Guidelines Document for Project 2.2

GitHub Organization: https://github.com/Project-2-2

Code repository: https://github.com/Project-2-2/GameInterop

Java Version: 10 (Download)

This guidelines were last modified on:

Monday, 24 February, 2020

The current version can be found on Google Docs.

Amendments (February 21, 16.00; from email):

- We just decided in another negotiation meeting that every group has to implement their game controller. We will provide a basic interoperability API for the agents, which makes sure that all groups adopt a certain standard for the project and the competition. Moreover, we will provide some basic guidelines for implementing the game controller, but without the specific implementation. That means that every group has to implement, for example, their own collision- and noise detection algorithm.
- We do not have time steps, we only have a turn-based system. For example, the maximum speed is defined as the maximum distance covered for one turn.
- We have rejected the idea that every action is executed at the same time. Teams move after each other, one agent at a time. If an agent moves on, the world state gets updated and the agents perceive the updated state.
- Since a team cannot smell the pheromones of the other team, there is no possibility that a team can annoy the other team by spamming the field with pheromones.
- Pheromones can move through walls, so we ignore the walls. Pheromones can be smelled behind a wall as well. The same applies to sounds.

Project management and code integration:

We have agreed on utilizing Git and GitHub functionalities. In order to facilitate interoperability each group is provided with code from:
https://github.com/Project-2-2/GameInterop

In short:

- Each group forks GameInterop repo on GitHub
- Groups are allowed to change code only in their dedicated directories
- Code is integrated through pull requests on GitHub

In detail, the foreseen workflow is as follows:

1. Each group must forks the GameInterop repository on GitHub (You can do that from a private account or by creating an organization for your group.)
2. Your group works on that fork (or a separate private clone of that repository)
    a. Each group can make changes only in the following two directories:
        i. src/main/java/GroupX (here goes the main code of group X)
        ii. src/test/java/GroupX (here go the tests of group X)
    b. Any changes outside of the two above directories have to be agreed by all groups!
3. In order to take part in the competition you need to implement the AgentsFactory that is located in your group directory. You will also need the implementation of:
    a. Interop.Agent.Intruder interface
    b. Interop.Agent.Guard interface
4. At some point that will be agreed later (probably a week before the presentations of phase 1) all groups will be requested to submit pull requests with their GameController implementation. The best game controller will be selected by group representatives. Each group will be given time to test their agents with the selected game controller.
5. Some time before the competition, during one day window, all groups will be requested to make pull requests with your agents code. Agents code will have to be made public to everyone at that point (before that you are free to keep it secret). No agent changes will be allowed after that agent submission window.
6. The best controller will be used for the competition.

Any changes made outside your group directories, and not previously agreed with other groups, will result in automatic pull request rejection! If any group will make anything that will make code integration difficult, then that group will be responsible for fixing it in agreement with other groups.

Each group needs to implement their own game controller.

Rationale: This decision was made because developing a common game controller will very likely result in undue burden on a few people doing a lot of work for ~100 other people. During the February 21 meeting when the decision was made there were only 4 representatives out of 11.

The best game controller should have:

1. high automated test code coverage - we want to avoid bugs
2. the game execution decoupled from the UI (the game should work without UI)
3. good documentation
4. good speed of execution

How to keep your agents code secret?

- First of all, you are allowed to keep the code secret.
- You need to fork the GameInterop repo on GitHub only to be able to submit pull requests. First with game controllers and later on with agents.
- Before making a pull request you can host your code anywhere you like e.g. private GitLab repo, or anywhere else.
- You can also just copy paste your code into the forked repo just before making a pull request. Make sure beforehand that your code works as expected when integrated with GameInterop!
- Be mindful that at some point you may need to separate your game controller from your agents code in order to keep your agents code secret!
- There may be bug fixes necessary in the GameInterop. You will need to be able to sync them with your code.

General Rules:

- Continuous space
- Discrete time (effectively turn-based with a lot of turns)
    - Think about numerical integration algorithms with discrete time step, not chess
    - Why no threads (CPU parallelism) -> simplicity and the threads need to be synchronized anyway
- Groups are free to implement agents as they wish as long as they are implementing the appropriate Java Guard interface and Intruder interface
- Explicitly:
    - Scripting (hard coding) actions is allowed
    - Learning is allowed. This includes:
        - Pretraining agents
        - Preserving state between actions
- We do not try to simulate a real world situation:
    - Specifically, we make compromises in order to reduce complexity of this specification and following implementations
- Agent is a circle with a radius of 0.5, so the diameter is 1.0
- We are using: https://github.com/Project-2-2/GameInterop
- Intruders are spawned, knowing the direction of the target and no other knowledge of the map
- Guards are spawned with no knowledge of the map

