Distributed Artificial Intelligence and Intelligent Agents

ID2209

Final Project
Group 22

Ayushman Khazanchi
Vasigaran Senthilkumar
31.12.2021

Introduction

This project involved dealing with multiple agents and the interactions between the agents. It is also involved testing with various behaviours of the agents. The varying personality of each type of agents as a variable at specific locations and specific interactions were considered in the project. Also a global variable functionality was added and monitored for changes in the personality variable and functionality of each type of agent. We also conducted some experimental analysis by changing and controlling some variables in our model to see how they would affect the global mood of the simulation. The communication between the agents was performed using the FIPA protocol. Also the movement of the various agents was monitored on two dimension to get a clearer visualisation. We considered having a different agents with different interests in music meeting at different locations and randomly interacting with each other.

How to run

To run the final project, just run *project.gaml*

Species

Bar
This is a location agent for the guest agents to interact.

ConcertHall
This is a location agent for the guest agents to interact.

VirtualHall
This is a location agent for the guest agents to interact.

DestinyAgent
This agent is responsible for choosing two agents among the n number of participating guest agents RockGuest, RapGuest, PopGuest, ClassicalGuest, IndieGuest. The DestinyAgent is responsible for computing the overall average for the global mood based on different factors described in detail in the sections below.

GuestAgent (Parent Class)
This agent is parent class for the other guest agents RockGuest, RapGuest, PopGuest, ClassicalGuest, IndieGuest. The agent consists of reflexes and actions which are inherited by all the other child agents. The parent class describes reflexes for FIPA protocol communication as well as the logic for interaction between child agents.

RockGuest (Child of GuestAgent)
Each RockGuest agent is initialised with five attributes which are talkative, creative, shy, adventure and emotional.

Personality Traits	Talkative	Creative	Shy	Adventure	Emotional
Weightage	1.0	0.75	0	0.5	0.25

Location	Bar	ConcertHall	VirtualHall
Change in Personality	+5.0	-5.0	-2.5

RapGuest (Child of GuestAgent)
Each RapGuest agent is initialised with five attributes which are talkative, creative, shy, adventure and emotional. Their weightage is shown below -

Personality Traits	Talkative	Creative	Shy	Adventure	Emotional
Weightage	0.25	0	0.5	1	0.75

Location	Bar	ConcertHall	VirtualHall
Change in Personality	No change	No change	-2.5

PopGuest (Child of GuestAgent)
Each PopGuest agent is initialised with five attributes which are talkative, creative, shy, adventure and emotional. The attribute vales are chosen randomly. Then we calculate the weighted personality variable of the agent based on what attribute they like more.

Personality Traits	Talkative	Creative	Shy	Adventure	Emotional
Weightage	0.75	0.5	1.0	0	0.25

Location	Bar	ConcertHall	VirtualHall
Change in Personality	+5.0	-5.0	-2.5

ClassicalGuest (Child of GuestAgent)
Each ClassicalGuest agent is initialised with five attributes which are talkative, creative, shy, adventure and emotional. The attribute vales are chosen randomly. Then we calculate the weighted personality variable of the agent based on what attribute they like more.

Personality Traits	Talkative	Creative	Shy	Adventure	Emotional
Weightage	0.5	1.0	0.75	0.25	0

Location	Bar	ConcertHall	VirtualHall
Change in Personality	-5.0	+5.0	-2.5

IndieGuest (Child of GuestAgent)
Each IndieGuest agent is initialised with five attributes which are talkative, creative, shy, adventure and emotional. The attribute vales are chosen randomly. Then we calculate the weighted personality variable of the agent based on what attribute they like more.

