Game Oriented Multi Agent System, based on Jade and Jason

Derived and adapted from material of

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n. 0622 Valencia (Spain)

Antonio Barella Álvarez
email: tbarella(at)dsic(dot)upv(dot)es

Professor Vicente J. Botti Navarro
email: vbotti(at)dsic(dot)upv(dot)es

Dr. Carlos Carrascosa Casamayor
email: carrasco(at)dsic(dot)upv(dot)es

Presentation

- General overview
- JGOMAS Description

- Agent platform for simulations and videogames on a 3D environment
  - Cooperative and competitive environment
  - Goal: to improve the individual and colective behaviour
- Game: ***C**apture **T**he **F**lag - **CTF***
- Basically …
  - Two teams: ALLIED vs. AXIS …
  - … with different goals …
  - … within a virtual environment

- There is a finite number of agents
- There is also a time limit
- Each agent belongs to one side:
  - Allied
  - Axis
- Allied agents must go to the base of the Axis, capture the flag, and take it to its base
- Axis agents must defend the flag and, if captured, return it to its base

- Architecture
- Agent Taxonomy
- Execution Loop
- Tasks
- Interface (Jason API)
- Communication

Architecture: agent platform

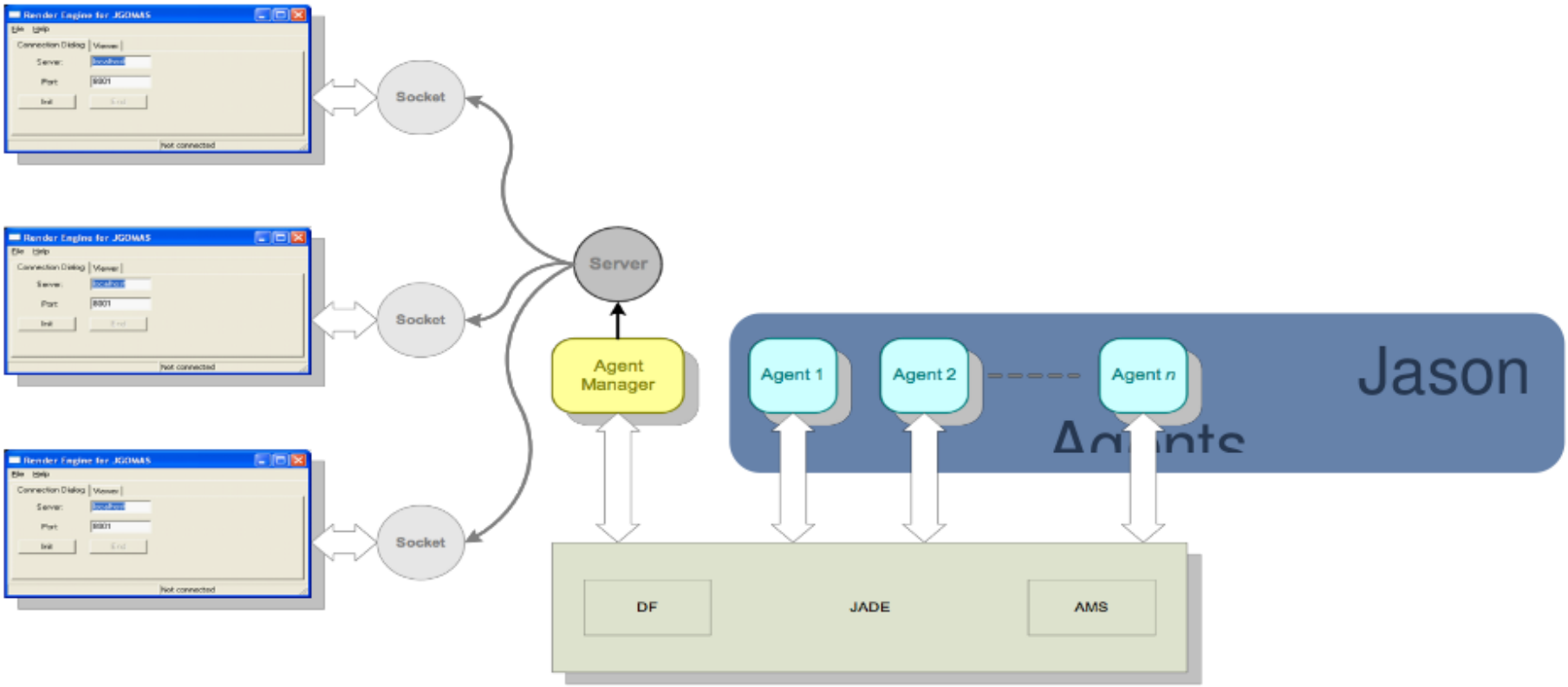
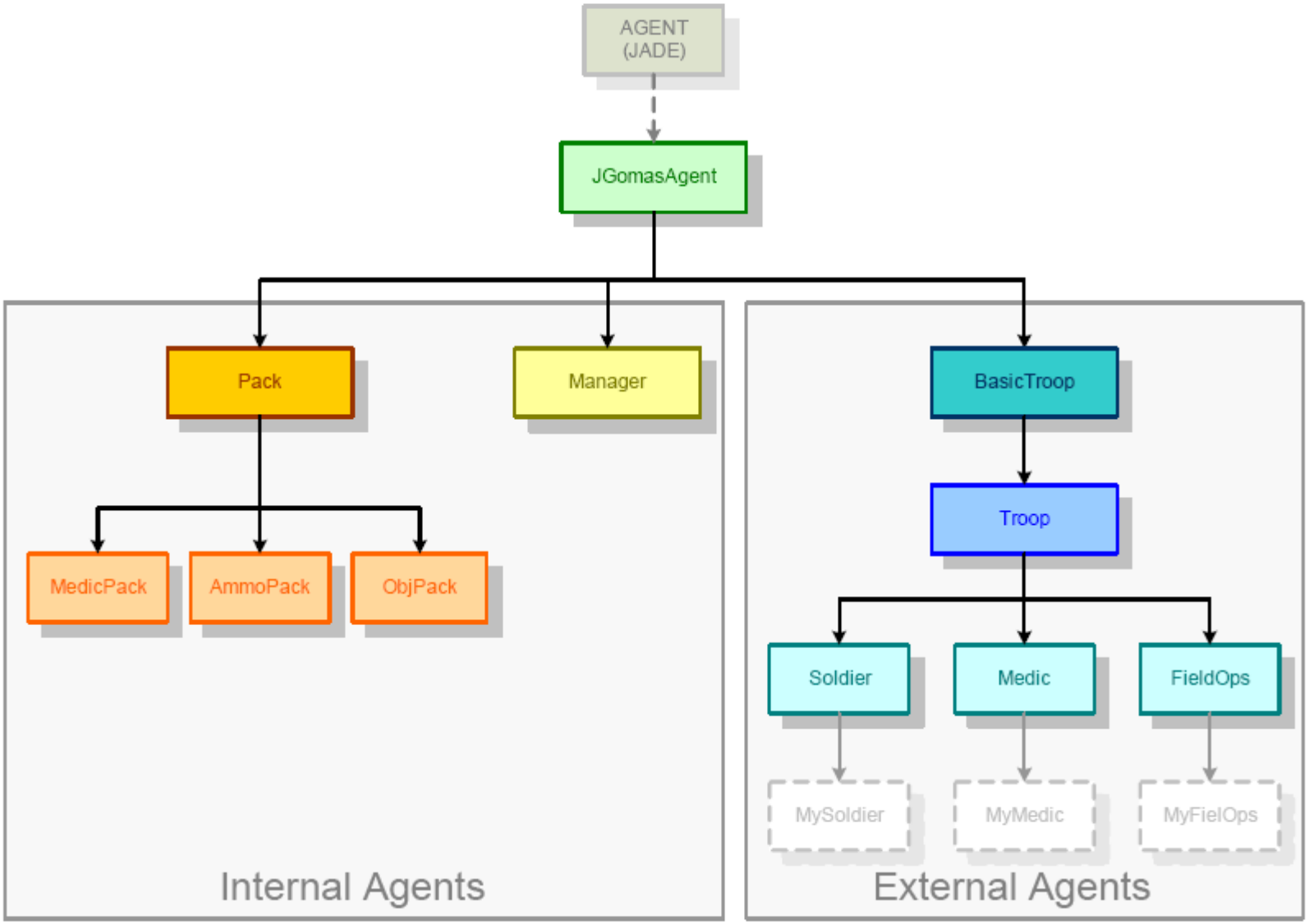