Victory conditions:

- Intruder is captured by guard if all of the following apply:
    - If guard is within capture distance of an intruder as defined in the scenario
    - If intruder is in the field of view of the guard
- Two game modes:
    - Game mode 0 (capture all intruders)
        - Intruders win if any intruder reaches the target area and stays in the target area continuously for number of turns defined in the scenario
        - Guards win if all intruders are captured
    - Game mode 1 (capture one intruder)
        - Intruders win if any intruder reaches the target area and stays in the target area continuously for number of turns defined in the scenario
        - Guards win if any intruder is captured

Levels:

- Levels can be implemented by fully separating parts of 2D map with walls. But the map is nonetheless a simple, continuous 2D space.
- Intruders can sense the direction of the target area on each turn even though getting to that area may be impossible without teleporting. Agents do not sense the direction of teleports.

Game control:

- Agents make moves in turns
    - See “Turn system”, “General action semantics” and “Possible actions”
- Actions are executed one after another, as soon as the action is issued by an agent
- Each agent perceives world state with the action of the previous agent

Turn system:

- World state is updated after each action issued by an agent
- Intruders move first, Guards second
    - whole loop is one turn
    - a turn is the only unit of time
- Each group can define their own order of agents inside their team
- The order inside a team is defined through AgentsFactory
    - AgentsFactory returns a list of agents (an ordered collection).
    - The order of that list will be used for deciding agents' action order.
- No agent is allowed to move twice during one turn.
    - Specifically, you are not allowed to return twice a reference to the same agent object instance from your group AgentsFactory.

- Rationale: we have decided on turn system where agents move after one another, because this system can be more precisely defined, and therefore it is less likely to lead to inconsistencies between game controllers

General action semantics:

- Agents issue only one action at a time
- Game controller is responsible for executing action (updating world state)
- If action is invalid, then controller ignores that action
    - Agent gets informed by boolean flag in Percepts that previous action was ignored
- Some actions can have a cooldown period lasting amount of time specified in the scenario file
    - All actions issued during cooldown period, besides the NoAction action, are invalid.

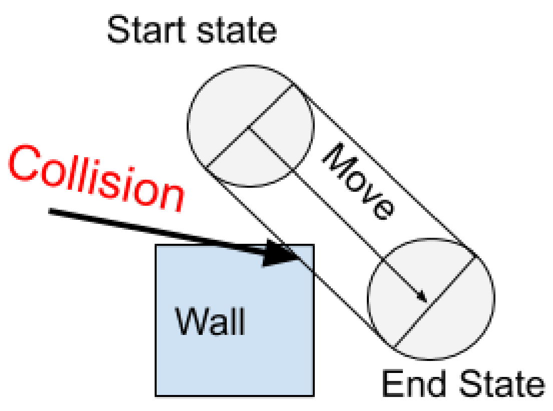
Possible actions:

- Move
    - Move happens on the straight line in the direction that an agent is facing
    - Agent can specify the distance
    - Senario specifies the max distance that an agent can move
        - The max distance can be modified based on the slowDown parameter in the scenario
    - Issuing move intention above allowed max distance is invalid and ignored
    - Creates sound
- Sprint
    - Only intruders can sprint
    - Move happens on the straight line in the direction that an agent is facing
    - Agent can specify the distance
    - Senario specifies the max distance that an agent can move
        - The max distance can be modified based on the slowDown parameter in the scenario
    - Scenario specifies the cooldown period (number of turns) (see “General action semantics”)
    - Issuing sprint intention above allowed max distance is invalid and ignored
    - Creates sound
- Rotate
    - Agent specifies the angle
        - Negative angle results in anticlockwise rotation
        - Positive angle results in clockwise rotation
    - Senario specifies the absolute angle by which an agent can rotate
    - Issuing rotation intention above allowed max angle is invalid and ignored
- DropPheromone
    - (Side note: the semantics are modeled after Sprint where applicable)
    - The pheromone is dropped immediately after issuing the action

    - Scenario specifies the cooldown period (number of turns) (see “General action semantics”)
    - Pheromone details:
        - Both Guards and Intruders each have 5 pheromone types
        - Guards don’t smell intruder’s pheromones and vice-versa
        - Pheromones don’t mix
        - Pheromones have a radius, which decreases linearly with time
            - Scenario specifies radius of dropped pheromone
            - Scenario specifies in how many turns the pheromone disappears completely
        - Agents perceive only the distance to the pheromone
- Yell
    - Only guards can yell
    - Every agent perceives yell
        - The semantics of yell perception are defined by the sound system
- NoAction
    - NoAction is always valid
    - No action is always “executed”; specifically if an agent issues NoAction in the next turn the agent will perceive that action as executed

