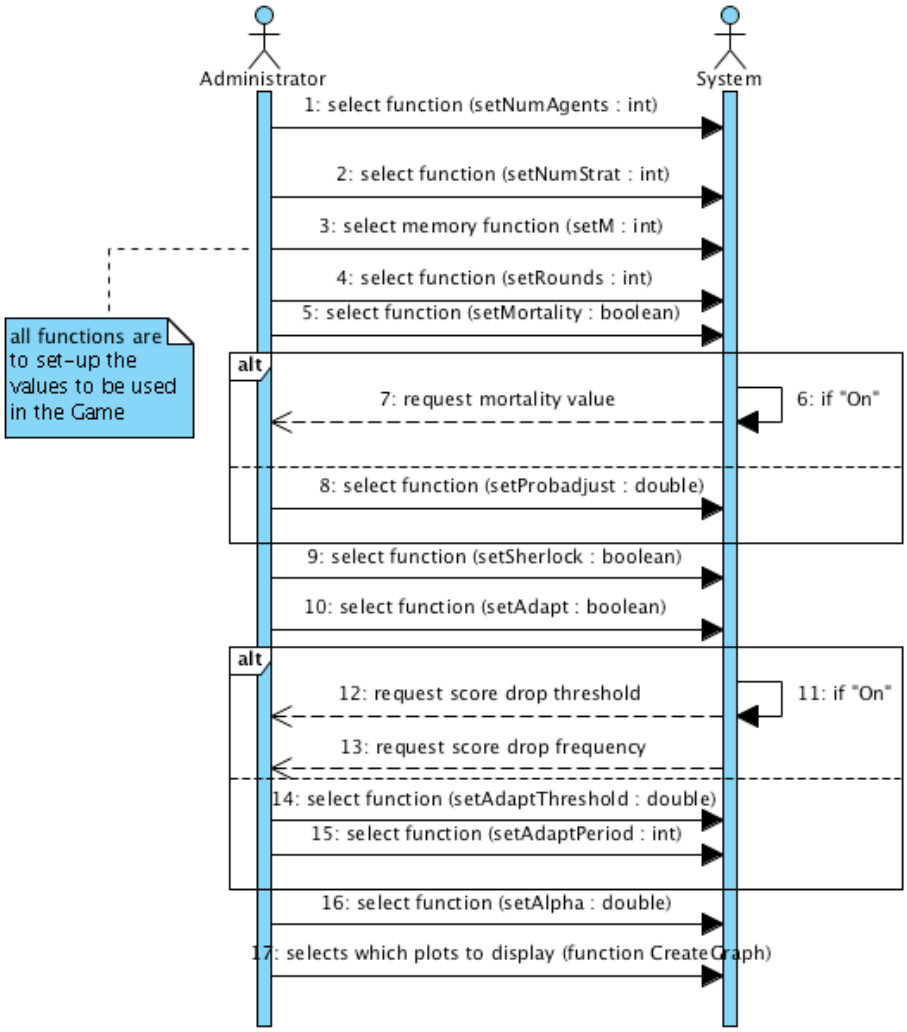
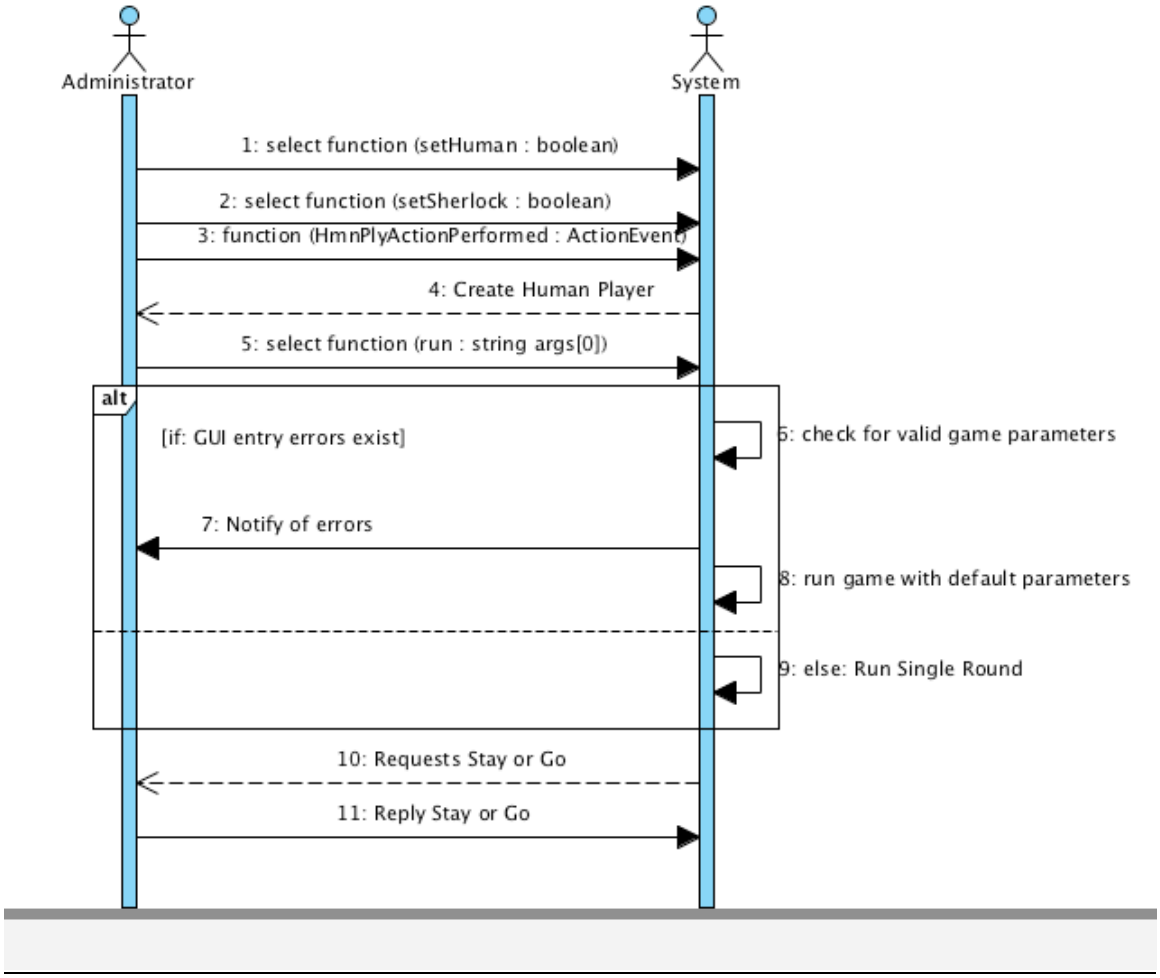
System Sequence Diagrams for Use Cases:

