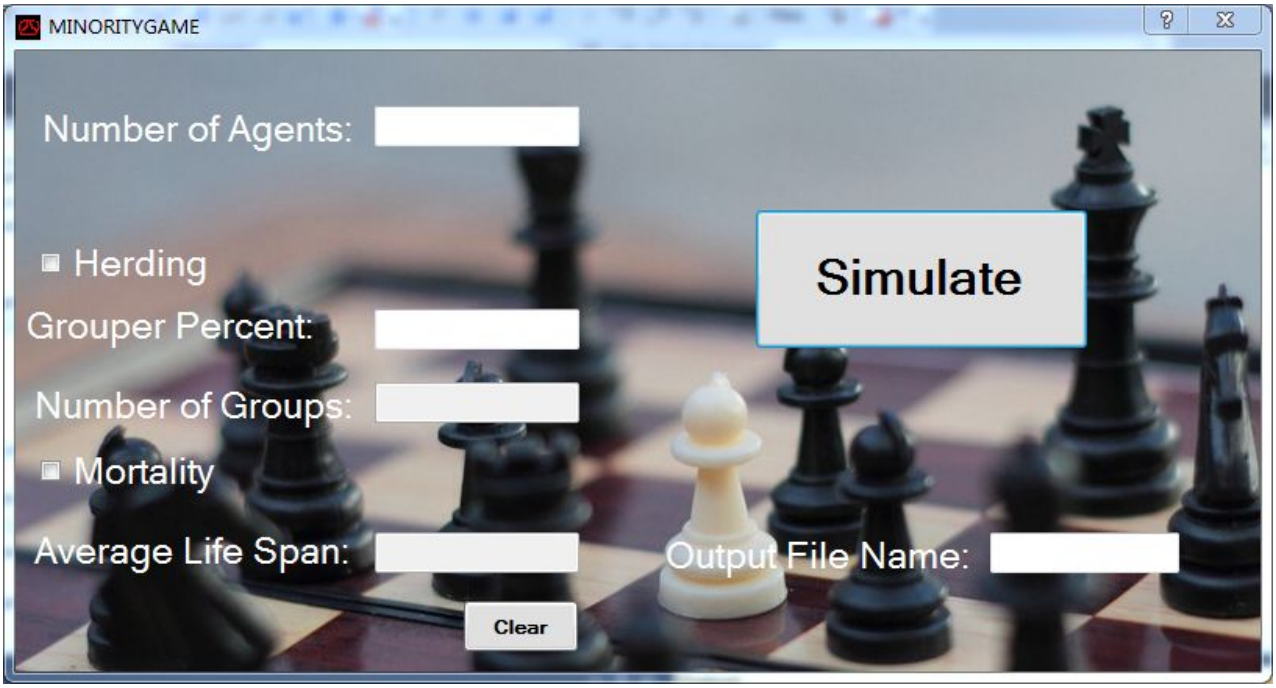



Figure 10-1

Since demo 1, we have made great effort to modify our main user interface, as our target is to make it user-friendly and idiot-proof. In addition, some functions for main UI need to be claimed as we step further into the project. That is: to check the values users type in and give feedback before our program start to run. Figure 11-1 is our new interface.

To see the interface, we need to let the user to feel that they are experiencing a journey rather than stiff software. On the other hand, our software aims to provide a professional graphing and statistics. We need to design an interface with brief and professional style.

Initializing value:

First graph is the original mode, only agents,no other extra feature. Input 1000 agents all only for simple, you can choose any integer number.


Figure 10-2-1 initialization - original function

Second graph is with all of our features, like group, mortality, and broadcast function. The number fill in the blank is not the standard one, you can


Figure 10-3-2 initializing -all feature

Third graph of initialization is about the wrong input, if user input the wrong number, the application will appear wrong icon and tell user where is wrong. User should clear the number and fill in them agagin.