Collisions:

- Agent can see whether objects are concrete (not passable).
- ~~Agent-agent collisions are ignored (agents can pass through each other)~~ (updated on 21 Feb - agents are solid)
- The external borders of a map have semantics of walls (for vision, collision etc.)
- Controllers decide whether an action issued by an agent was valid
    - No part of an agent can collide with any part of an external object during the movement, or at the end state.
        - Implementation note: because agents are circles and motion is on a line, in practice collisions can be checked by:
            1. Checking whether the end state results in intersections
            2. Checking whether the rectangle that agent moved through has



intersections.
- Actions resulting in collisions are ignored:
    - the agent state stays unchanged, as if no action was issued

    - the agent is informed whether a previous action was executed in the next turn percept.

Agent scenario settings perception:

- gameMode
- captureDistance
- winConditionIntruderRounds
- maxRotationAngle
- maxMoveDistanceIntruder
- maxSprintDistanceIntruder
- maxMoveDistanceGuard
- sprintCooldown
- pheromoneCooldown
- radiusPheromone
- slowDownModifierWindow
- slowDownModifierDoor
- slowDownModifierSentryTower

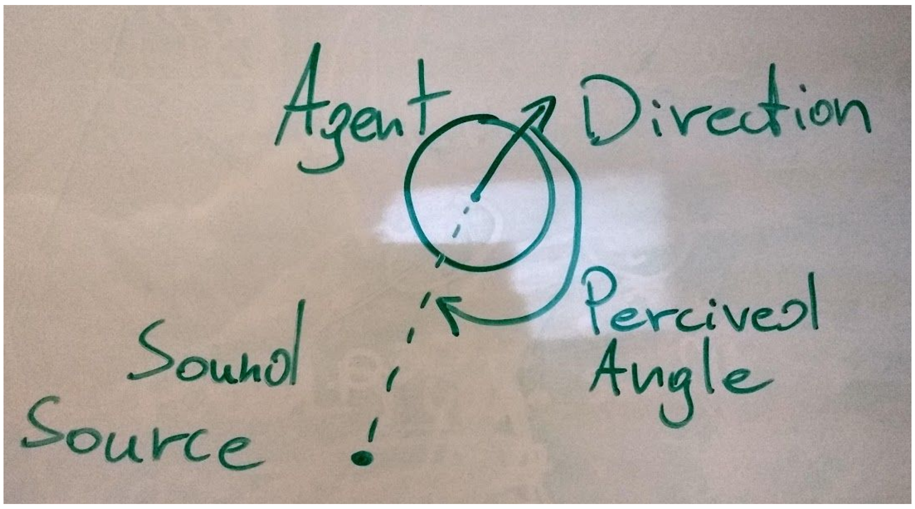
Vision:

- Vision is based on rays system (vectors) (also known as ray casting)
    - Agent perceives the interception of a ray with an object
    - Agents perceive the type of an object (even if the object is partly visual)
    - Agents do not perceive continuity of objects (i.e. they need to guess whether the points indicated by rays are connected)
    - The points are reported in relation to the agent (agent is not aware of the absolute position)
- There is finite amount of rays (vectors) defined in the scenario
    - One ray can result in perception of two or more points if the ray passes through not opaque objects
- Each agent has the same view angle defined in scenario
- Agent sees only in the direction it is facing (the percepts are computed based on the location of the agent independently of how agent got there)
- Shaded area is …
    - Before we set on the rays system, we decided that you can be either in or out of shaded area
    - You can not see into the shaded area from outside (like a wall), but you can enter it from any direction
    - If an agent is in the shaded area, they can see what’s in it but with limited vision
- Agents can not see beyond and into solid objects/shaded areas
- Guard tower (see pheromones and objects board)

- Use Project Manual and provided tutor recommendations for other constants/values (vision distance from tower, vision limits for sprinting etc…)

Sounds:

- Sounds penetrate all obstacles
- Two types of sounds:
    - Noise (just noise, not a specific noise)
    - Yell
- Sounds can be heard by every agent in a radius specific to each type of a sound
- Agents perceive sounds by
    - the type of a sound
    - direction (see figure for how the angle should be computed)
        - Each agent receives direction with an adjustment sampled from a uniform distribution in range (-10, 10) degrees



Objects semantics:

- In the map definition solid objects must not intersect i.e. one solid object must not be on top of another solid object
- Maps including solid objects intersection must result in error during map loading
- Groups are not allowed to introduce objects different from the objects defined here.

