Contents

[Contents 1](#_Toc451527713)

[1 Overview 2](#_Toc451527714)

[2 First Version limitations 2](#_Toc451527715)

[2.1 Bot 2](#_Toc451527716)

[2.2 GM 2](#_Toc451527717)

[2.3 Map 2](#_Toc451527718)

[2.4 Web client(website) 2](#_Toc451527719)

[3 Main Flows in Application 2](#_Toc451527720)

[3.1 Initialization Flow 2](#_Toc451527721)

[3.1.1 Bot initialization activity 3](#_Toc451527722)

[3.2 Bot move action flow 4](#_Toc451527723)

[3.2.1 Move activity 4](#_Toc451527724)

[3.3 Visualization flow 6](#_Toc451527725)

[3.3.1 Starting Flow 6](#_Toc451527726)

[3.3.2 Visualization flow 7](#_Toc451527727)

[4 Components 8](#_Toc451527728)

[4.1 Game mechanic server(GM) 8](#_Toc451527729)

[4.1.1 Functions 8](#_Toc451527730)

[4.1.2 API 8](#_Toc451527731)

[4.2 Bot 11](#_Toc451527732)

[4.2.1 Functions 11](#_Toc451527733)

[4.2.2.API 11](#_Toc451527734)

[4.3 Web client(website) 12](#_Toc451527735)

[4.4 Visualizer 12](#_Toc451527736)

[4.4.1 Functions 12](#_Toc451527737)

[4.4.2 API 12](#_Toc451527738)

[5 Deployment 14](#_Toc451527739)

[6 Tests 14](#_Toc451527740)

1. Overview

This technical specification is made for developers to implement the first version of a game.

2 First Version limitations

The first version has its limitations both in application design and game process. The full high-level design described in FRS\_4.docx

2.1 Bot

1. Bot can make only “move” action
2. Animals have following characteristics:
   1. Permanent speed = 3
   2. No hearing radius
   3. No visible radius, bot receives information on all enemies’ animals
3. When animal moves all other(enemy’s) animals can see the move.

2.2 GM

1. GM encapsulates receiving, sending, validating message functions (in full version these functions made by another modules).
2. GM interacts with bots using OSGi java call. In latest versions of the game the java call will be used for communication with other applications. Message queue will be used for GM-bots communication.

2.3 Map

1. Map is a part of GM in First Version. Lately it will be a separate application. Map has only “ground” tiles.
2. The size of the map is 500\*500(constants in GM application)
3. The map is not circular.

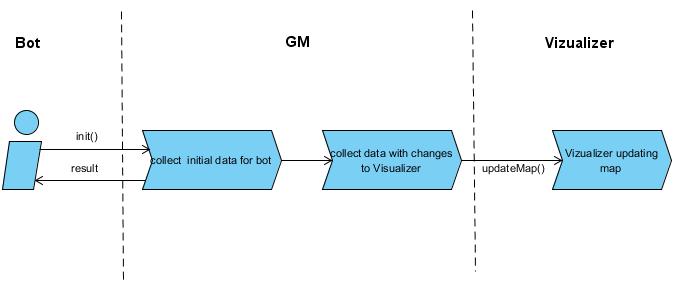
2.4 Web client(website)

1. Web client will show all map without possibility of zoom. All users see the same game.

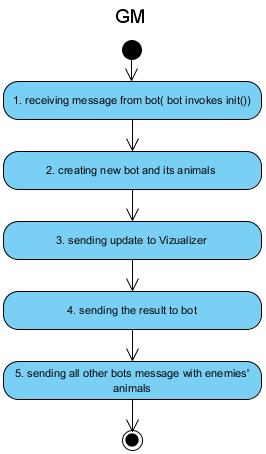
3 Main Flows in Application

3.1 Initialization Flow

Initialization flow describes adding new bot (and new animals) to the game



3.1.1 Bot initialization activity



Actions:

1. Receiving message from bot - bot invokes init() method of GM.
2. Creating new bot and its animals
   1. GM creates an id for bot(UUID) and add bot to GM’s bot list
   2. GM creates 4 animals for a bot and put them randomly on a map but no more than 10 tiles between them.
3. Sending update to Vizualizer using updateMap()
4. Sending the result to a bot.
   1. botId
   2. array with all animals on a map. If animal has field botId = this.botId than this is bot’s animal. If Animal has botId = null – this is enemy’s animal.

