MARS AI Game Simulation: Laser Tag

Conceptual Description of the Game, Rules, and Agents

1ST DRAFT

Daniel Osterholz, Nima Ahmady-Moghaddam

Hamburg University of Applied Sciences (HAW)

Concept:

This virtual multi-agent game simulation is modeled after the rules and dynamics typically associated with a real-world laser tag game. The game world is inhabited by teams of four agents. Each agent behaves as an autonomous entity and calibrates its behavior based on its current attributes and environment. The game progresses on a timeline divided into ticks. During each tick, each agent completes the actions programmed for it one time. Agent attributes and states (listed below) are adjusted according to the outcomes of those actions.

The agent teams will compete against each other in a simulation run of the game to determine a winning artificial intelligence (AI) by means of a points system.

Game world (map):

- Shape: a two-dimensional square grid with grid cells of equal size
- Divided by walls and into rooms (possibly of square, rectangular, or other straight-lined shapes) that can be entered and exited via specific entrance points (gaps in the wall)
- Agent teams spawn at distinct corners of the map

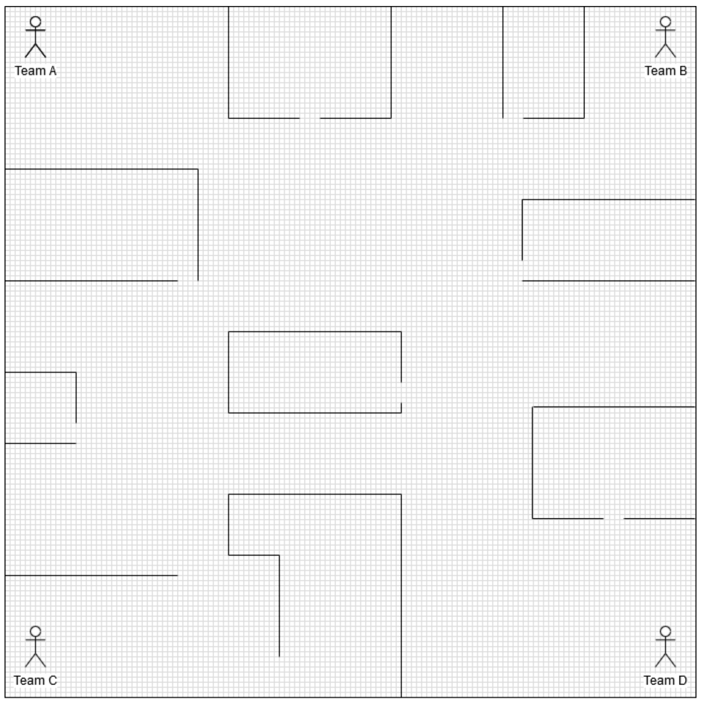

Figure 1: Possible layout of the game world (not necessarily to scale). The grid indicates the grid layer's cells on which in which agents can be located

Agents:

- Attributes:
    - `Stance`: standing, kneeling, lying
    - `Energy`: ranging from 0 to some maximum energy level
    - `Vital_status`: alive or temporarily disabled
    - `Tag_status`: states whether agent was "tagged" (i.e., hit) by an enemy agent in previous tick
    - `Enemy_list`: a list of all enemies spotted in the current tick (and possibly some number of past ticks) by all agents on the team.
- Operations:
    - `Explore()`: explores the agent's immediate environment (a radius of some radius r) and returns a list of all objects of interest within r that are visible to agent
    - `ChangeStance()`: changes agent's `stance` value from current stance to one of the other two available stances (specified by agent)
    - `Move()`: moves agent from its current position to some chose position that can be reached within one tick
        - There might be different ways of moving based on the type of movement desired by the agent (move towards enemy, move along wall, retreat, take cover, etc.).
    - `Shoot()`: agent shoots with his/her laser gun in an attempt to tag an enemy agent
        - Aimed shot: fires once at a specific agent who was spotted within reach on a specific grid cell
        - "I-feel-lucky": fires once at one (or more than one) grid cell(s), hoping that an agent who was not spotted is on that grid cell and gets tagged

Game dynamics:

The above definitions of agent attributes and operations result in certain game dynamics that need to be considered when programming an AI's behavior for the game simulation:

- Stance system:
    - An agent's stance affects the following parameters:
        - Visibility: a standing agent can be spotted by an enemy agent from the higher distance, than a kneeling agent, then a lying agent
        - Visual range: a standing agent can look farther (i.e., has a higher r), than a kneeling agent, than a lying agent
        - Change of getting tagged: a standing agent is more likely to get tagged, than a kneeling agent, than a lying agent
        - Motion speed: a standing agent can traverse a greater distance within one tick, than a kneeling agent, than a lying agent
    - An agent's stance can be changed only once per tick
- Team communication:
    - An agent's `explore()` operation returns all objects of interest that are within the agent's range and that are visible to the agent. Each team member's enemy sightings are communicated to all other team members. For example, if Team A consists of Agent x and Agent y, and if x spots an enemy Agent z, then x stores z's location and attributes in his `enemy_list` AND forwards z's location and attributes to y to store in his/her `enemy_list`.

- Vital status:
    - Getting tagged reduces the value of the `energy` attribute. If an agent's `energy` attribute drops to or below 0, the agent's `Vital_status` value changes from `alive` to `disabled` and the agent must skip a few ticks. After the number of ticks to be skipped has passed, the agent will respawn at his/her team's spawn location at the beginning of the simulation.

AI Rules:

- An agent may complete each available operation once per tick

Points system:

- An agent who tags an enemy agent produces points for his/her team
- For statistical purposes and to compare individual agents' performances, an agent-level point count might be incorporated as well
- The team with the highest score at the end of the simulation time wins the game

<u>Other ideas:</u> the following features have not been discussed yet. If deemed sensible, might be included in the game:

- A operation `CallForHelp()` that allows an agent to call one of more of his/her team members to his/her location for assistance (this might require an attribute `team_location` agents of the same team to know each other's locations and possibly other features)
- To include the rooms more directly in the game dynamic, hidden items might be added to the game world that, when found, add to the team's point count (this might require an attribute `has_been_visited` to allow an agent/team to track which rooms have already been explored as well as an attribute `item_tracker`)