Personality Traits	Talkative	Creative	Shy	Adventure	Emotional
Weightage	0	0.25	1.0	0.75	0.5

Location	Bar	ConcertHall	VirtualHall
Change in Personality	-5.0	+5.0	-2.5

Implementation Overview

Agent Personalities

On initialization, the different types of guest agents all inherit the properties and behavior of the parent specie GuestAgent. Each agent type then initializes its own "personality traits" or "attributes" with some random values between 0 and 10.0. These attributes are then combined with different weightages for each guest type (shown above) and the result of that is the unique "personality" of our guests. In our simulation, as in real life, the personalities of the guests are also affected by their location. Sometimes a guest who is in a bar may not have as pleasant a personality as when they're in the ambience of a concert hall. This is factored into our simulation as the location either increases, drops, or does not affect the personality depending on the guest type. The above computation forms the basis of our ruleset for creating the personalities of our guest agents.

A RockGuest initialization and personality creation is shown below as an example.

```
  init {
    talkative <- rnd(0,10.0); // start with random traits
    creative <- rnd(0,10.0);
    shy <- rnd(0,10.0);
    adventure <- rnd(0,10.0);
    emotional <- rnd(0,10.0);
    mood <- rnd(0,10); // start with a random mood
    // weighted personalities based on what attributes they like more
        //  weightage based on scale of 0 to 1
    myPersonality <- (talkative * 1) + (creative * 0.75) + (shy * 0) + (adventure * 0.5) + (emotional
* 0.25);
    barPersonality <- myPersonality+5.0;    // Personality increases in bar
    concertHallPersonality <- myPersonality-5.0;
    virtualHallPersonality <- myPersonality-2.5;
  }
```

Destiny

Our model starts with 20 agents of each of the following types of guests: RockGuest, PopGuest, RapGuest, ClassicalGuest, IndieGuest. There is only one DestinyAgent which is responsible for selecting the other guest agents that interact at various locations. When the simulation is run the DestinyAgent chooses two guest agents at random to meet at the bar, concert hall, and virtual hall. The Destiny Agent ensures that those already selected to go to one location are not selected again for a different location. Once the agents are selected, the information is relayed to them by the DestinyAgent via the "ask" function. The agents then stop "wandering" and start "traveling" towards their destination. The destination "target" is informed to each agent as part of the "ask" request from the DestinyAgent.

Within the DestinyAgent, the reflexes responsible for arranging meetups are arrangeBarMeeting, arrangeConcertMeeting, and arrangeVirtualMeeting. Within the DestinyAgent is also the reflex that computes the average global mood of the simulation.

Travel, Arrival, and Finding Neighbours

The agents that are selected to meet at various locations start traveling to their locations; however, nothing more happens until they arrive. We wanted to simulate a real-life situation of two people meeting at a bar or a concert hall or a virtual hall and starting a conversation after they meet. You cannot start a conversation with someone who has not arrived yet and so, in our simulation as well, if one of our agents arrives before the other agent they wait until the other agent arrives before starting the conversation. Once an agent reaches their destination coordinates, they will search for any neighbouring agents that are also at the location. This is done in the **findNeighbor** function as shown below

```
reflex findNeighbor when: (location = targetPoint) and (travel = true) {
	list<agent> neighbors <- agents at_distance(1); // finds any agent located at a distance <= 1 from the caller agent. This will only happen if an agent is already at the same target point
	agent neighbor;
	loop n over: neighbors {
	  if (n.name != 'ConcertHall0') and (n.name != 'Bar0') and (n.name != 'VirtualHall0') {
	    neighbor <- n;
	  }
	}
```

Once the second agent arrives, they are "found" by the first agent and a conversation is started where they exchange their personalities based on the location they are interacting in. If they're at a bar, they "show up" with their *barPersonality*. If they're at a concert hall, they exchange their *concertHallPersonality*. After each interaction the mood of the interacting agents either goes up or down depending on the personality of the agent with whom they have interacted.