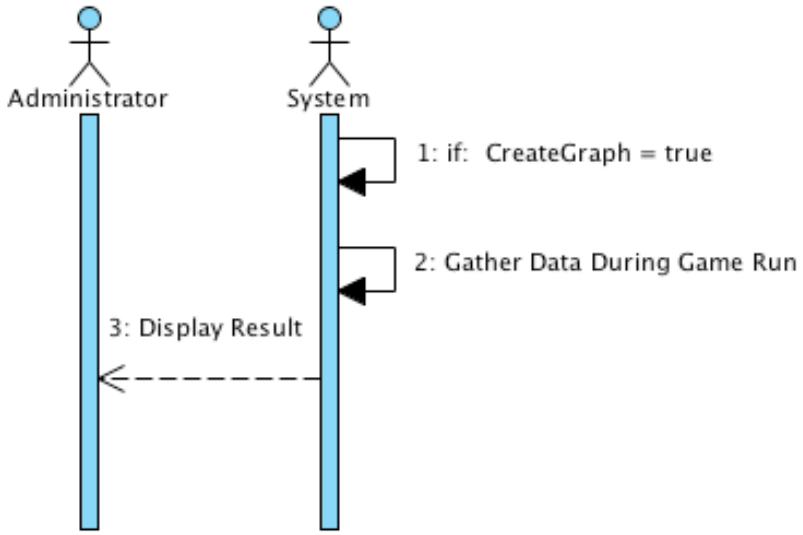
UC-1 ConfigureGame:



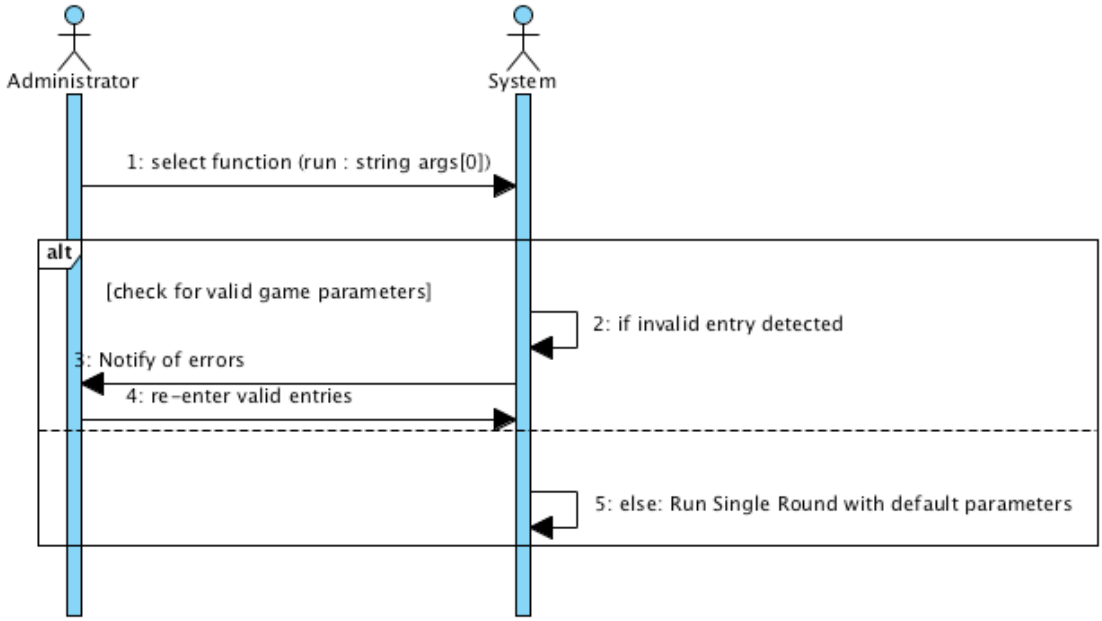
UC-2 PlayAlong:



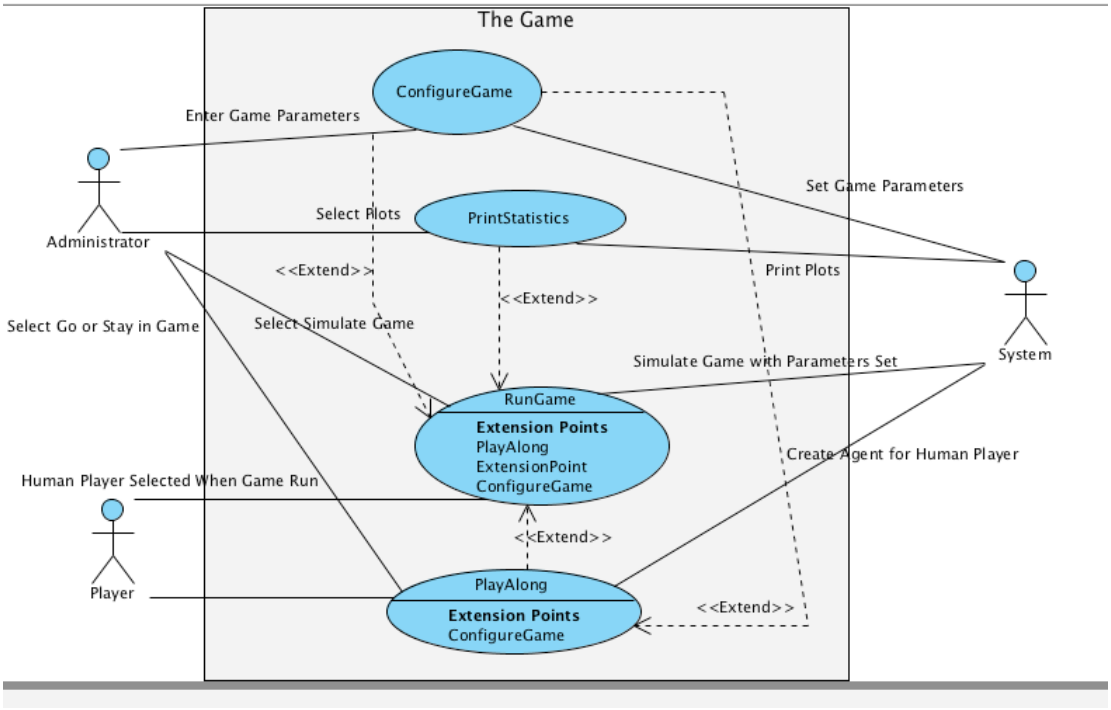
UC-3 PrintStatistics:



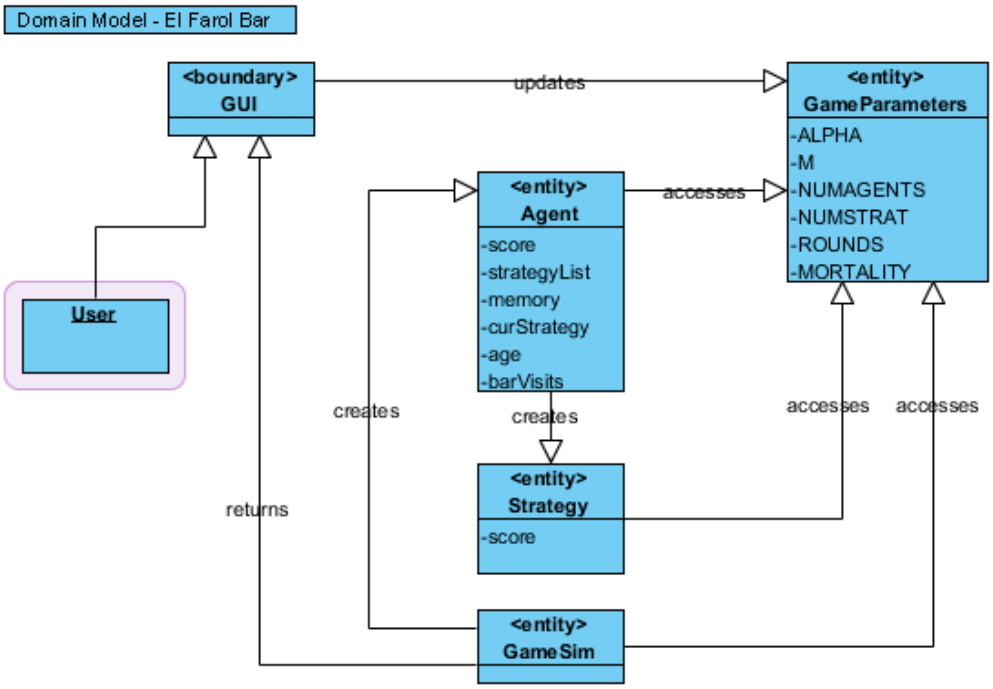
UC-4
RunGame:



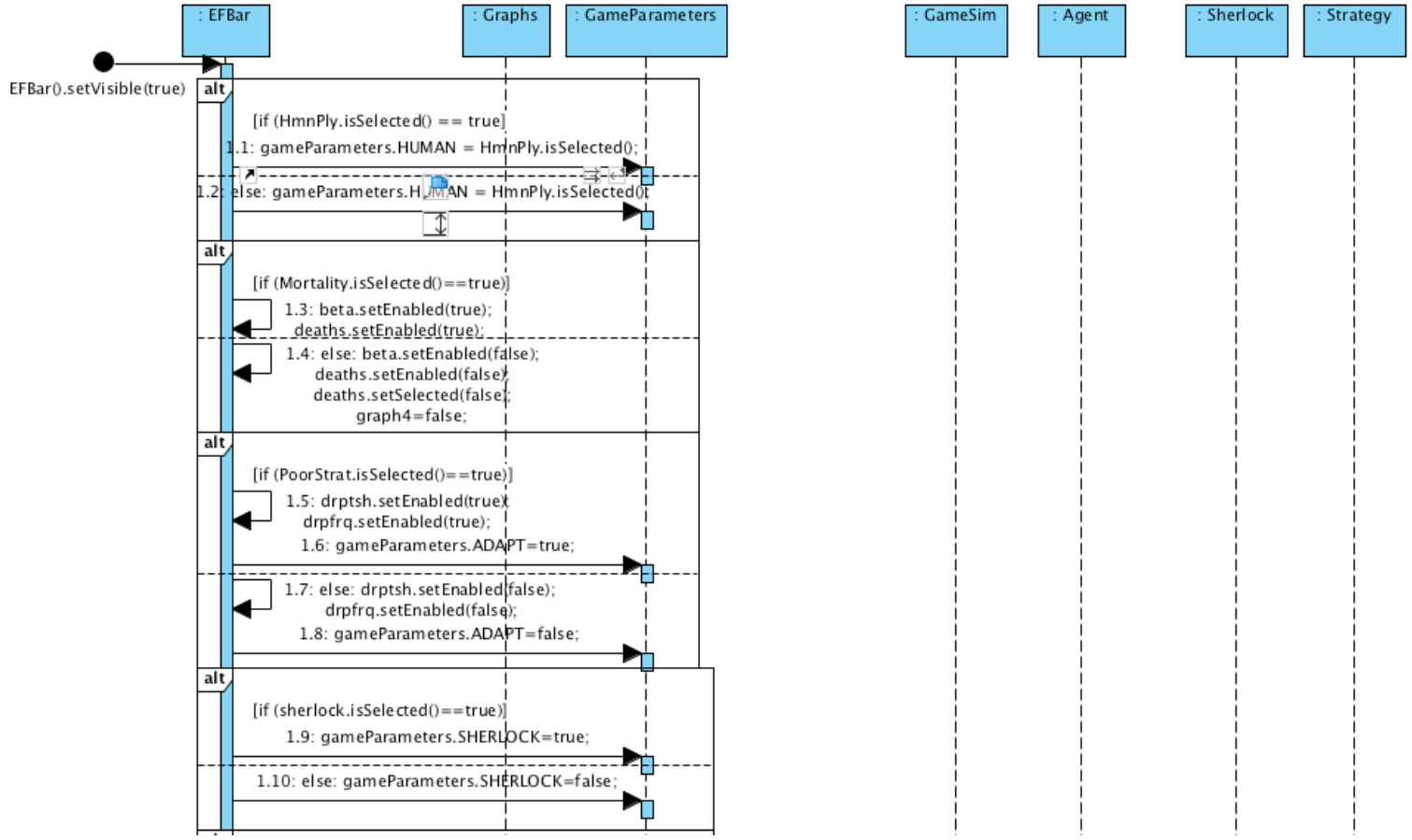
Use Case Diagram

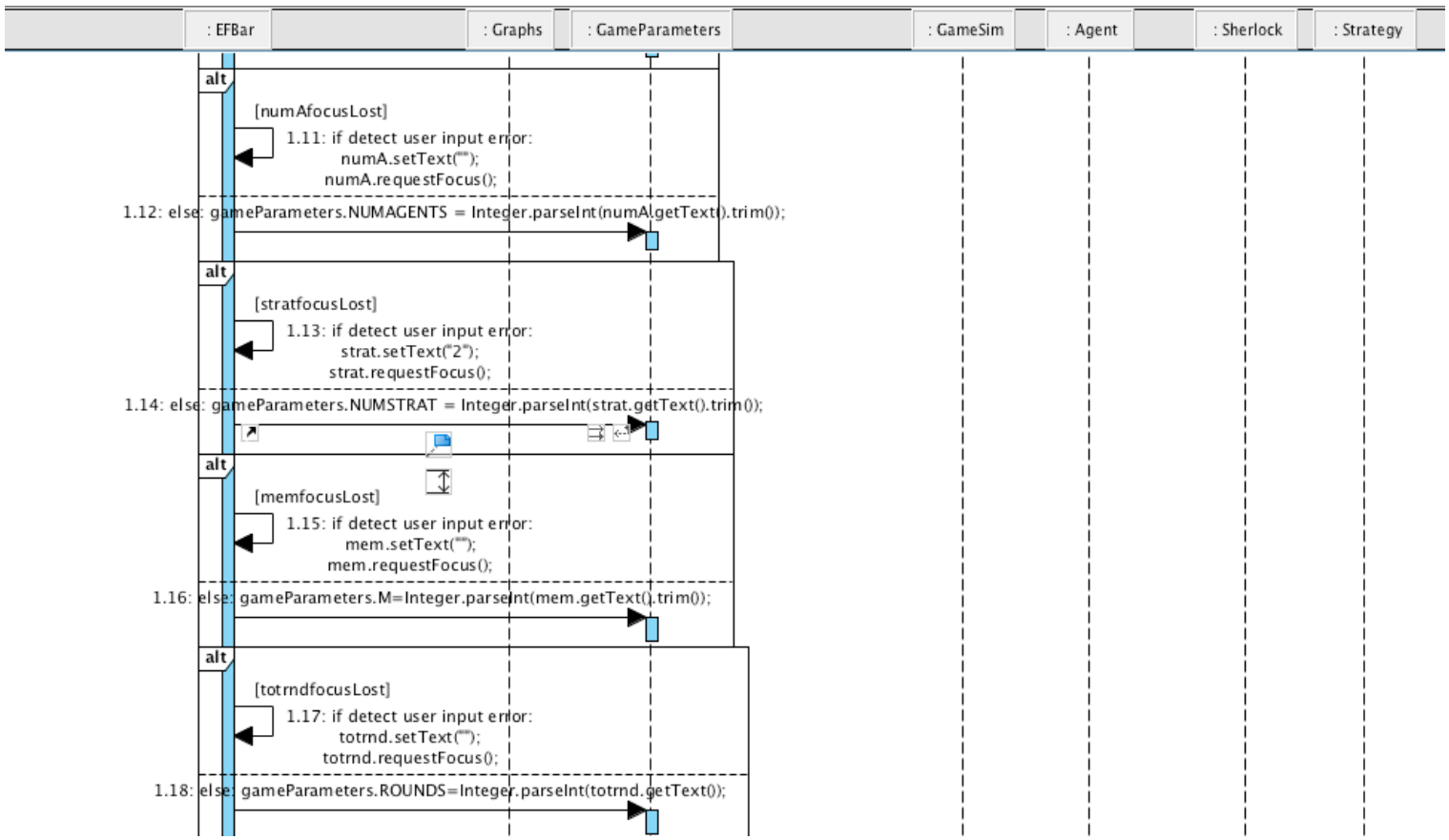


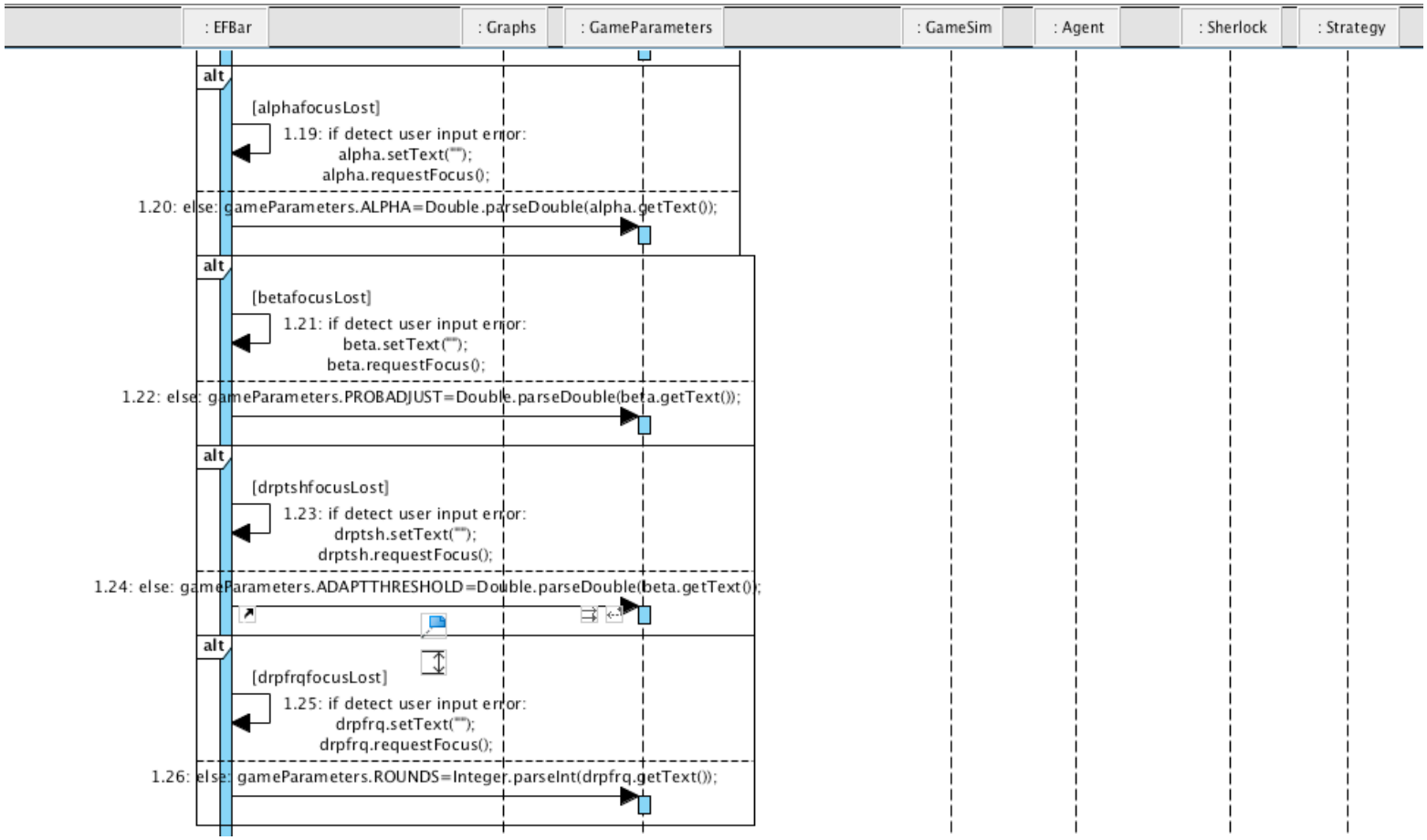
Domain Model

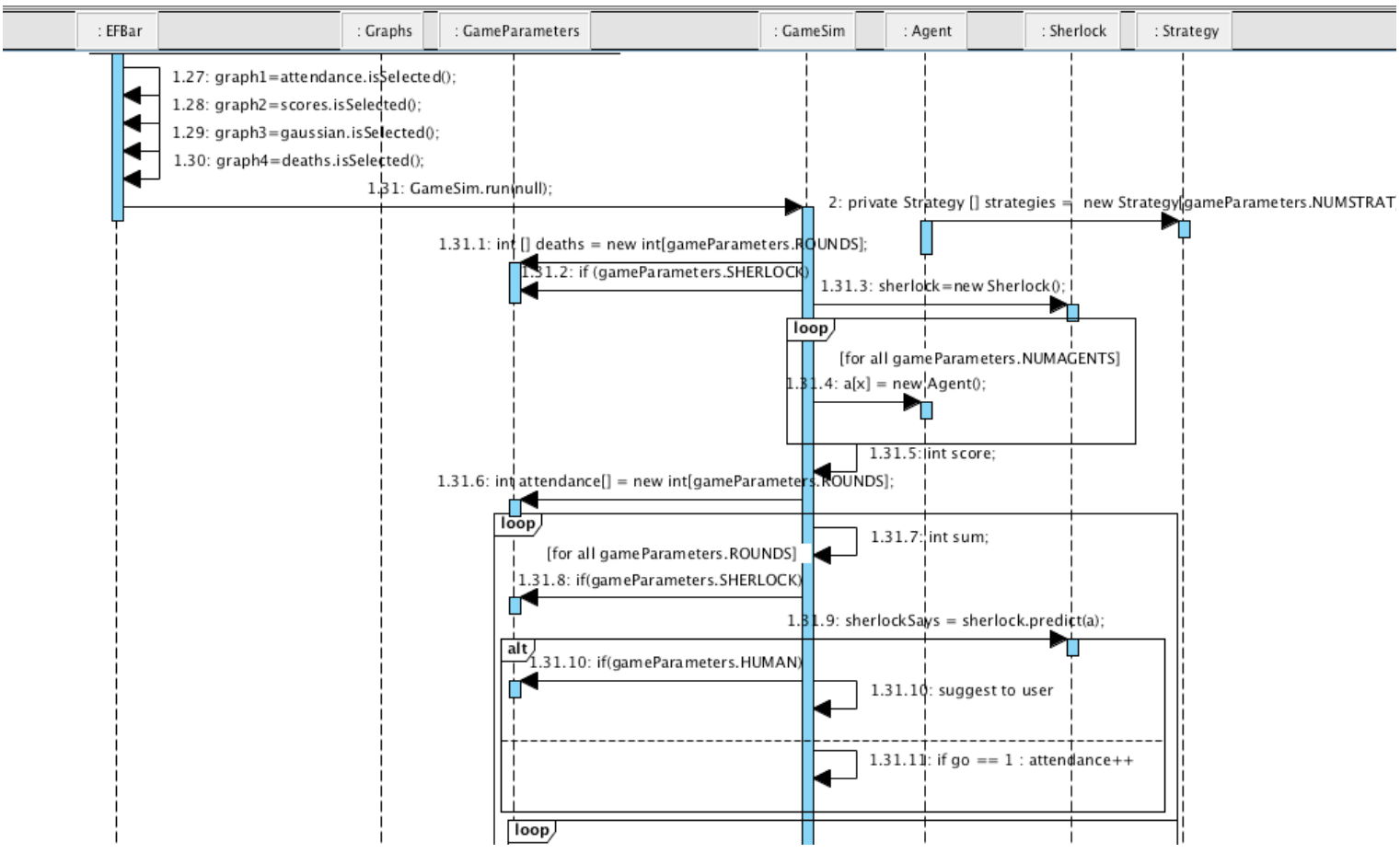


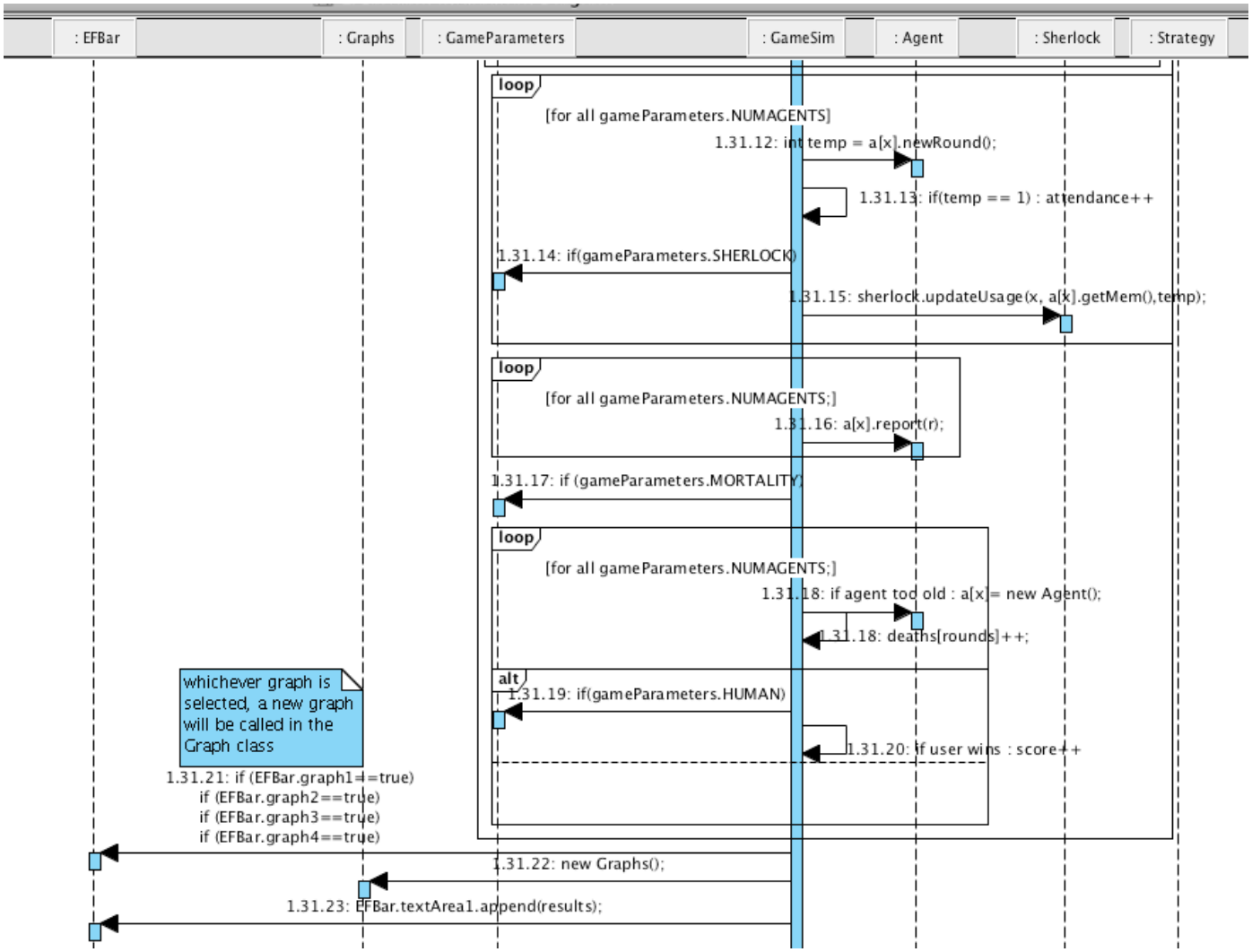
Full Interaction Diagram



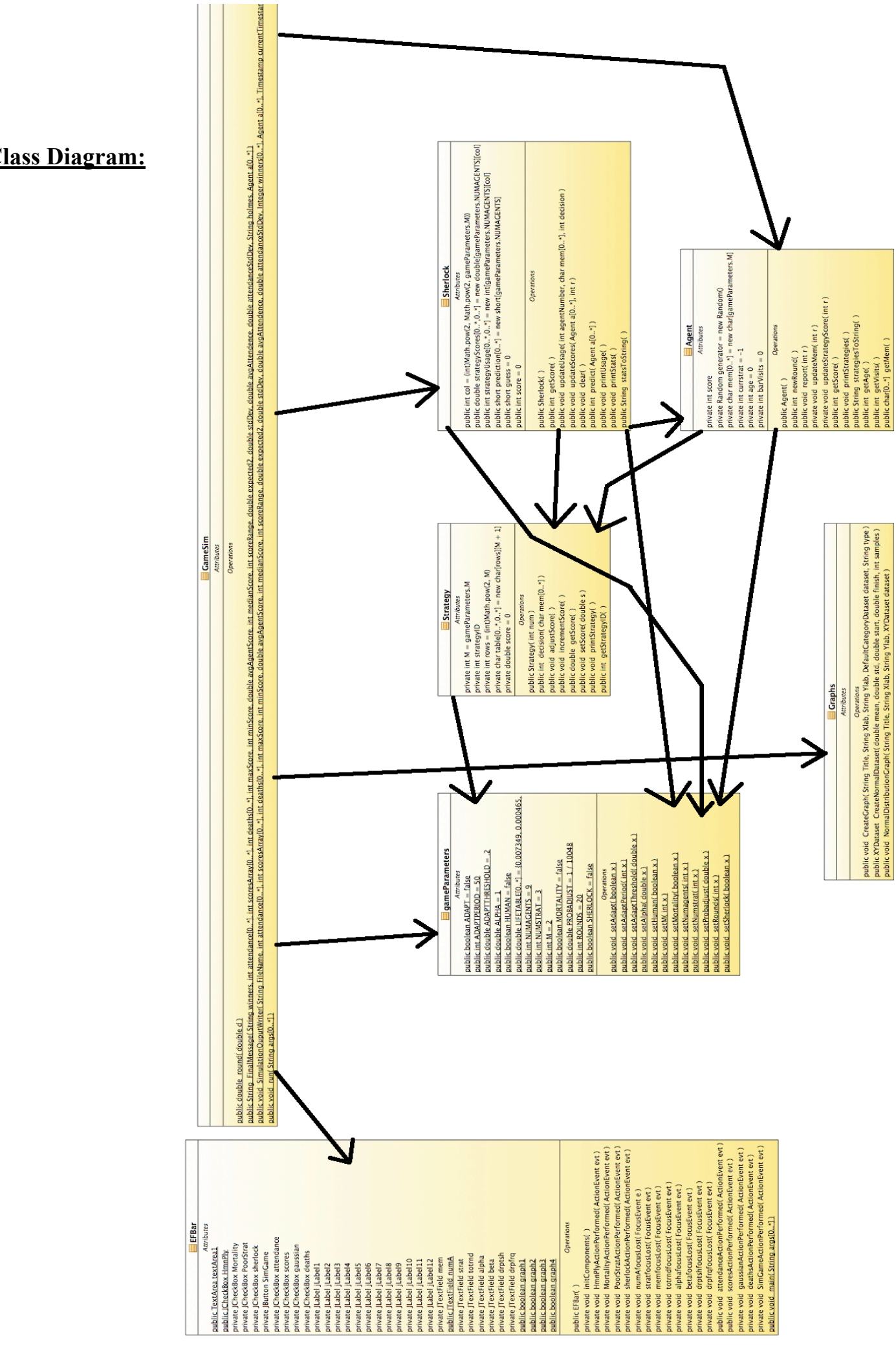



: EFBar
: Graphs
: GameParameters
: GameSim
: Agent
: Sherlock
: Strategy
alt
[alphafocusLost]
1.19: if detect user input error:
alpha.setText("");
alpha.requestFocus();
1.20: else: gameParameters.ALPHA=Double.parseDouble(alpha.getText());
alt
[betafocusLost]
1.21: if detect user input error:
beta.setText("");
beta.requestFocus();
1.22: else: gameParameters.PROBADJUST=Double.parseDouble(beta.getText());
alt
[drptshfocusLost]
1.23: if detect user input error:
drptsh.setText("");
