Task 1 - Implement an ALLIED agent that shows his position and distance to the flag and his distance to the base.

For this task we create a new file called `jasonAgent_ALLIED_T1.asl`, is a variation of the original `jasonAgent_ALLIED.asl` file. To accomplish the task, we edited the `get_agent_to_aim` plan in the new file, the plan is executed every time the agent looks for objects in his Field Of View (*FOV*). Using the beliefs `?mi_position`, `?objective`, `?distance`, `?base_position`, etc. We calculate the distance to the flag and to the base, and we print agent's position.

jasonAgent_ALLIED_T1.asl:

```
 ?my_position(X, Y, Z);
.println("[TASK 1] My position: ", math.round(X),", ", math.round(Z));

?objective(FlagX, FlagY, FlagZ);
!distance(pos(FlagX, FlagY, FlagZ)); // Calculate de distance from agent's position to the flag
?distance(D);
.println("[TASK 1] Distance to the flag: ", math.round(D), " units.");

?base_position(BaseX, BaseY, BaseZ);
!distance(pos(BaseX, BaseY, BaseZ)); // Calculate de distance from agent's position to the base
?distance(Db);
.println("[TASK 1] Distance from the base: ", math.round(Db) , " units." );
```

Task 2 - Implement a "crazy" AXIS agent that moves randomly.

First of all, we create a new file `jasonAgent_AXIS_T2.asl` for the new agent, and we initilized the `is_crazy(1)` and `rand_mov(1)` beliefs in the "init" plan.

As in the first task, we edited the `get_agent_to_aim` plan, but in this case we edited it in order to make the soldier move randomly. The random move is performed every 10 *ticks*, using the `rand_mov(N)` belief, using `N` as a counter. When `N` reaches 10, a random number is generated, using this number the soldier moves *up*, *down*, *left* or *right*, this is performed using the apropiate belief in each case: `order(up)`, `order(down)`, `order(left)` or `order(right)`.

Until we reach the 10 ticks to generate the random number, the tick counter is increased, removing `rand_mov(N)` and adding a new `rand_mov(N+1)`. We also removed all the `order(...)` beliefs.

jasonAgent_AXIS_T2.asl:

```
 +!init
    <-
    ?debug(Mode); if (Mode<=1) { .println("YOUR CODE FOR init GOES HERE.")};
    -+is_crazy(1);
    -+rand_mov(1).

...

+!get_agent_to_aim
    <-
    ...
    ?current_task(T);
    ?is_crazy(C);
    ?rand_mov(N);

    if(is_crazy(1) & (N mod 10) == 0){ // Tick reaches 10 and is crazy
        -rand_mov(N);
        +rand_mov(1);
        .random(X); // Generate the random number
        .println("[TASK 2] Moving randomly...");
        if(X < 1/4){
            -+order(up);
        }else{
            if(X < 2/4){
                -+order(right);
            }else{
                if(X < 3/4){
                    -+order(down);
                }else {
                    -+order(left);
                }
            }
        }
    }else{ // If the 10 ticks haven't passed yet... increase the counter
        -rand_mov(N);
        +rand_mov(N+1);
        ...
```

Task 3 - Implement an AXIS agent that locates his "crazy" partner and follows him.

The behaviour of the new soldier created at `jasonAgent_AXIS_T3.asl` is quite similar to a regular soldier's one, but, in this case, he follows his crazy partner who moves randomly.

To complete this task, we edited the "crazy" soldier, in the `jasonAgent_AXIS_T2.asl` file. We added a new plan `wanna_follow_crazy_one[source(A)]`, which is activated once another soldier sends him a message with the `wanna_follow_crazy_one` plan. The crazy agent creates a new belief `wanna_follow_me(A)`, where `A` is the agent who wants to follow the crazy one.

Also, we edited the `get_agent_to_aim`, so everytime the tick counter is increased, the crazy agent sends to the follower an order to move to his actual position.

jasonAgent_AXIS_T2.asl:

```
...

+wanna_follow_crazy_one [source(A)]
<-
    .println("[TASK 3] I am the crazy one, follow me!");
    -+wanna_follow_me(A);
    -wanna_follow_crazy_one.

...

+!get_agent_to_aim
<-
    ...

    if(wanna_follow_me(A)){
        ?wanna_follow_me(A);
        ?my_position(X,Y,Z);
        .concat("order(move,",X,",",Z,")",Content);
        .send_msg_with_conversation_id(A,tell,Content,"INT");
        .println("[Task 3] Come and protect me!");
        -+wanna_follow_me(A);
    }
    ...
```

For the new follower soldier, we started setting some instructions in the "init" plan. This instructions consist in sending a `wanna_follow_crazy_one` message to every axis soldier. Then the `wanna_follow_crazy_one` plan of the crazy soldier will be trigger, so the follower soldier will be register and every tick the crazy soldier will send him a message with an order to move to his actual position.