Conversation

Once a neighbour is found the agent that arrived first at the destination will start a conversation with the neighbouring agent. The first agent sends a request to the second agent which includes the first agent's location-specific personality as shown below.

```
        if (target = 'bar') {
        do start_conversation (to::list(neighbor),protocol::'fipa-contract-
net',performative::'request',contents::[barPersonality]);
      } else if (target = 'concerthall') {
        do start_conversation (to::list(neighbor),protocol::'fipa-contract-
net',performative::'request',contents::[concertHallPersonality]);
      } else {
        do start_conversation (to::list(neighbor), protocol::'fipa-contract-
net',performative::'request',contents::[virtualHallPersonality]);
      }
```

The second agent receives this request and computes its mood based on the personality received. Then it informs the first agent of its own personality as well.

```
        if (target = 'bar') {
        // compute mood based on personality received
        do changeMood(barPersonality, initiatorPersonality);
        // respond with your own personality
        do inform with: (message: requestFromInitiator, contents: [barPersonality]);
}
```

The first agent receives the personality and computes its mood as well.

```
        if (target = 'bar') {
        // compute mood based on personality received
        do changeMood(barPersonality, guestPersonality);
      } else if (target = 'concerthall') {
        // compute mood based on personality received
        do changeMood(concertHallPersonality, guestPersonality);
      } else {
        do changeMood(virtualHallPersonality, guestPersonality);
      }
```

Thus, we have both agents involved in the interaction and both moods altered based on personalities of each other. Additionally, because the personalities are location-specific the mood changes can vary based on the location.

Mood Computation and Global Mood

Our mood computation is simple but quite important to the simulation. As in real life, we're maintaining a "mood" value for each guest agent. When the guest agents are initialized, they are all initialized with random mood values between 1 and 10.0. When two agents interact, their moods change. This is based on a simple formula that we used where if an agent meets someone with a "greater" personality than them, their mood goes up, but if they meet someone who has a "lesser" personality than them, their mood drops. If an agent meets someone with the exact same personality

as them, moods of both agents increase by a factor of two. The function for mood change is shown below and is inherited by all guest agents from the parent class.

```
action changeMood (float personality, float otherAgentPersonality) {
    // compute mood based on personality received
    if (personality < otherAgentPersonality) {
        mood <- mood + 1; // guest is greater personality, mood rises
    } else if (personality > otherAgentPersonality) {
        mood <- mood - 1; // guest is lower personality, mood falls
    } else {
        mood <- mood + 2; // guest and I are exact matches, mood rises double
    }
}
```

Mood change for the interacting agents is tracked after each interaction and, as a result, is something that changes over time as the simulation goes on. We can then measure a "global mood" of our simulation that is an average of all the individual moods of each of our agents. We keep track of all our interactions using the *globalCycles* variable which is then used to compute the average global mood as shown below. The *computeGlobalMood* reflex is called after each interaction is finished.