drptsh.requestFocus();
1.24: else: gameParameters.ADAPTTHRESHOLD=Double.parseDouble(beta.getText());
alt
[drpfrqfocusLost]
1.25: if detect user input error:
drpfrq.setText("");
drpfrq.requestFocus();
1.26: else: gameParameters.ROUNDS=Integer.parseInt(drpfrq.getText());


: EFBar
: Graphs
: GameParameters
: GameSim
: Agent
: Sherlock
: Strategy
1.27: graph1=attendance.isSelected();
1.28: graph2=scores.isSelected();
1.29: graph3=gaussian.isSelected();
1.30: graph4=deaths.isSelected();
1.31: GameSim.run(null);
2: private Strategy [] strategies = new Strategy[gameParameters.NUMSTRAT
1.31.1: int [] deaths = new int[gameParameters.ROUNDS];
1.31.2: if (gameParameters.SHERLOCK)
1.31.3: sherlock=new Sherlock();
loop
[for all gameParameters.NUMAGENTS]
1.31.4: a[x] = new Agent();
1.31.5: int score;
1.31.6: int attendance[] = new int[gameParameters.ROUNDS];
loop
1.31.7: int sum;
[for all gameParameters.ROUNDS]
1.31.8: if(gameParameters.SHERLOCK)
1.31.9: sherlockSays = sherlock.predict(a);
alt
1.31.10: if(gameParameters.HUMAN)
1.31.10: suggest to user
1.31.11: if go == 1 : attendance++
loop


: EFBar
: Graphs
: GameParameters
: GameSim
: Agent
: Sherlock
: Strategy
loop
[for all gameParameters.NUMAGENTS]
1.31.12: int temp = a[x].newRound();
1.31.13: if(temp == 1) : attendance++
1.31.14: if(gameParameters.SHERLOCK)
1.31.15: sherlock.updateUsage(x, a[x].getMem(),temp);
loop
[for all gameParameters.NUMAGENTS;]
1.31.16: a[x].report(r);
1.31.17: if (gameParameters.MORTALITY)
loop
[for all gameParameters.NUMAGENTS;]
1.31.18: if agent too old : a[x]= new Agent();
1.31.18: deaths[rounds]++;