Figure 10-3-3 Initialization - wrong input

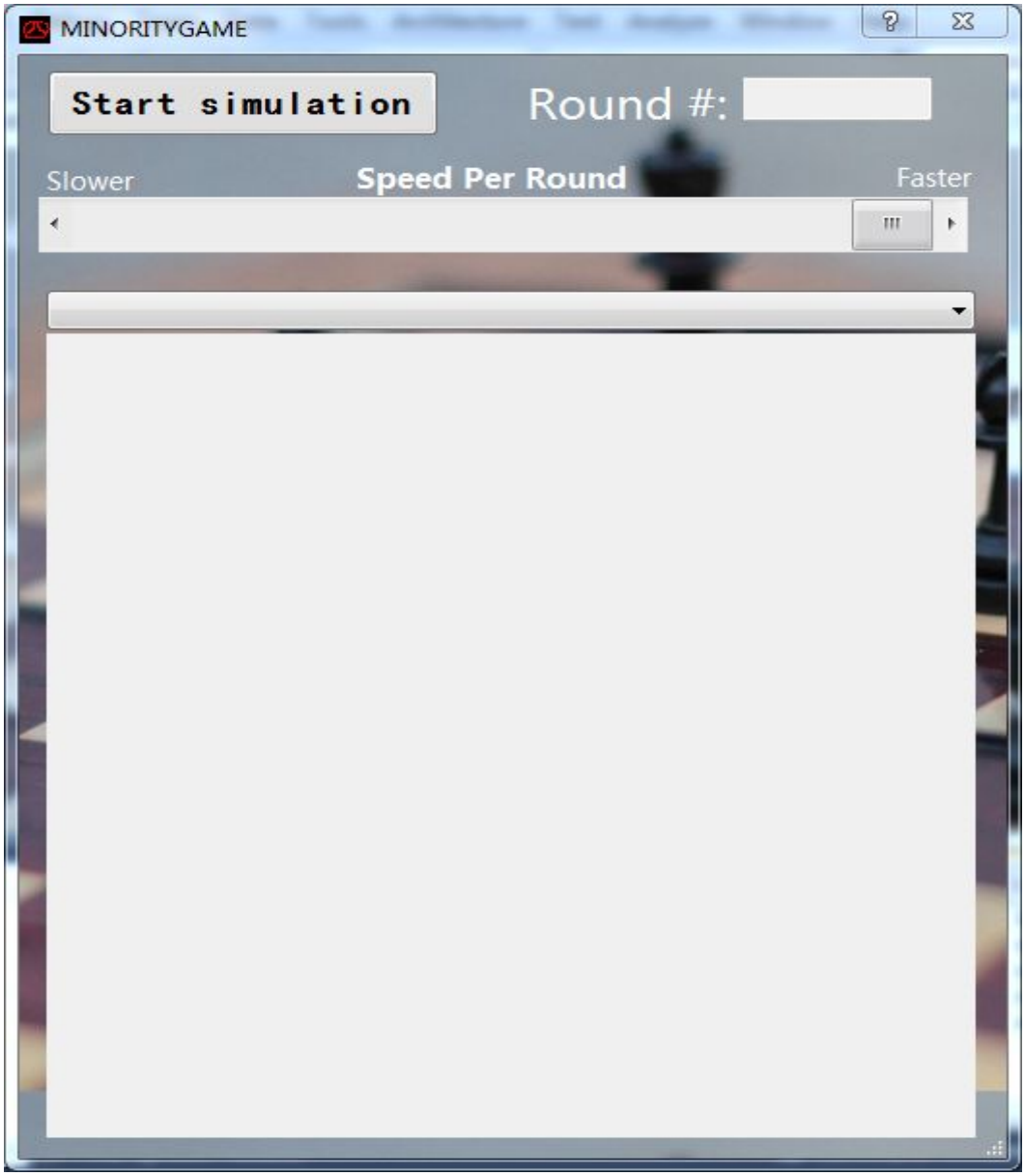



Figure 10-3 simulation graph

After checking the parameters, a window will come up and display the statistics information. As the simulation goes on, the graph will meanwhile show the live information and it is accessible to switch the content to another type of diagram for the customer's sake. Also, user can stop running whenever he wants.

