ID2209 – Distributed Artificial Intelligence and Intelligent Agents

Final project – Behavior of different agents

Group 10

Nico Catalano, Giulio Marcon

23/12/2019

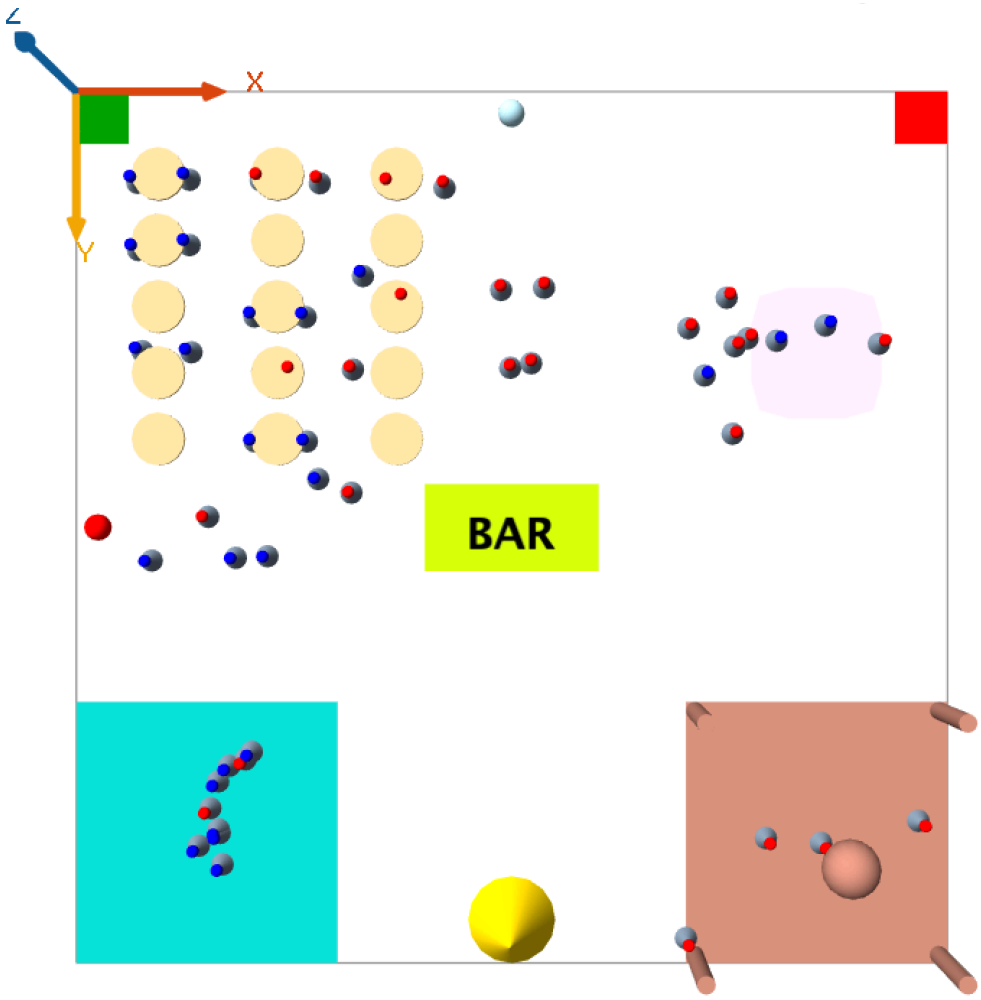
Base Part

In this final project we were asked to simulate an environment where different types of agents communicate and interact with each other using GAML and the GAMA Platform.

In the basic task we had to implement at least five different types of agents, using at least 50 agents in our scenario.
The agents have at least one different set of rules on how they interact with other types and 3 personal traits that affect these rules.
The agents will be able to meet in two different types of places and will communicate entirely with the FIPA protocol.
The simulation can be continuously run.

How to run

1. Run GAMA 1.8.0
2. Import all the files in the folder “base_creative”
3. Press the green button “Festival” in the top left corner to start the simulation



Species

Guest

The species guess is by far the most complex in our environment since it represents the actual attendees of our festival simulation, and it's where most of the conversations between agents are happening. All the code can be found inside the Guest.gaml file.
Inside a Guest we defined different traits:

- *chill2dace*: this is the trait that we use to indicate whether an agent wants to dance or to chill. It's implemented as a float value between 0 and 1. When this value is greater than the *danceTrashold* the Guest will go to the Dance floor, otherwise it will go to the Chill Zone. *chill2dace* increases over time with the reflex *updateChill2Dance* and can also decrease by talking to other agents.

