Elisa Gonzalez Boix (email:egonzale@vub.ac.be, office:10F731)
Jorge Vallejos (email:jvallejo@vub.ac.be, office:10F724)

7 Distributed Programming in AmbientTalk with Tuples: TOTAM

Ambienttalk's tutorial and language reference are available at http://soft.vub.ac.be/amop/. The lab session material is available at http://soft.vub.ac.be/amop/teaching/dmpp

7.1 Idea

The purpose of this exercise is to get you more familiar with distributed programming using a tuple space-based model. To this end, you will implement part of an ubiquitous campus game called *Flikken* where users use their mobile devices to chase dangerous (virtual) gangsters around the city. Players are organised in two teams which determines their role in the game. The policemen work together to shoot down dangerous gangsters on the loose before they achieve their goal of earning 1 million euro by committing crimes. Players have a representation of all nearby team members on their devices which allows them to keep track of their location (e.g. using GPS coordinates) and to orient themselves in the city in order to coordinate their movements. For example, proximate team members can send messages to each other in order to decide on a group strategy to defeat the other team. Additionally, players can collect (virtual) objects around the city (e.g. knives, explosives, guns, etc.), which can be used to damage members of the opposite team and to commit crimes. In this session, we will implement the following functionalities of Flikken:

1. Players can track the position of nearby team members on a map.
2. When a player does not update its position for a certain amount of time, its visual representation on the map should be grayed out.
3. Players die when they step into a mine virtual object.

7.2 Implementing Flikken

We will implement these functionalities starting from the skeleton code shown below:

```
def makePlayer(username, team, userInterface := nil) {
   //stores the player current location
  def location := [0,0];
   // the interface for coordination with remote players
  def tupleSpace := .. ;
  // local interface with Java gUI.
  def localInterface :=  object: {
    def initialize() {  //TODO  };
    def updatePlayerLocation(newLocation){ //TODO };
  };
   // TODO: setup the binding with Java gui
   localInterface;
}
```

`makePlayer()` takes as parameter the team and username of the player, and the user interface of the application. The user interface is implemented by a Java AWT GUI provided with the lab material which simulates the player movement around the VUB campus.[1] The GUI displays the position of a given player at the VUB campus by means of a dot. Each time the dot in the GUI gets moved, the `updatePlayerLocation` method is called with the new position expressed as [x,y].

(a) Complete the implementation of the skeleton code so that players display on their game GUI the nearby team members. Assume for now that tuples use the default tuple propagation protocol.

Note: An `inEuclideanDistance` helper function is provided with the skeleton code to calculate when two locations expressed as [x, y] are within a certain range.

(b) Adapt your implementation (if necessary) to pass the `testAsyncTracking-TeamPlayers` unit test that checks the functionality implemented in (a).

(c) Adapt your implementation to add the functionality number 2: when a player does not update its position for a certain amount of time, its visual representation should be grayed out.

(d) Add a unit test that checks the functionality added in (c). Make use of the **disconnect:** construct to simulate disconnections in your unit test.

Note: Some helper functions are provided with the skeleton code to easily set up remote players and get the nearby members of a player's team.

(e) Adapt your implementation to introduce a *Mine* virtual object which kills *all* players stepping on it, i.e. his position is within the damaging range of the mine. You can assume that throughout the campus some supporting devices are placed to inform players about virtual objects by injecting the necessary tuples.

(f) Adapt the functionality of the Mine virtual object so that it only kills the *first* player stepping on it.

(g) The default propagation protocol disseminates tuples to all connected players. This can lead to network flooding, performance repercussions on mobile devices and endanger privacy (since all devices can potentially access all information). Adapt the propagation protocol of the tuples used in this application as follows:

(g.1) The information shared amongst team members (e.g their positions or messages with the agreed strategy) should not be accessed by the players of the other team who could use it to their own advantage. Such information should be sent only to team members.

[1] The Java GUI files must be placed at `at/labsessions` directory. Otherwise, update the Java package name accordingly.

(g.2) Adapt your implementation so that tuples targeting team members are only spread to one-hop neighbours. In other words, tuples should be transmitted only to connected devices at the time of the insertion (providing a similar behaviour as default semantics of federated tuple space-based models).