In the second User Interface, we can see there is a slide. User can choose the graph they want to show on the Screen.

Average number of groups:

Shows how many groups exist in every rounds.

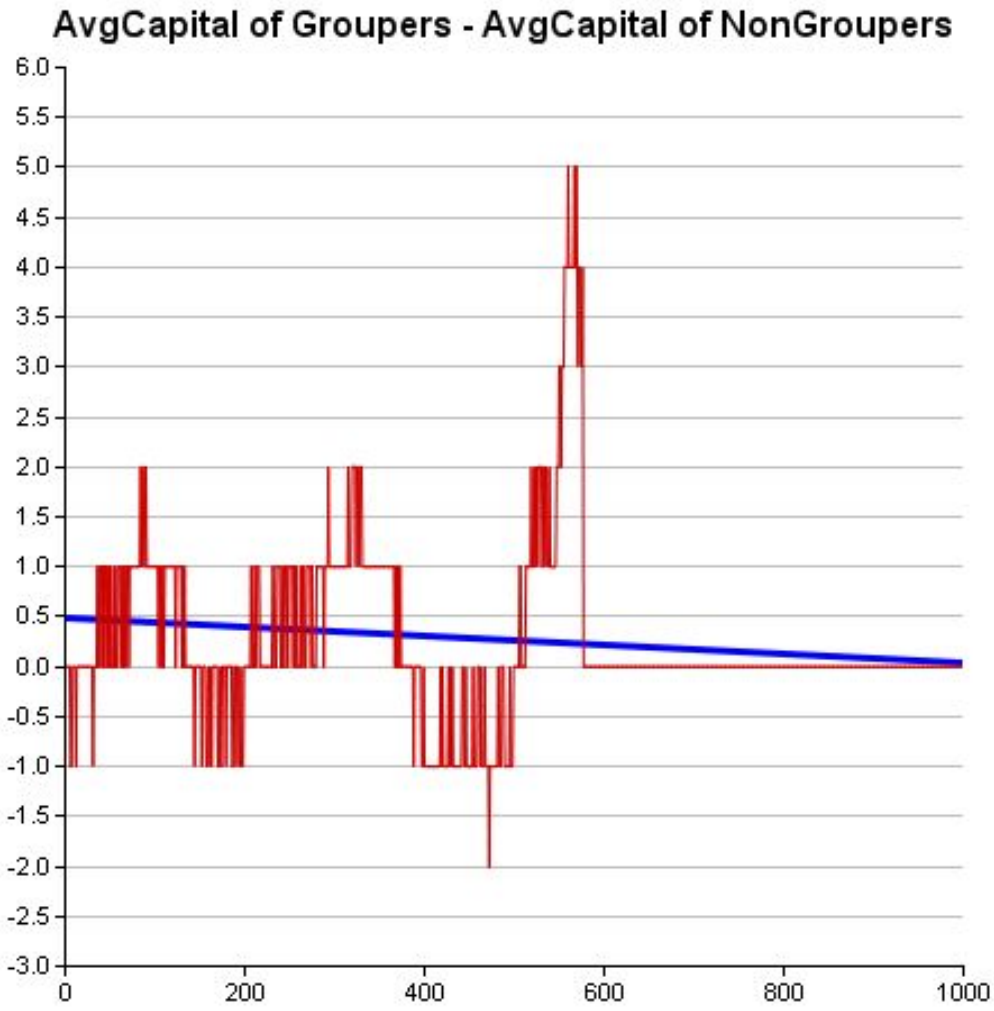



Figure 10-4 Average number of groups

Average Score of all Strategies:

Every agents have their own strategies, here's the average score of all strategies.