Figure 1: Jason-JGOMAS Architecture

Architecture: render engine

Agents taxonomy



Manager

- Control the state of the game:
  - Keeps the information of all objects and sends it to the viewer
- Management of game logic
  - Life cycle management
  - Coordination and management of agent services
  - Current state of the game
  - Deal with the requests of the agents regarding the environment (shoot, look, ...)
  - Statistics

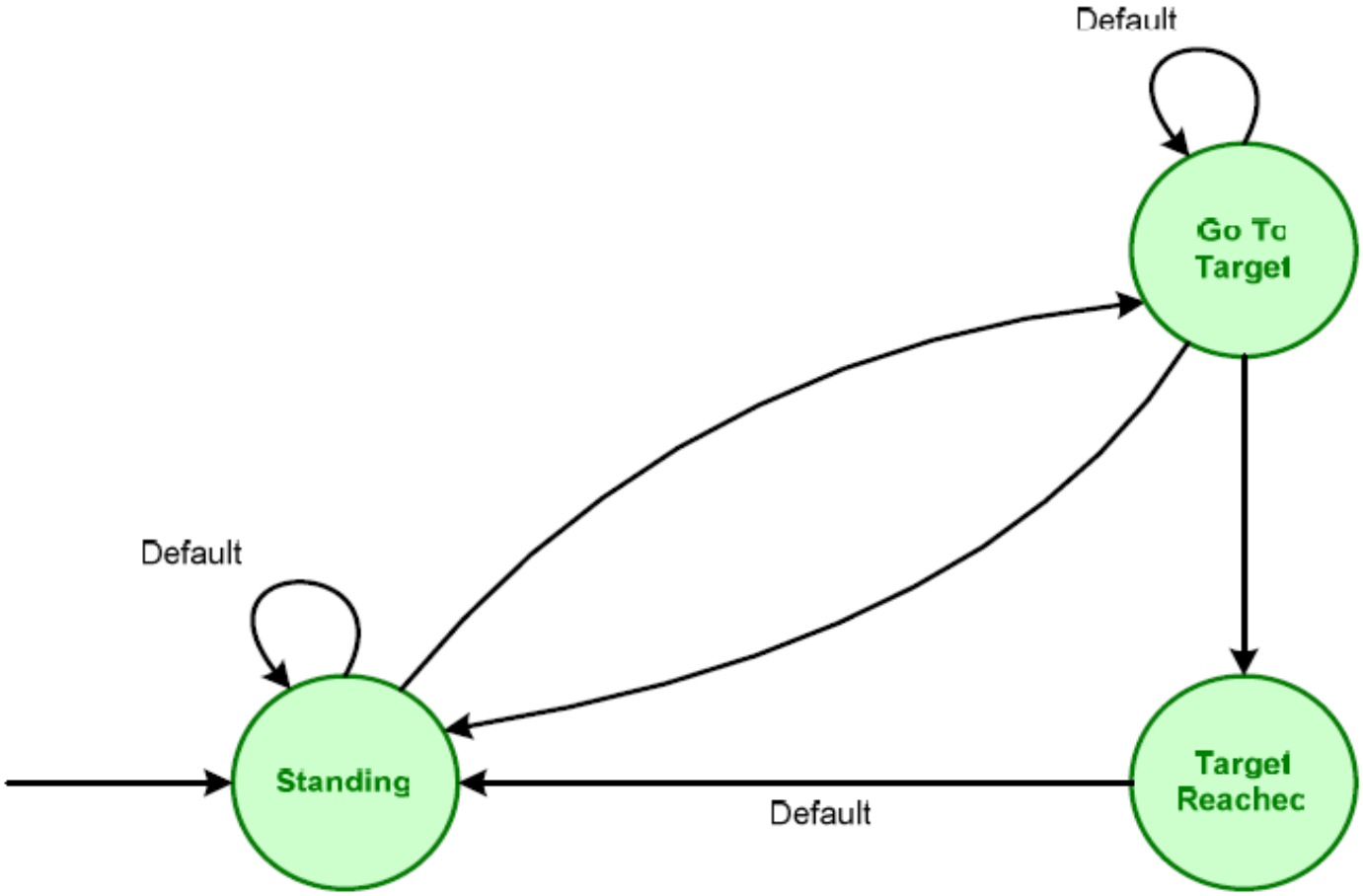
Troops

- There are **three types of roles** defined:
  - Soldier: provides support - CallForBackup CFB
  - Medic: provides treatment - CallForMedic CFM
  - FieldOps: provides ammunition - CallForAmmo CFA
- An agent assumes a single role during the whole game
- Each role has some characteristics and offers certain services. It has a **basic** behavior in JASON that should be improved.