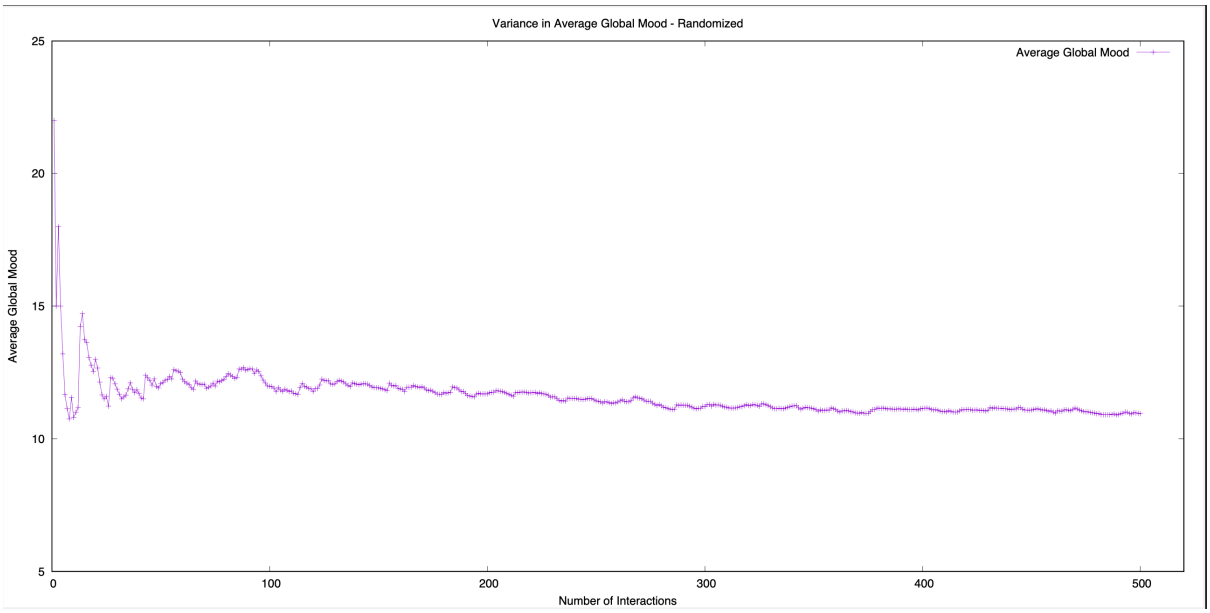
```
reflex computeGlobalMood when: (barMeet = "mood" or concertMeet = "mood" or virtualMeet =
"mood"){

    globalCycles <- globalCycles + 1; // keeps overall counter of global mood computations in order
to get average
    averageGlobalMood <- globalMood / globalCycles;
    write 'Average global mood after interaction ' + globalCycles + ': ' + averageGlobalMood;
    ...
}
```

As part of monitoring average global mood, we ran different experiments. The global mood is calculated after each conversation is completed. We charted the value of average global mood (*averageGlobalMood*) over the total number of interactions (*globalCycles*) to see how the average global mood changes. The global mood is evaluated for >500 interaction cycles for each type of implementation and the results are plotted using gnuplot software.