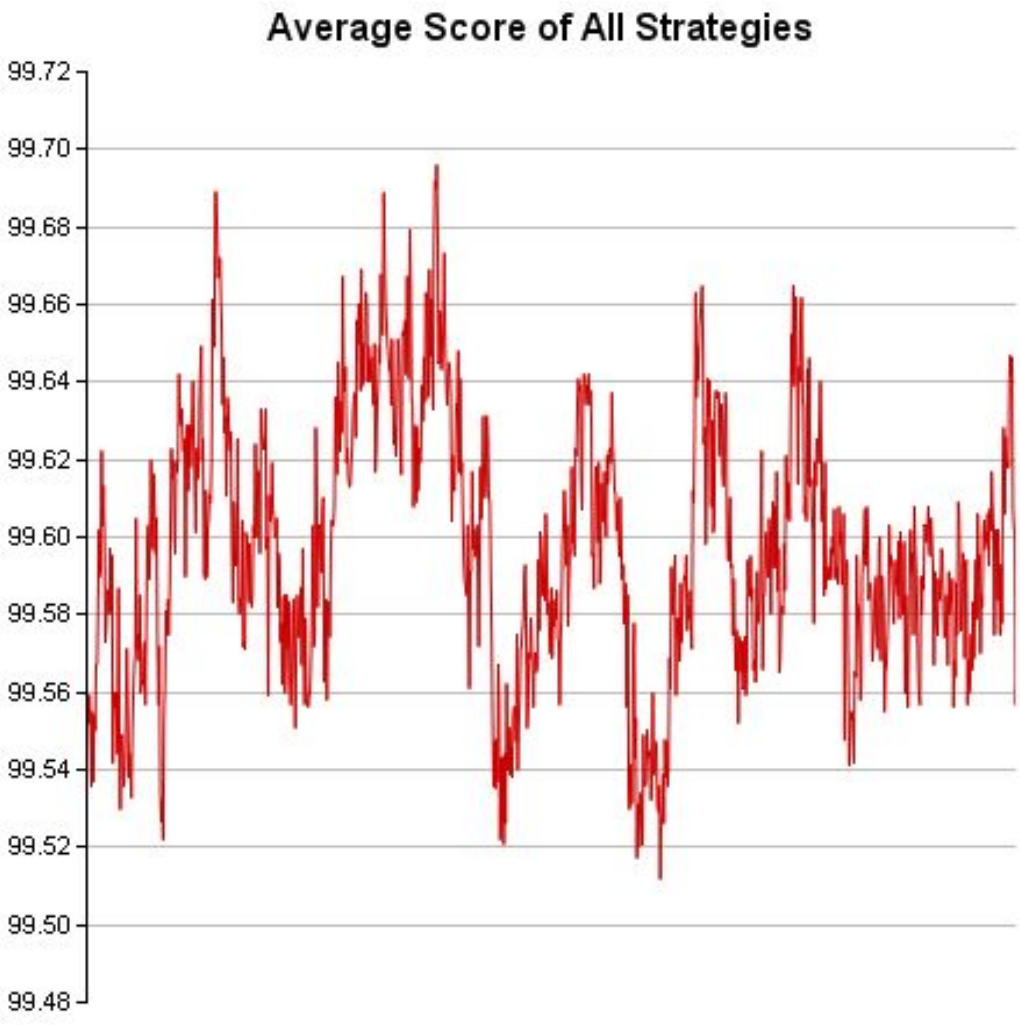



Figure 10-5 Average Score of all Strategies

Number of Death per round:

When we set the mortality feature, every agents have ages, and this graph shows how many agents dead per round. We can see the distribution of ages of all agents.