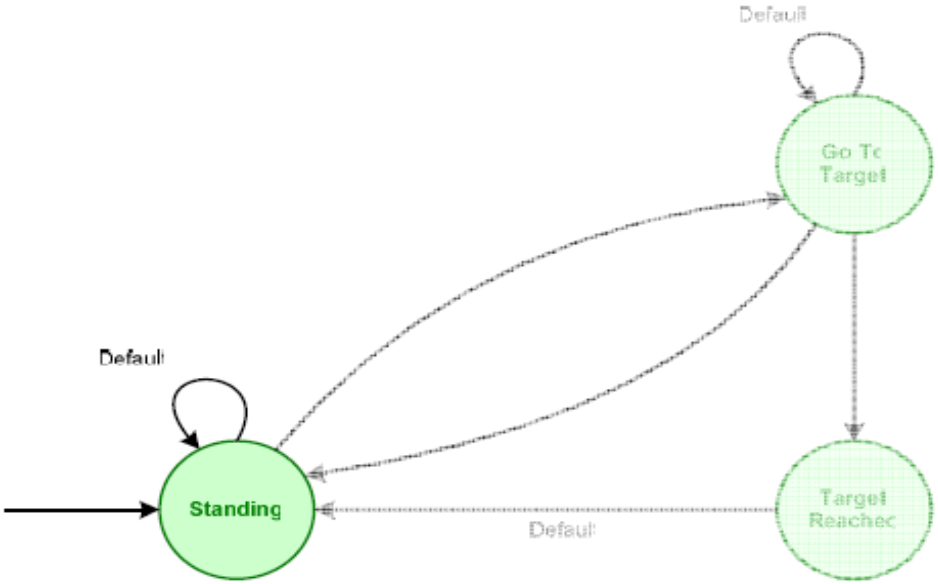
Execution loop

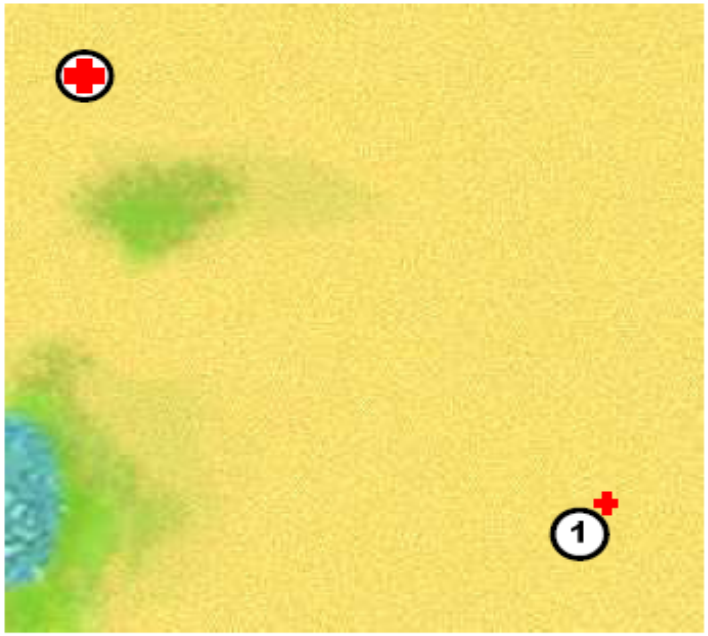
- Each agent executes an FSM:
  - STATE_STANDING
  - STATE_GOTO_TARGET
  - STATE_TARGET_REACHED
- FSM is used to perform tasks:
  - Start (Launch)
  - Development (Execution)
  - Final (Action and Destruction)
- The highest priority task is always selected