alt
1.31.19: if(gameParameters.HUMAN)
1.31.20: if user wins : score++
whichever graph is selected, a new graph will be called in the Graph class
1.31.21: if (EFBar.graph1==true)
if (EFBar.graph2==true)
if (EFBar.graph3==true)
if (EFBar.graph4==true)
1.31.22: new Graphs();
1.31.23: EFBar.textArea1.append(results);

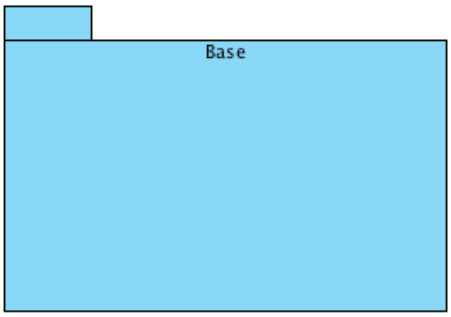
Class Diagram:



Subsystems

In the programs current state the subsystem diagram is minimal. The entire system is contained in a single package as illustrated below.

FIGURE 1



Assuming development on the program continued, the resulting subsystem diagram would look different. In order to make additions easily integrated the design would be split into three packages. The primary package would be called “Base” and would contain all the necessary portions of the system to run the most basic simulation. This includes the GameSim, Agents, Strategies, and EFBar. Another package “Visualization” would depend on Base. The purpose of this package is to handle the all graphs and illustrations to interpret the data visually. Finally a package called “Extensions” would be added, also depending on Base. The purpose of this package is to change the parameters of the simulation in GameParameters, and add new parameters in future builds. Extensions would be responsible for running the variants of the simulations under different conditions. This new revised structure of subsystems allows for the program to be updated more easily in the future. The subsystem diagram is below.

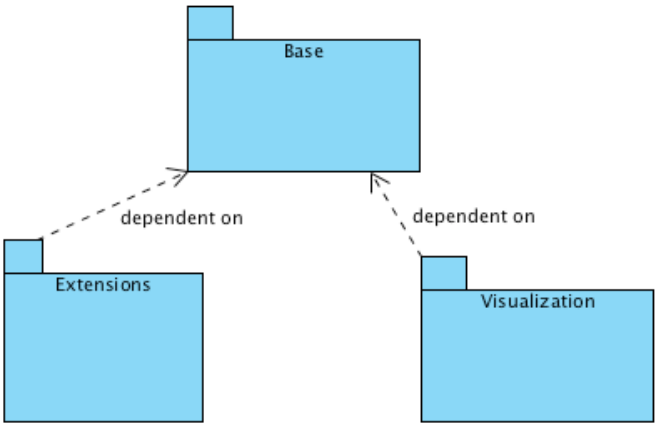



FIGURE 2