Experiments and Discussion

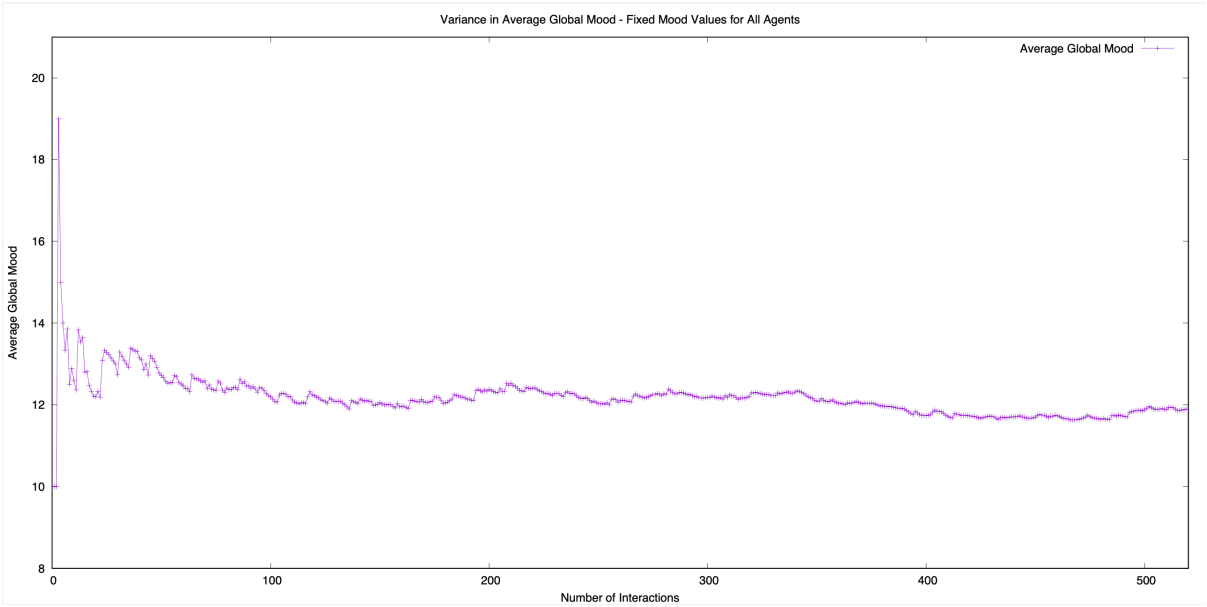
Random Moods and Random Personalities

The setup that we have above is based on randomization of both initial mood and initial personality traits. Two agents of the same type (ex, ClassicalAgent) can meet and still have very different personality numbers because of the random initial values they start with. This is as close as we can say to real-world interactions as often people can have different moods and different personalities at different times and at different locations. When we monitor global average mood in this method we see that it mostly tends to remain consistent (this is because in our case, unless the personalities are an exact match, after each interaction one agent's mood will decrease and one agent's mood will increase).



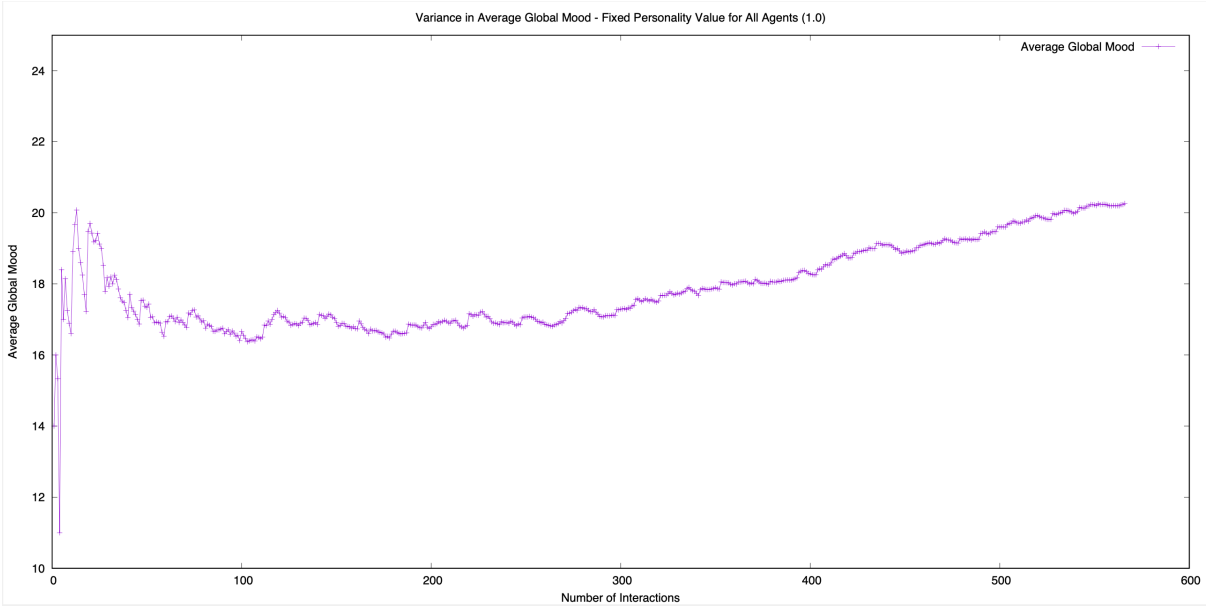
Fixed Mood and Random Personalities

In this experiment we started with a fixed initial value for the mood of all agents. We started all agents with a fixed value of 5 for mood. We maintained the randomness of the personalities and got a similar graph where the average global mood doesn't change very much once settled. As a result, we can say that the initial random mood values of the agents don't affect the simulation too much in the long run.