Execution loop

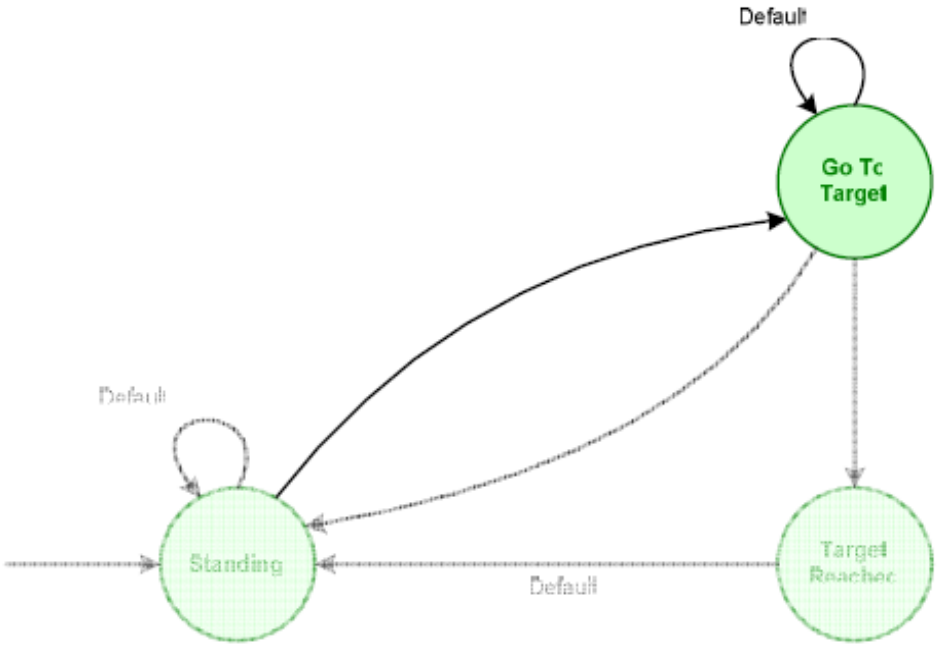


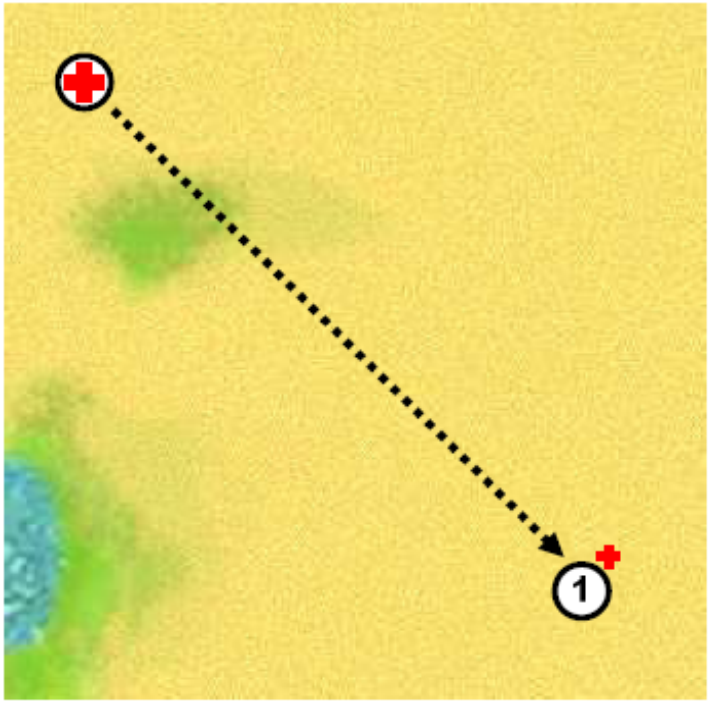
Execution loop



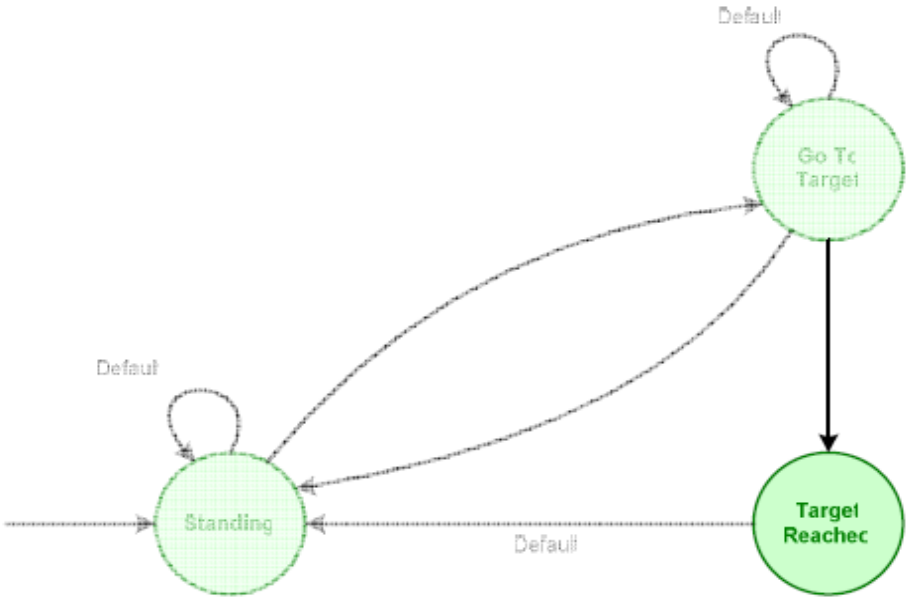


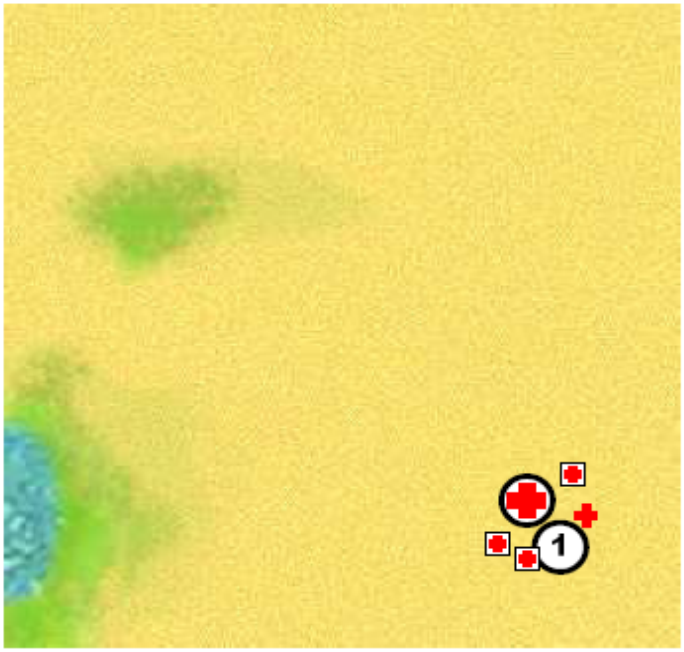
Execution loop



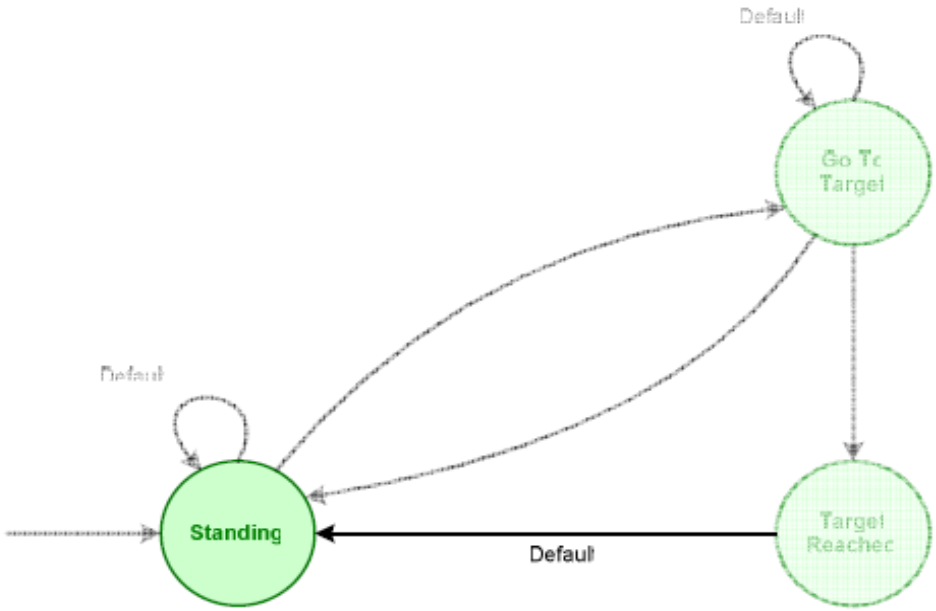


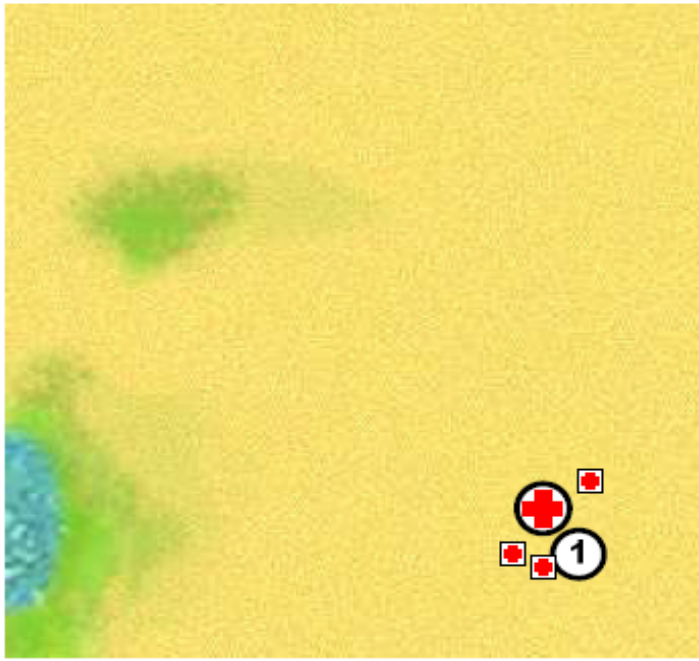
Execution loop





Execution loop





Task examples

!add_task(task(TaskPriority, TaskType, Agent, Position, Content))

- Priority: Priority of the task
  - The highest priority task is always selected
  - It is possible to redefine the priority of each type of task
  - The tasks are executed by the system, not the user!
- TaskType: Task type
- Agent: agent associated with the task
- Position: Position where to carry out the task
- Content: Possible additional content

!add_task(task(**1000**, "TASK_GET_OBJECTIVE", M, pos(X, Y, Z), ""));

Main tasks

- TASK_GIVE_MEDICPAKS: A medic must generate medipacks in a particular place (the current position of the agent who asked for medicines).