Note that whenever the crazy soldier gets stuck in a wall, this follower soldier will stop following him and will continue as a regular soldier.

jasonAgent_AXIS_T3.asl

```
  +!init
    <-
    ?debug(Mode); if (Mode<=1) { .println("YOUR CODE FOR init GOES HERE.")};
    .my_team("AXIS", E1);
    .println("[TASK - 3] Who is the crazy one??");
    .concat("wanna_follow_crazy_one", Content);
    .send_msg_with_conversation_id(E1,tell,Content,"INT").
```

Task 4 - Implement an ALLIED agent that locates the "crazy" agent and kills him. The "crazy" agent can defend himself.

From the part of the crazy agent, this task is similar to the previous one, the agent also need to follow the him, but in this case a new `wanna_kill_crazy_one[source(A)]` plan is created. This new plan behaves as the previous `wanna_follow_crazy_one` plan.

The new plan is activated once another soldier sends him a message with the `wanna_kill_crazy_one` plan, the crazy agent creates the new belief `wanna_kill_me(A)`, where `A` is the agent that want to kill the crazy soldier.

As in the previous task, in the `get_agent_to_aim` plan, the crazy agent sends to the follower (the killer) an order to follow him.

jasonAgent_AXIS_T2.asl:

```
...

+wanna_kill_crazy_one [source(A)]
    <-
    .println("[TASK 4] I am the crazy one, try to kill me!");
    -+wanna_kill_me(A);
    -wanna_kill_crazy_one.

...

+!get_agent_to_aim
    <-
    ...
    if(wanna_kill_me(B)){
        ?wanna_kill_me(B);
        ?my_position(X,Y,Z);
        .concat("order(move,",X,",",Z,")",Content1);
        .send_msg_with_conversation_id(B,tell,Content1,"INT");
        .println("[Task 4] Come, and kill me!");
        -+wanna_kill_me(B);
    }
    ...
```

For the new killer agent, as the previous follower soldier, we started setting some instructions in the "init" plan, sending a message to every axis soldier. Then the `wanna_kill_crazy_one` plan of the crazy soldier will be trigger, registering the soldier, and sending an order with the actual position to him every tick.

jasonAgent_ALLIED_T4.asl:

```
+!init
    <-
    ?debug(Mode); if (Mode<=1) { .println("YOUR CODE FOR init GOES HERE.")};
    .my_team("AXIS", E1);
    +speak(1);
    .concat("wanna_kill_crazy_one", Content);
    .send_msg_with_conversation_id(E1,tell,Content,"INT").
```

Important note: there is a bug that we could not solve, that is when this agent crosses with the crazy one, our killer agent stops its execution.

Task 5 - Include a new task at your choice.

For this task we created a new **BOSS RUNNER** soldier, that his main objective is to take the flag and return to his base. This soldier is limited in many aspects, for example it has no ammo, so it can't shoot. But in other aspects is superpowered, for example, it has 1500 HP (health points), so is tricky to kill it.

As the agent can't defend himself, when this soldier is under fire, his whole team goes to his position to help it. The `perform_injury_action` is used for this, when the soldier gets hit, the plan will check if the counter `H` that is defined in the `help_ticks(H)` belief has reached the threshold (5). So, if the threshold is reached, the soldier will create a `order(help)` belief and will restart the help tick counter with `help_ticks(0)`.

When the `order(help)` is introduced, the team will help the soldier, nota that this is only performed when the `H` counter reached the threshold value (5).

jasonAgent_ALLIED_T5.asl:

```
 ...

+!perform_injury_action
    <-
    if(help_ticks(5)){
        +order(help);
        -help_ticks(5);
        +help_ticks(0);
    }
    ?help_ticks(H);
    -+help_ticks(H+1).

...

+!init
    <-
    ?debug(Mode); if (Mode<=1) { .println("YOUR CODE FOR init GOES HERE.")};
    +my_health(1000);
    +my_ammo(0);
    +help_ticks(5).
```