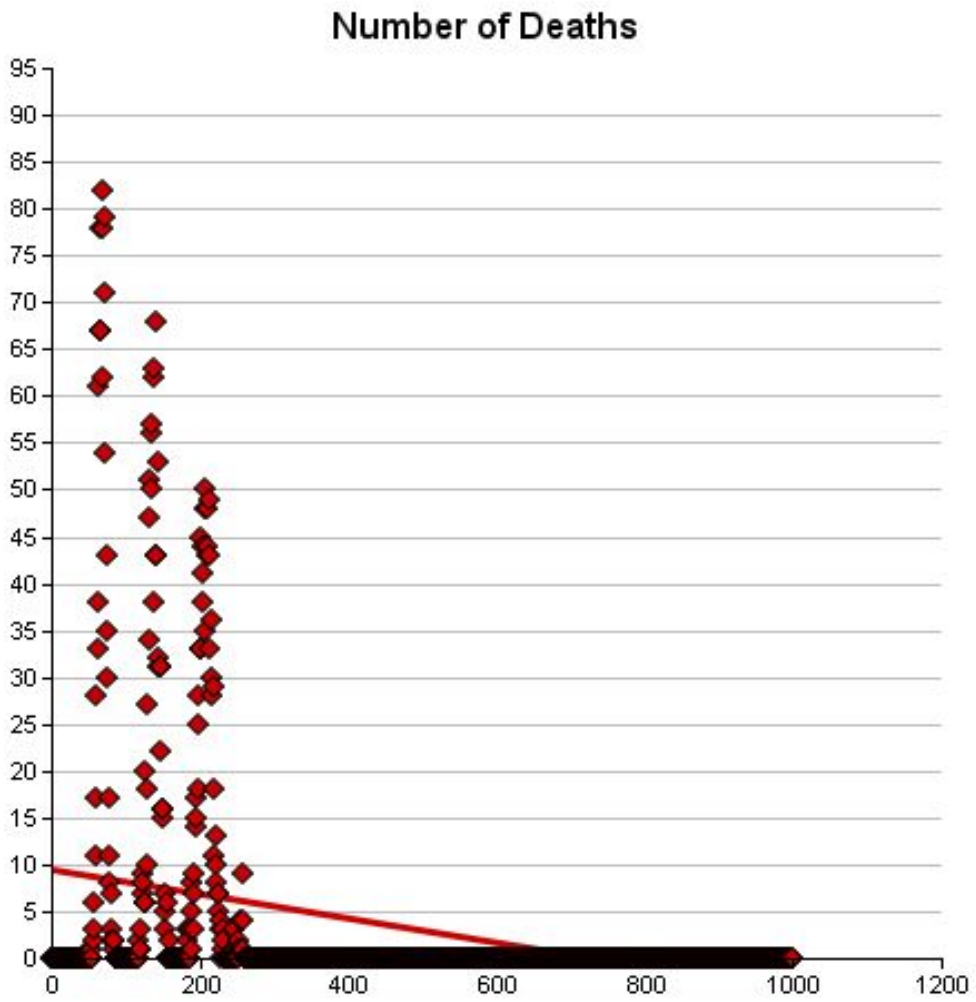



Figure 10-6 Number of Death per round

Number of Winners per round:

This graph shows how many agents win per round. You can see that sometimes very few agents win, this situation happens since the feature of “richest guy” and “broadcast”. The richest guy broadcast the wrong suggestion, and most of the agents will follow him.


Figure 10-7 Number of Winners per round

Score of Best strategy:

Every strategies have their own score, we have show the average score of all strategies before, and this graph is show the best. It does not mean that this graph always show the same one, since the best one will always change. In fact, there is no forever best strategy.


Figure 10-8 Score of Best strategy

The graph below shows the comparison between agents in groups and agents who are individuals. The y-axis is the value of “average score of group agents / averages score of individuals”. You can see the trend of this value keeps growing. So the conclusion is that most people in groups do a better job then individuals. But all we said is “most”, not all agents. Some individuals also have a high score.