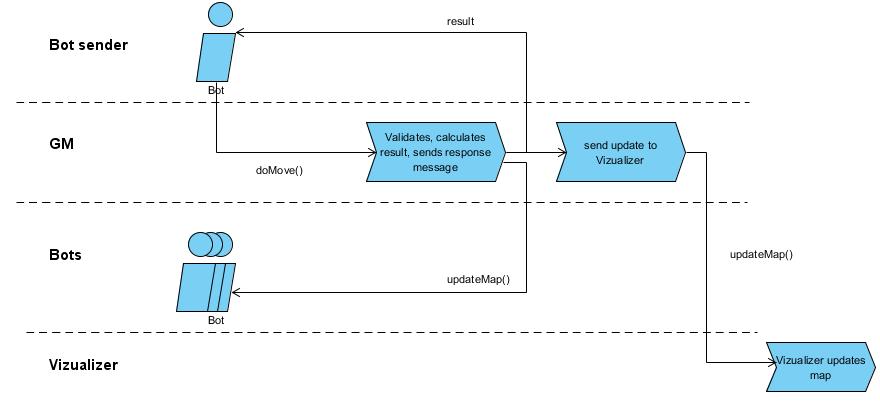
**Important:** before sending an array to bot all botIds of enemies’ animals should be replaced by null

* 1. map size

1. Invokes updateMap() method of all other bots(except of action sender bot). Bots receive a list of enemies’ animals.

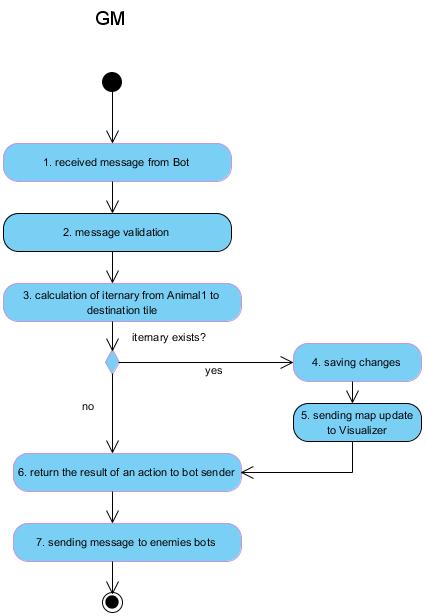
3.2 Bot move action flow

The flow describes interaction between bots and Game Mechanic server when bot decide to make an action.



Bot and GM communicate via OSGi java call.

3.2.1 Move activity



Actions:

1. Receiving message from bot - bot invokes doMove() method of GM.
2. Input parameters(message) validation
3. Checking all mandatory fields for emptiness
4. Check if this animal exists (Bot (botId) has an animal on a tile)
5. Check if all itinerary tiles exist
6. Check if the all itinerary tiles are empty (except of starting tile)
7. Check if the number of tiles in itinerary <= speed + 1
8. Itinerary calculation check.

Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | A |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

A – animal

|  |  |
| --- | --- |
|  | Possible destinations for animal with speed = 3 |

* 1. Check if destination tile is in possible destinations : the animal can move orthogonal, the number of passed tiles <= animal speed.

**Formula for checking**

(a1, a2) – row and column of start tile

(b1, b2) – row and column of destination tile

|a1 - b1| + |a2 - b2| <= speed

* 1. Check if tiles go one by one.

(a1, a2) – row and column of first tile

(b1, b2) – row and column of next tile

|(a1+a2) – (b1+b2)| = 1

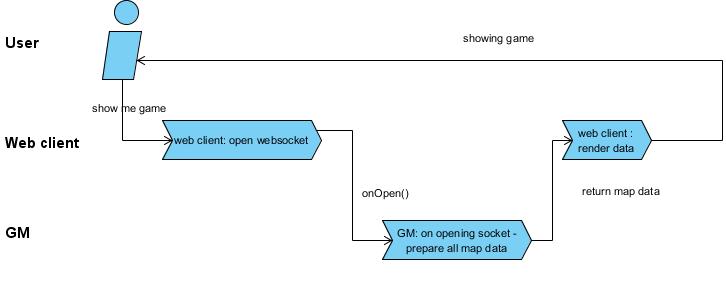
1. Saving changes – save animal with a new address
2. Sending data to Visualizer
3. Send the result of an action to bot-sender. Bot-sender receives a list of animals moved.
4. Invokes updateMap() method of all other bots. Bots receive a list of animals moved.

3.3 Visualization flow

The flow describes the visualization process of the game. The process uses websockets for pushing information to frontend.