- *Thirsty* is a value used to check if the agent wants to go to the Bar and drink. This value updates through the reflex *updateThirsty*, which increases the value by a small random amount

- *Talkative* is used to indicate how much a Guest is willing to talk to other agents and it is used in multiple reflexes where an agent is trying to start a conversation

- *Love:* this variable represents how much an agent wants to meet a soulmate, and it is used together with *loveTrashold*, *gender* and *desiredLoveMateGender* in reflexes that take place inside the Tinder Area.

- *Drunkness* is used to check how much alcohol a Guest drunk during the simulation. If *drunkness* exceeds a certain value the guest is kicked out from the festival by the Security Guard.

Other variables are used for other parts of our simulation and will be described if and when necessary.

Our guests are divided into either **Chill** or **Party**: the first one will tend to prefer the Chill Area and its *chill2dance* parameter will increase much slower compared to the Party Guest.

To better represent these two types of agents we wrote two distinct species, *ChillGuest* and *PartyGuest*, both children of the *Guest* specie.

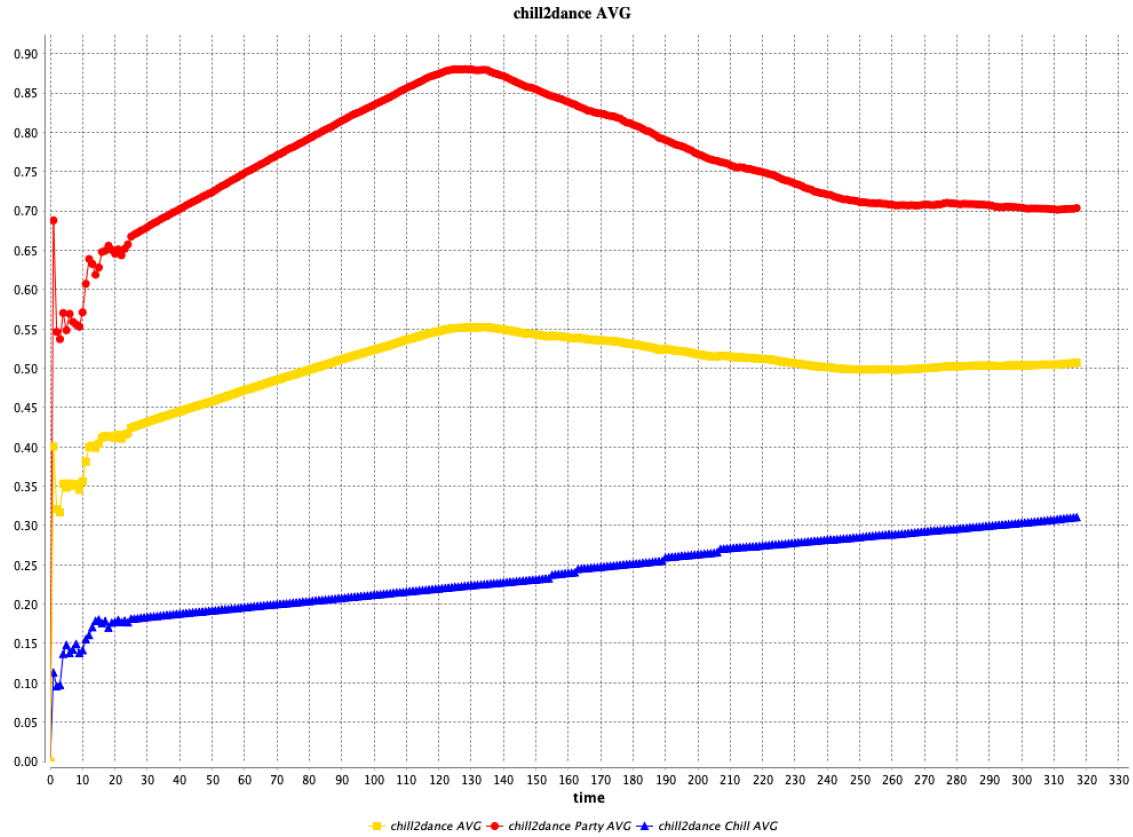


Figure 1 Graph that show how the chill2dance parameter changes over time, showing both Chill (blue) and Party (red) guests and the average in yellow

Files Structure

The codebase is divided in multiple files in order to improve maintainability and allow simultaneous development of different agents.