Objects:

(Make sure to have a look at: Interop.Percept.Vision.ObjectPerceptType in GameIntrop)

- Teleport (Stairs). See: Teleport semantics
- Walls
    - Solid, opaque objects.
- Shaded Areas
    - Shaded areas are like a “room” you can enter from every side with decreased view range. See vision for details.

- Sentry Towers
    - A sentry tower can be entered from every side.
    - An agent in a sentry tower can see over walls.
    - An agent in a sentry tower has an increased long range view but cannot see in shorter distances.
- Flags (Pheromones)
    - See pheromones
    - Decreasing Radius by time
    - Time Limit
    - Both can spawn a marker
    - No detection of the other's team marker
    - 5 types of pheromones
    - Intensity by range to coordinates
    - Radius
    - Pheromones don't mix
- Spawn Area
    - Agents are placed
    - Placements of agents on spawn area is the same as when teleporting
- Target Area

Solid objects:
- Agent
- Wall

Opaque Objects (Agents can not see through opaque objects at all):
- All solid objects

Object opaque from outside
(an agent can not see in the area when the agent is outside of the area):
- Door
- Shaded area
- Sentry tower

Doors and windows:
- Doors and windows are special areas
- The semantic of doors are the same as:
    - Shaded area
    - Noise Making area
    - Slowing down area
    - There are no special actions associated with doors
- The semantics of windows are the same as:
    - Noise making area
    - Slowing down area
    - There are no special actions associated with windows
    - Shaded area is explicitly not part of windows
- The effects of doors and windows are controlled by parameters of specific areas

Areas:

- Slow down area
    - When an agent enters in the slow down area then the agent max move distance and max sprint distance get multiplied by value specified in the scenario

Teleport semantics:

- There are two teleport areas connected together
- Teleports work both ways
    - After a teleport a flag is set for the teleported agent that prevents that agent from teleporting back
    - The “no teleport” flag is reset after you leave the teleport area
    - Agent can perceive the “no teleport” flag
- After teleportation an agent is placed in random (sampled uniform) coordinates of the connected area
    - In case of collisions with walls next to teleport target area the position is sampled again until there are no more collisions

Map file specification:

General coordinate system: x1,y1,x2,y2,x3,y3,x4,y4

General variables with example values:
gameMode = 0 (see “Victory Conditions”)
height = 80
width = 120
numGuards = 3
numIntruders = 2
captureDistance = 0.5
winConditionIntruderRounds = 3 (how many rounds does the intruder have to stay in the target area)
maxRotationAngle = 45.0 (in degrees)
maxMoveDistanceIntruder = 1.4
maxSprintDistanceIntruder = 4.0
maxMoveDistanceGuard = 1.4
sprintCooldown = 2 (rounds, integer)
pheromoneCooldown = 3 (rounds, integer)
radiusPheromone = 5.0
slowDownModifierWindow = 0.5
slowDownModifierDoor = 0.5
slowDownModifierSentryTower = 0.1
viewAngle = 45.0

```
viewRays = 45
viewRangeIntruderNomal = 7.5 (length of each vector)
viewRangeIntruderShaded = 4.0
viewRangeGuardNomal = 6.0 (length of each vector)
viewRangeGuardShaded = 3.5
viewRangeSentry = 2.0, 20.0 (not visible short range, visible high range)
yellSoundRadius = 30.0
maxMoveSoundRadius = 10.0
windowSoundRadius = 10.0
doorSoundRadius = 5.0
```

Area definitions:

```
targetArea = 20,20,40,20,20,40,40,40 (example)
spawnAreaIntruders = x1,y1,x2,y2,x3,y3,x4,y4
spawnAreaGuards = x1,y1,x2,y2,x3,y3,x4,y4
wall = x1,y1,x2,y2,x3,y3,x4,y4
wall = x1,y1,x2,y2,x3,y3,x4,y4
wall = x1,y1,x2,y2,x3,y3,x4,y4
wall = x1,y1,x2,y2,x3,y3,x4,y4
wall = x1,y1,x2,y2,x3,y3,x4,y4
teleport = x1,y1,x2,y2,x3,y3,x4,y4,tx1, ty1 (t = target coordinate)
shaded = x1,y1,x2,y2,x3,y3,x4,y4
door = x1,y1,x2,y2,x3,y3,x4,y4
window = x1,y1,x2,y2,x3,y3,x4,y4
sentry = x1,y1,x2,y2,x3,y3,x4,y4,ix1,iy1,ix2,iy2 (i = inside area)
```