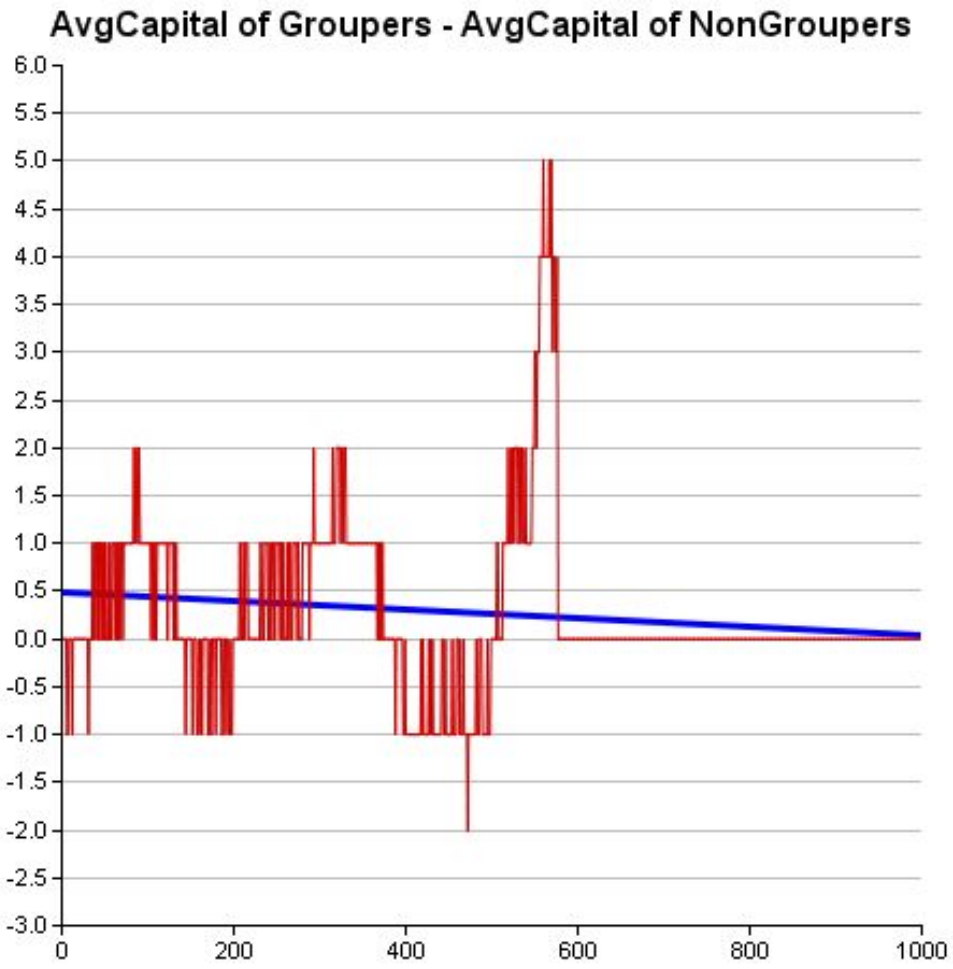



Figure 10-9 comparison between agents in groups and individuals

# 11. Test Design

11 test cases are below to determine the correctness of 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 new window pops up with display information for graphs
Pass/Fail: Passes if system prompts a new window to pop-up. 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, speeds slider

Steps to be executed:

(1) Press Run simulation button

(2) Change charts using drop down

(3) Move speed slider to an arbitrary amount to the left

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 left, the rate of animation 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 Speed 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.

**Output file**

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.

**Agent Model**

Unit to test: Agent 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.

**Herd Model**

Unit to test: Group 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.

**Town Model**

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.

# Finale

**History of work**

This project about minority game is not a new project in software engineering course, there are about 2 or 3 precursors before us to work on it. We are so lucky that they build a strong base of this project and make us like standing on the shoulder of giant. And We are also very lucky that we found the right team and right members at the very beginning.

At the start of September, after we decided to choose this project as ours, we started to read the codes of former editions deeply, and run the former project to find what's their drawback and what should be changed. What's more, we went though all the reports of former editions and had a profile of what we should do during the semester. Moreover, we also read some materials and papers about minority game to make us understand what we are doing more clearly. We find that the former edition create more than one bars in the project however it is against to our goal since we want to go deep of Minority game. Thus, we give up the former mode and go back to the original one(only one bar here), but more features to influence on agents to make decision.