The files are:

- finalProject_0.gaml : this file contains all the global variables and costants. It also contains the experiment, and it's responsible for spawning the guest as well as the other agents.

- ATM.gaml : this file contains the agent ATM and its behaviour. When a guest needs money, it just goes near the ATM an asks for a precise amount of cash to withdraw.

- Bar.gaml : this file contains the agent Bar and its behaviour. The bar has a menu list containing the prices and alcohol percentage of each drink.

- Security.gaml : this file contains the agent Securty and its behaviour. The security is in charge of kick off the bad behaving guest when reported.

- Supplier.gaml : this file contains the agent Supplier that supplies the bar when it finishes its stock.

- Guest.gaml: this file contains the father agent Guest and the 2 children classes PartyGuest and ChillGuest. All the social interactions of the guest are coded inside this file as well as the behaviour changing rules.

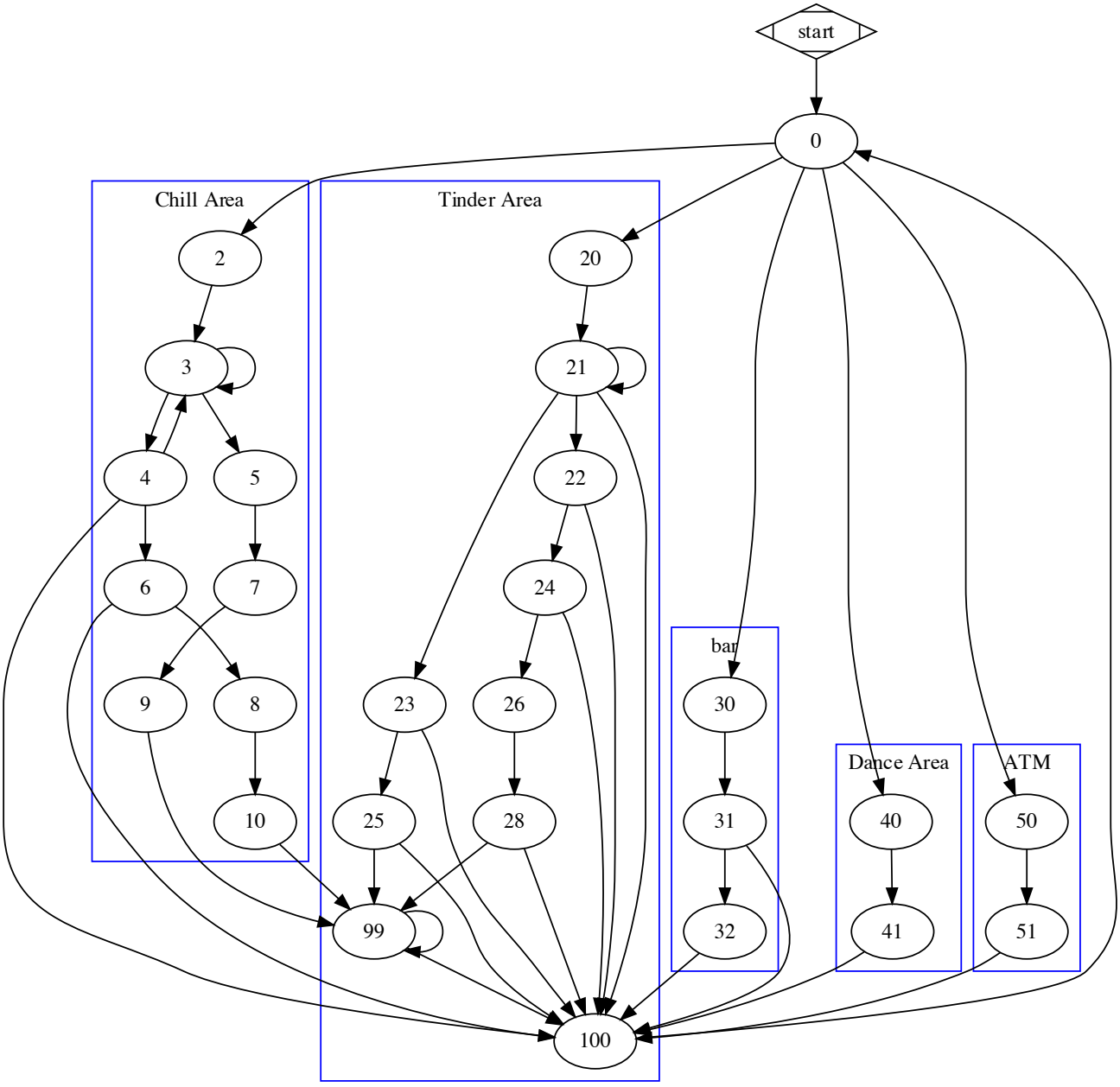
- Misc.gaml: this file contains the descriptions of all the objects in the simulation that have only graphical purposes such as the tables, the dancefloor or the chill area and tinder area.