- TASK_GIVE_AMMOPAKS: A fieldops must generate packets of ammo in a particular place (the position of the agent who asked for ammo).
- TASK_GIVE_BACKUP: A soldier should go to help a teammate to a particular place (the position of the agent who asked for help).
- TASK_GET_OBJECTIVE: An ALLIED agent must go to the starting position of the flag. If he manage to grab the flag, the agent goes back to his home base.
- TASK_GOTO_POSITION: The agent must go to a specific location.
- ...

Interface (Jason API)

- jgomas.asl : non-modifiable behavior of the agents

- jasonAgent_ALLIED.asl
- jasonAgent_ALLIED_MEDIC.asl
- jasonAgent_ALLIED_FIELDOPS.asl
- jasonAgent_AXIS.asl
- jasonAgent_AXIS_MEDIC.asl
- jasonAgent_AXIS_FIELDOPS.asl

Interface (Jason API): agent beliefs

- **tasks(task_list)** : Contains the list of active tasks of the agent.
- tasks([task(1000,"TASK_GET_OBJECTIVE","Manager",pos(224,0,224),""),task(1001,"TASK_GIVE_MEDIPAKS","A2",pos(204,0,228),"")])
- **fovObjects(object_list)** : Contains the list of objects currently seen by the agent. The structure of an object is:

  [#, TEAM, TYPE, ANGLE, DISTANCE, HEALTH, POSITION].
- Example: [1,200,1,0.58,14.76,78,pos(214,0,219)], the object 1 of the team 200 (AXIS), type 1 (agent), with angle 0,58, with a distance of 14,76, a health 78 and its position is pos(214,0,219)
- Note: The TEAM values are: 100 (Allied), 200 (Axis), 1003 (flag)

Interface (Jason API): agent beliefs

- **state(current state)** : indicates the state of the agent in his state machine: standing (selecting which task to do or waiting), go_to_target (going to its next target), target_reached (it has reached the target), quit (has to finish).
- **my_health(X)** : Stores the health of the agent. The initial (and maximum) value is 100. When this value reaches 0, the agent dies.
- **my_ammo(X)** : Stores the amount of bullets of the agent. The initial value is 100.
- **my_position(X,Y,Z)** : Stores the last position known by the agent.