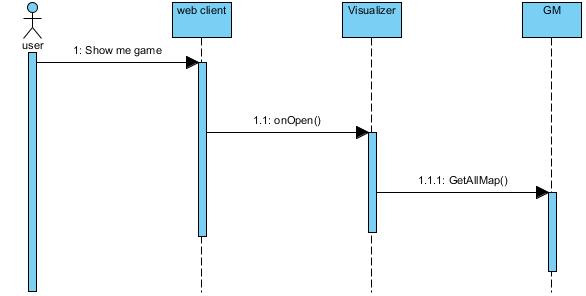
3.3.1 Starting Flow

The flow describes start of the visualization. The same flow is using on restarting visualization.



1. User presses button show game (or on websocket event onError for restarting).
2. Web client opens websocket
3. Server(vizualizer) gets data from GM using getAllMap() and answer with map size (part to be shown, in First version it will be all map), array with animals
4. Web client puts this data to rendering engine to render the game
5. User sees the game

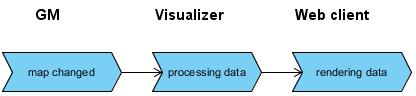
Operation sequence:



Steps:

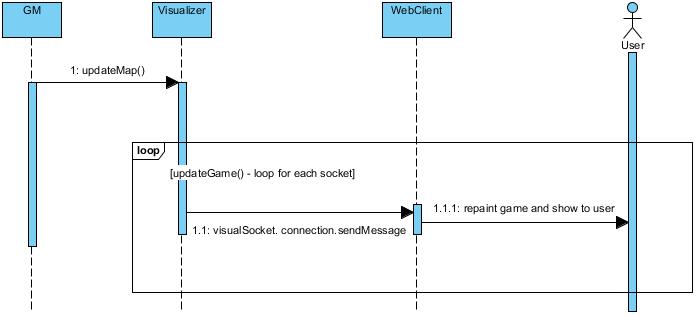
1. User open a game
   1. Web client creates and opens websocket
      1. Visualizer get all needed data from GM

3.3.2 Visualization flow



1. Map changed because of animals moving
2. GM invokes visualizer method updateMap()
   1. Visualizer pushes data to web client using opened websocket.

Operation sequence:



Steps :

1. UpdateMap() – GM invokes updateMap()
   1. updateGame() - For each socket Visualizer pushs data to webClient using visualSocket.connection.sendMessage()
      1. Web client repaint game

4 Components

4.1 Game mechanic server(GM)

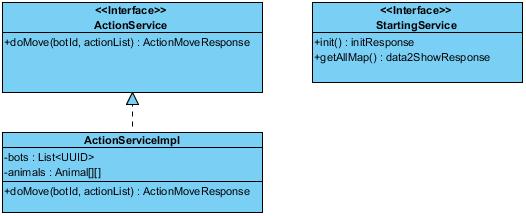
application processing physics of the “world”.

4.1.1 Functions

1. Process move action

4.1.2 API

GM has 2 interfaces.



1. ActionServiceImpl – API for bots’ actions
   1. bots - - list of botIds;
   2. animals - Array of array Animal. Array size 500\*500. Row and column in arrays – is an address of an Animal on a map(tile).
   3. doMove()

Input parameters for doMove()

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| botId | UUID | yes | Id of a bot |
| actionList | List<ActionMove> | yes | Contains list of move actions |

Type ActionMove

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| animalId | UUID | yes | The reference id will be returned in response message |
| itinerary | List<Tile> | yes | Animals Itinerary to destination tile  Ex: start tile (2,2) dest tile (3,4) , speed =3, valid itinerary: (2,2),(2,3),(2,4),(3,4) |

Type Tile

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| rowId | int | yes | Row number of tile |
| colId | int | yes | Column number of tile |

output

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| response | ActionMoveResponse | no | Encapsulates return parameters |

Type ActionMoveResponse

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| myAnimals | List<Animal> | no | Bot’s animals with changed addresses(not all animals could be moved) |

Type Animal

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| animalId | UUID | no | The reference id from request message. Mandatory for bot-sender. |
| botId | UUID | No | Animal’s bot id |
| speed | int | No | Speed in tiles. Speed = 3 in this version |
| address | Tile | yes | Address(row and column of tile) |

1. StartingService – API for starting components
   1. init()

input – void

output

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| response | InitResponse | yes | Encapsulates return parameters |

InitResponse type

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| botId | UUID | yes | Id of a bot |
| animals | Array of array of Animal\* | yes | All animals on a map. Indexes of array – is an animal address on a map |
| maxRow | int | yes | The max row of a map(499 for first version) |
| maxCol | int | Yes | The max col of a map(499 for first version) |

\* - Animal type described above

b. getAllMap()

Input : void

output

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| response | GetAllMapResponse | yes | Encapsulates return parameters |

GetAllMapResponse type

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| maxRow | int | yes | The max row of a map(499 for first version) |
| maxCol | Int | Yes | The max col of a map(499 for first version) |
| Animals | Array of array of Animal (Animal[][]) | No | Animals on a map. Indexes of array – is an animal address on a map |

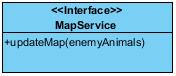
4.2 Bot

Application with AI. It has a number of animals and it “rules” the animals.