Finite State Machine

The entire Guest specie functions as a Finite State Machine, where each state represents a specific situation that can or will happen during our simulation. This choice was made because it guarantees a robust implementation of our agents, while still keeping the code relatively clear to understand.

Unfortunately, GAML does not support any enumerative mechanism to lable the states with significative names, so each status is identified by an integer number, and in the following the description of each status is reported.

We have maintained the rule to give even number to status relative to the initiator of conversation while giving odd numbers to the status in that performed response to the initiator messages.



State 0 is the first state in which our agents are “spawned” in. Here each agent will evaluate its own traits in relation to the various thresholds and go, thanks to reflexes such as *goToChillArea*, *goToTinderArea*, *goToBarLocation* and *goToDanceArea*, to the area that best fit the Guest parameters, changing the status from 0 to a new one.

State 99 is a loop status that makes the guest stay at the table for a certain number of iterations (*WAITING_ITERATIONS*). This is used to better represent the interaction between people. When both agents find an intresting chat parther they will stay at the table talking for a while before going back to the festival activities.

State 100 is used to reset each agent to the state 0 if deemed necessary and it represents the end of all the conversations with other types of agents.

Chill Area States

When an agent's *chill2dance* is below its own *danceTrashold*, that Guest's status is set to **2**, and with *goToChillArea* the guest will set its own targetPoint to the *ChillLocation*.

Once arrived, *arrivedAtChillArea* is triggered and the status is set to **3**. The Guest is now executing *randomWaitChillArea*, where the agent is wandering while looking for potential conversations with other Guests. A conversation is not always initiated thanks to the *flip* statement that prevents Guests with a low *talkative* to talk as much as others.

If the *flip* returns true, the agent will go to **State 4** and start looking for potential neighbours to talk to in the reflex *startConverationChill*. Once a partner is found, the first agent will be the initiator of the conversation, and it will go to **State 6** and communicate the table chosen to go to talk, more on this later.

Other agents in the Chill Area that are still in status 3 will keep executing *catchConversationStart*, a reflex design to listen to potential initiators and begin a new conversation if approached.
If a guest is approached it will go to **State 5** and send a confirmation to the agent that started talking.

The reflex *gotACKfromChillGues* triggers and sets the initiator's destination to the chosen table, once there the status will be set to **8** with *initiatoreReachedTableChill*.

With *targetGoesToTableChill* the approached will also go to the same table as the other Guest and set its status to **7**, once reached the destination the status will become **9**. Now that both agents are at the table, the initiator will communicate its own *chill2dance* and go to **State 10**, all in the reflex *sendC2dToTarget*: this recreates a situation where the approacher is trying to convince the other guest to either go chill or dancing. The receiver now executes *gotC2D*, where a new *chill2dance* is calculated using the logic that we developed.

Tinder Area States

The second area where the Guests can meet is the Tinder Area, called like these because here the agents try to approach other guests based on what gender they are attracted to.

Once the *love* trait of a guest goes above the *loveTrashold*, that agents will go to the Tinder Area and set its status to **20**, reached the destination the status will be set to **21**.

At State 21 the agent will execute *lookAround*, where with the same mechanism that involves *talkative* previously described will be used to look for a partner. If the *flip* returns true, the agent will go to **State 22** and execute *lookingForSoulMate*.
The algorithm works similarly to the one developed for the Chill Area, the only difference being that here we have more messages and states because we need to communicate each gender to check if it's what the approacher is looking for.

Once 2 agents agree that they can talk, they will go to the booked table and start talking, exchanging the *chill2dance* parameter the same way as before.

Bar States

Once in **State 0** each need is evaluated, if the *thirsty* parameter's threshold is reached, the status is set to **State 30** and the target point to the bar location.

When the guest is near to the bar, it will go to **State 31** where it will ask the menu to the bartender while providing its level of drunkenness.