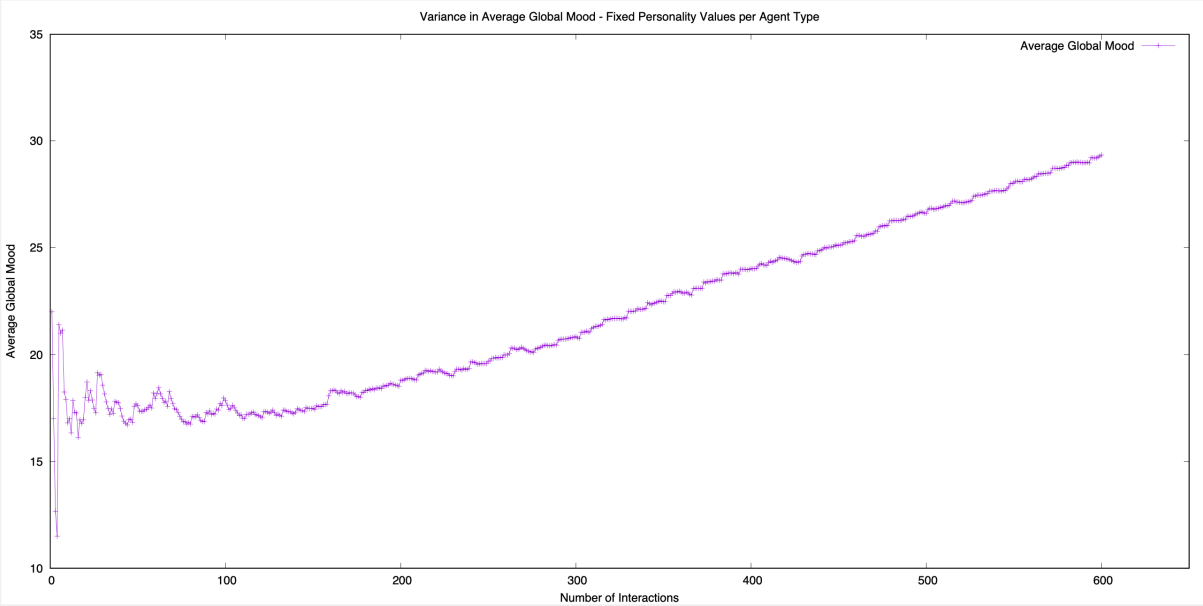
Exact Same Personality Traits

As we have seen it is mostly the personalities that have an effect on the average global mood. As a result, in this experiment we started with a fixed initial value for all the personality traits of all the agents. All agents had all their personality traits set to 1.0. The only difference then between each agent type was the weightage of the personality traits that were different for each type of agent. This raises an interesting scenario in our mood computation because now two agents of the same type (ex: ClassicalAgent1 and ClassicalAgent2) will have the same exact personality when they meet. In this case, the mood of both agents will increase two-fold as they're a "perfect fit". As more and more agents of the same type meet with the same exact personalities we can extrapolate that overall average global mood will rise over time. We can see this reflected in the chart as well when we run it for >500 cycles.



Fixed Personality Traits and Weighted Personalities

In this experiment we started with a fixed initial value for all the personality traits of each type of agent. Each type of agent had their personality traits set to a fixed value. Each type of agent also had their weighted personality set. Even in this implementation two agents of the same type will have the exact same personality but two agents of different types might still have different personalities (due to the weightage on each trait being different). As a result, the graph for average global mood for this experiment also tends to rise upward due to two-fold increase in mood for each interaction that is between agents of similar types.



Discussion / Conclusion

We enjoyed working on the simulation. The final project brought together the overall concepts we've learned about agent communication and behaviour as well as the FIPA communication protocol. Another interesting thing we learned during this was how to wait for agents to show up to the venue and how to find nearby agents at a given distance. Computing and observing a global value based on individual values from the agents also made the project quite interesting as we could conduct different experiments and observe the changes in our global mood variable by changing some factors of the experiment. Overall we enjoyed creating this implementation and tried to make it as close to real-world simulation as we could.