4.2.1 Functions

1. Storing information on a map, animals
2. Decide to make an action
3. Calculate itinerary for action (shortest or “smartest” way from start tile to destination tile.) All tiles in the itinerary should be empty.
4. Send a message with an action to GM
5. Receive a message with a result of its action
6. Receive a message on other animals moves.

4.2.2.API



1. updateMap()

Input for updateMap()

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| enemyAnimals | List<Animal> | no | Enemy’s animals |

Type Animal

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| animalId | UUID | no | The reference id from request message. Mandatory for bot-sender. |
| Address | Tile | yes | Address(row and column of tile) |

Type Tile

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| rowId | Int | yes | Row number of tile |
| colId | Int | yes | Column number of tile |

4.3 Web client(website)

Web client - is a frontend application written on html and javascript. Part of the webclient is a rendering engine. Rendering engine is used to show a game to user.

Rendering engine uses phazer.io library.

Incoming message for updating map

{"bots" :[{"botId":"fjfjfjfjf", "animals": [ { "row" :1, "col":1}]}, {"botId":"fjfjfjfjf", "animals": [ { "row" :1, "col":1}]}]}

Incoming message to visualize all map

{"maxRow" : 500,"maxCol" : 500,"bots" :[{"botId":"fjfjfjfjf", "animals": [ { "row" :1, "col":1}]}, {"botId":"fjfjfjfjf", "animals": [ { "row" :1,"col":1}]}]}}

4.4 Visualizer

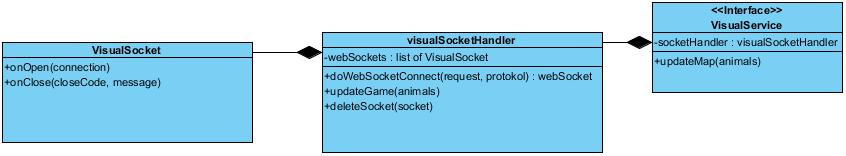
- is an application for game visualization. It is also a backend for a webclient(website)

4.4.1 Functions

1. Processing data for rendering engine
2. Push data to rendering engine

Vizualizer uses **jetty websockets** to push data to webclient.

4.4.2 API



1. VisualService – API for visualization. GM invokes updateMap() to push map changes to visualizer.
   1. updateMap(animals) – visualizer invokes socketHandler.updateGame().

Input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| Animals | List<Animal> | yes | Moved animals |

Output void

1. VsualSocketHandler extends org.eclipse.jetty.websocket.WebSocketHandler. The class used for handling websocket connections from multiple clients.
   1. webSockets – list of VisualSockets. One socket per client.
   2. doWebSocketConnect() - invoked on new connection. This operation creates new webSocket.

Input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| request | HttpServletRequest | yes | Servlet request |
| protocol | String | Yes | Possible values: wss, ws |

Output

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| socket | VisualSocket | yes | New web socket |

* 1. updateGame(animals) – send message with game changes to every websocket

Input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| animals | List<Animal> | yes | Moved animals |

Output void.

* 1. deleteSocket(socket) – remove socket from list of webSockets

input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| socket | VisualSocket | yes | web socket to remove |

Output void.

1. VisualSocket implements org.eclipse.jetty.websocket.WebSocket.OnTextMessage. Visual socket is a web socket for connection
   1. onOpen() – This operation invokes getAllMap() method of GM

input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| connection | Connection | yes | Connection for websocket |

Output – void

* 1. onClose() - This operation deletes websocket from list of websockets.

input

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Mandatory | Description |
| closeCode | Int | yes | Close code |
| message | String | No | Message on close |

Output - void

5 Deployment

All applications runs in OSGi container. Gm, bots, vizualizer run in one root container for the first version of the game.

1. Start Apache Karaf
2. Install and run GM bundle and bot’s bundles in root container

6 Tests

1. Integration tests
   1. For API
2. Unit tests for public methods