The bartender will reply to the guest with the lists of available drinks and the relative prices only if the guest is not drunk. If that is the case the bar will contact the security which will kick off the guest.

When the guest is **State 31** and has received both the lists it will decide which drink to buy, and check if it can afford it. In the positive case it will update its wallet, communicate it to the bar and go to **State 32.**
Differently the flag *needCash* is set to True, and the **State to 100.** In this way the agent will go to the ATM and then it still has the need to drink and go back to the bar.

In **State 32** the drink is served, the guest will update its drunkenness, it thirsty and go back to the festival activities.

ATM States

As previously said, if an agent needs to withdraw money it will go to the ATM location in **State 50** and will execute *atATM* once arrived do initiate the conversation with the other agent. The Guest sends a request to the ATM asking for a set amount of money. This amount is then added in **State 51** to the Guest's wallet, the status is set to 100 and the agent will go back to the previous desire.

Dance States

Agents at **State 41** will dance (using *wander*) at the Dance floor. To simulate Guests getting tired over time, we decrease *chill2dance* by a small amount at each iteration, this way one the value is lower than the threshold the agent will go to the Chill Area.

Other Species

Supplier

The supplier is in default in the **State 0**, which is its “waiting” status.

When it receives an inform message it will set its targetPoint to BarLocation and **State to 1**. Once arrived there it will provide the required beverage and quantity to the bar, set its targetPoint to the resting position and **State to 2**.

When the supplier reaches its resting position will reset also the **State to 0**.

Security

The security stays in its resting position in **State to 0**.

When it receives a report, it will set the reported agent as its target and move to the agent. The status is updated to **State 1**.

When the security reaches its target, it will communicate it to the exit location and move to **State 2**.

During **State 2** the security is escorting the agent to the exit, when it reaches the exit location it just goes back to resting status and position (**State 0**).

Creative Part

A requirement for this project was to simulate more than 50 agents communicating at the same time. To better show how these conversations take place we decided for our creative part to implement tables that Guests can book and go to talk more privately. This not only allows us to better recreate how a real-world conversation might happen, but also understand how the agents approach each other and conduct the conversation.

An agent, either in the Chill area or Tinder area, that wants to start a new conversation needs to first check if one of the tables is available. We’re doing this by checking if a false value is present inside *tableBookings*, and if it is, the agent will randomly choose one of the available tables to go to talk.

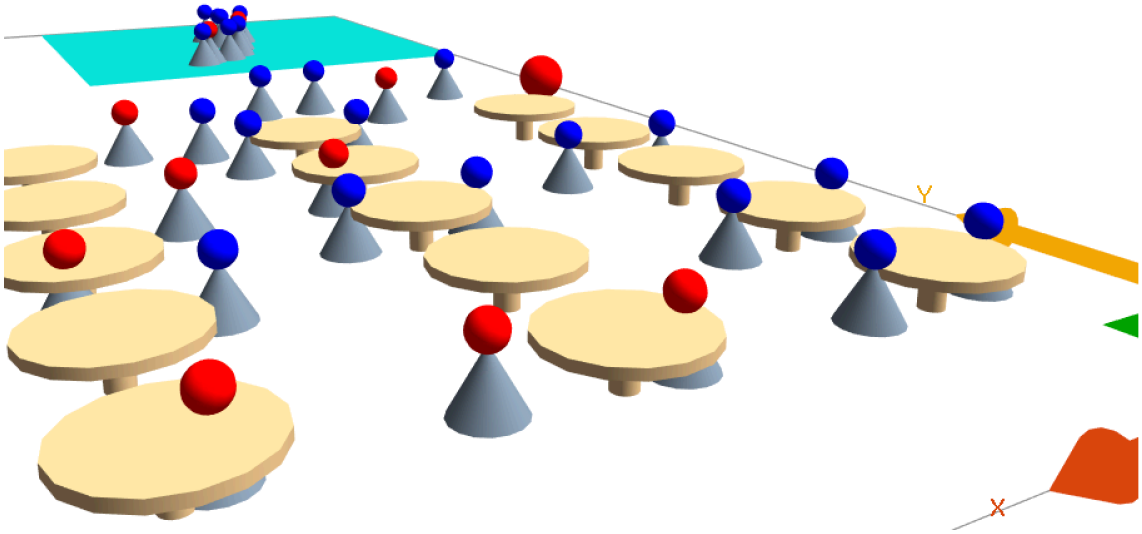


Figure 2 Agents having a conversations at the tables