One week later, mixing all of the knowledges and understanding, we finished our proposal. We are so clear that our goal is to make this project more like the real world. So we add new interesting but useful features like herding, giant, broadcasting. We want to simulate how people making decision when facing a problem with only two ways in the real world.

At the start of October, after a month study in class, we finished report 1 of this project. And as the same time, we finished the original mode of bar game theory. It included the most basic function, and what we need to do during the next two month is only to add our new feature there. Since we finish it too smoothly, we stupidly though that this project is very easy and no need to follow the rules of reports at that time. So during the report 2 time , at first we did not finish the requirement in the

report 2 very seriously. We just briefly describe the parts like Interaction diagram, Interface Specification, and the Design of Test . Then, before the Demo 1, the problem appeared. When we are adding the feature of herding, our coding structure became disordered. It was really hard to find where the logic errors were when debugging. It wasted much of our time. And disappointedly, we did not finished any of our new features before Demo 1. Maybe that was why our Demo 1 is so flat and not many new things.

Our truly milestone is when we were writing peer review of report 2 to other groups. We found some of the groups did a really good job that their logic is so clear and every steps were in orders. And go back to read our own, it was too frustrated. Suddenly, we realized that we were doing a big project, not the small project like what we did during undergraduate. We should follow the design pattern and test pattern to finish our work step by step. Hurry would hurt us badly. So we go back to our report, and change many of its contains especially about the interface and design of test. And then surprise happened. After we finished all these steps, we found our thought became clear and it became each to test each part and find where the problem was. So before the Demo 2, our progress became smooth again, but this time, it was not because the task was easy, but because we follow the rule of software development and our thought and logic were clear.

**Current status**

- Implemented GUI using Visual Studio follow the FURPS+
- Implemented the functions of herding, giant, broadcasting
- Implemented the optimization of Algorithm when making decision
- Implemented 6 diagrams to show different results of simulation.
- Implemented the output .txt file which includes all the results of simulation

- Implemented the optimization of Speed slide.

- Create a website about our project

**Future work**

As we said before, our goal is to make this project more like the real world situation. So we will follow our goal for the future work.

- To create a Stock-market-based simulation rather than bar-based.

- To separate agents into different characters like gender, age, even country and region.

- To create Mobile App

- Learn more skill to beautify the UI

# Reference

[1]Dr Richard J. Botting. Sample: The Object Constraint Language. (18 September 2007) from http://www.csci.csusb.edu/dick/samples/ocl.html

[2]Project #3, group #7, spring 2012 - http://www.ece.rutgers.edu/~marsic/books/SE/projects/MinorityGame/2012-g7-report3.pdf

[3]Textbook of Software Engineering by Professor Ivan Marsic

[4]El Farol Bar Problem and the Minority game Project Description

[5]Understanding of Financial Networks through Minority Game, Savio Jude D'souza.

[6]Software Architecture http://en.wikipedia.org/wiki/Software_architecture#Examples_of_Architectural_Styles_.2F_Patterns

[7] Software Engineering by Ivan Marsic

[8]El Farol Bar Problem - http://en.wikipedia.org/wiki/El_Farol_Bar_problem

[9]El Farol Bar Problem and the Minority game Project Description -http://www.ece.rutgers.edu/~marsic/books/SE/projects/MinorityGame/

[10] Project #3, group #4, Spring 2011 - http://www.ece.rutgers.edu/~marsic/books/SE/projects/MinorityGame/2011-g4-report3.pdf

[11]Project #3, group #7, spring 2012 - http://www.ece.rutgers.edu/~marsic/books/SE/projects/MinorityGame/2012-g7-report3.pdf

[12]Project #3, group #10, spring 2012 - http://www.ece.rutgers.edu/~marsic/books/SE/projects/MinorityGame/2011-g4-report3.pdf