Interface (Jason API): plans

- **!perform_look_action** : this objective is invoked when the agent looks around and update the list of surrounding objects fovObjects(L). It would be necessary to implement the plan associated to the creation of this event to be able look what is around.
- **!perform_aim_action :** This objective is triggered if there is an enemy to aim, which can be used to take a decision about what to do with the aimed agent. An easy implementation of the plan is available. It is interesting to improve this plan to take a more refined decision about who to aim.
- **!get_agent_to_aim** : This objective is invoked after !perform_look_action, and it would be used to decide if there is any enemy to aim. An easy implementation of the associated plan is available. It is interesting to improve the mentioned associated plan to take a more refined solution about who to aim.

Interface (Jason API): plans

- **!perform_no_ammo_action :** This objective is triggered when the agent shoots and has no ammo. It is necessary to implement the associated plan to take a decision. For example, to run away.
- **!perform_injury_action :** This objective is triggered when the agent is shot. It is necessary to implement the plan associated to the creation of this event to take a decision. For example, run if the agent has a low life value.
- **!performThresholdAction :** This objective is triggered when the agent has less life or bullets than the thresholds my_ammo_threshold(X) and my_health_threshold(X). An easy implementation of the associated plan is available, which asks for help to medics or fieldops of his team. It is interesting to improve the plan to take a more refined solution.

Interface (Jason API): plans

- **!setup_priorities :** This objective is triggered during agent initialization to fix agent task priorities. Each agent has its own priorities. A simple implementation is available. It is interesting to modify it to add new tasks or modify the priorities to have agents that behave in a different way.
- **!update_targets :** This objective can be used to update the tasks and its priorities. It is invoked when the agent changes to the *standing state* and has to choose a new task among the available ones. It is necessary to implement the plan associated to the creation of this event.

Communication

- Should help agent **coordination**
- **No communication (implicit)**: coordination is achieved by sensing the environment. When an agent looks around itself the objective !perform_look_action is triggered. By rewriting the associated plan it can be decided what to do according to the perception.
- **With communication (explicit)**: In this case it is used the message passing using the internal action

  .send_msg_with_conversation_id

Communication

- Example: A1 wants to send a message to its team telling them to go to (to help, coordinate, regroup...)

```
...
?my_position(X,Y,Z);
.my_team("AXIS", E1);
.concat("goto(",X, ", ", Y, ", ", Z, ")", Content1);
.send_msg_with_conversation_id(E1, tell, Content1, "INT");
```

Communication

- Following the same example, the rest of agents of the team could have a plan with this shape:

```
+goto(X,Y,Z)[source(A)]
    <-
    .println("Received a goto message from ", A);
    !add_task(task("TASK_GOTO_POSITION", A, pos(X, Y, Z), ""));
    -+state(standing);
    -goto(_,_,_).
```

Lab 0 with JGOMAS (1)

- Visit JGOMAS site:
  - http://gti-ia.dsic.upv.es/sma/tools/jgomas/index.php
- Download and install
  - jgomas-2.0.1.zip (jason version)
  - Unity 3D Render Engine (also older render)
    - Linux, mac, windows
- Play a match with the default configuration
- Change default parameters (*.bat or *.sh)
- Revise code *.asl
- Configure Eclipse …
- ...

Lab 0 with JGOMAS (2)

- Perform different executions with different numbers of agents, classes and maps!
  - 3 allies vs. 3 axis: 1 soldier, 1 medic and 1 fieldop
  - 6 allies vs. 3 axis (allies should win)
  - 3 allies vs. 6 axis (axis should win)
  - Test the maps map_01, map02, map_03, map_04 ...
  - Modify the duration of the game
  - The belief debug(X) allows to change the verbosity of the agent (its value is between 1 and 3), try changing the verbosity
  - ...

Lab 0 with JGOMAS (3)

- Revise Jason JGOMAS Manual.pdf
- Revise *.asl code
- Start Homework3
- ...