[Components](#h.gjdgxs)

[Animal bot](#h.pio09w3ey531)

[Game mechanic](#h.75eribbvf8k)

[Visualizer](#h.2qlim2jg5l0q)

[Map](#h.o5mcl4keyf8)

[Website](#h.bctvu043f6lt)

[Logging/loader server](#h.30j0zll)

[Score calculator](#h.uy6kx2j8k9o4)

[Replication database (NoSQL)](#h.1fob9te)

[Game data database (SQL DB)](#h.no3ik08zsepm)

[Proxi](#h.1a6s97yuyyej)

[GM message broadcaster](#h.iuvn0jufktaf)

[Changes Buffer](#h.if14bplr6kni)

[Components interaction](#h.2et92p0)

[Login/bot loading process](#h.1dks8rw5w2n7)

[Overview](#h.k5zjaqv1vaah)

[Loading starter package subprocess](#h.erycjg6yuspj)

[Game process](#h.6bv7vemear1r)

[Overview](#h.i7fzt3f1n4m3)

[Detailed description](#h.uiyplqafp7n1)

[GM action loop](#h.wr5oz7rxvnic)

[Data(changes) to be send:](#h.jsmxwu42qygh)

[Action types](#h.rqzu7cnrgqjd)

[GM main loop](#h.w1w2uct0mar)

[Alternative way of game process](#h.4d34og8)

[Visualization process](#h.17dp8vu)

[Overview](#h.98ofjeezkgs5)

[Detailed description](#h.lnxbz9)

[Starting visualization](#h.p8ymw0j3tqqq)

[Visualization process after starting](#h.40800lc7xv92)

[Testing](#h.xhdgn3bwgpg)

[open issues](#h.f2iy1qz67t8j)

# 

# Components

## Animal bot

Bot is a program written by the user. The bot program is running on our servers. User logins to the web site, downloads the starter package, writes the code, uploads it to the website.Bot rules all animals of its breed. World provide start data when bot starts the game. Bot’s “knowledge” of the “world” updates in 2 ways:

1. Action of the bot:
   1. Bot decide to make an action.
   2. Bot sends a message with an action to the “world”
   3. “World” calculates the result and sends all changes to bot
2. Receiving data from the “world”. World can decide to send changes to the bot in case of:
   1. [Other animal moves to bot’s visibility/hearing radius](#h.s0716z6g6d6g)
   2. [At this server step “world” made changes in bot’s animals’ life(newborns, deaths, food)](#h.x5gdtvm82agy)

Every bot must do:

1. Receive message with data
2. send message with action

Animal properties – all objects, can be add to bot’s map propertiesMap

Messaging queue – is the transport between bots and game mechanic server.

## Game mechanic

GM is an application ruling the”world”.GM starts all components. GM calculates the result of all bot’s action, calculates the data to be send to all components. GM processes physics of the “world”([GM main loop](#h.x5gdtvm82agy)).

1. Server sends properties update to the animals:
   1. Health
   2. Energy
   3. Pregnancy status
   4. Location
   5. Speed
2. server send info to bots:
   1. visual
      1. Terrain
      2. Other animals
   2. Audio
      1. Other animals’s voices(noise)

GM have a list of animals for each bot.

bot {list{Animals{address, list{Animal properties}}}}

food and it’s addresses

## Visualizer

Visualizer is an application, part of website backend. It’s a “backend” for a webpage with a game. It receives changes from GM server, processes them, and sends to the webpage. It’s not a database, Visualizer uses database only when starts(restarts).

## Map

* is an object, part of Game mechanic server. It should be loose coupled with the other parts of GM, because it could be singled out in an independent part. Initially it consists of pregenerated tiles.

Map included:

1. pregenerated tiles List{Tile}

## Website

User can login, download starter package for bot, upload bot, see the game

Page where game is running:

1. Map with animals
2. Life points(health)
3. Energy
4. Zoom in, out
5. Number of animals for each user’s breed on a timeline like a graph

## Logging/loader server

Signing up, logging user, check possibility of connection to the game

Save the user's code on server, check it for malformed code, choose compiler and compile the code. Run the code, register it in a messaging queue and configs.

## Score calculator

number of animals

## Replication database (NoSQL)

Every n-minutes store game state: number of animals, it’s place, map etc. Needed if server crashed. NoSQL like MongoDb

## Game data database (SQL DB)

stores information like logins, scores, params of the game etc. SQL DB.

## Proxi

is an application transmitted messages from Bot to GM and otherwise. It superwises the order of actions of the bot. Only when the answer with previous action received bot can send a message with a new action

## GM message broadcaster

is an application, which sends messages from GM to subscribers.

## 

## Changes Buffer

is an application. It accumulates changes in large chunks and storing them to DB.

# Components interaction

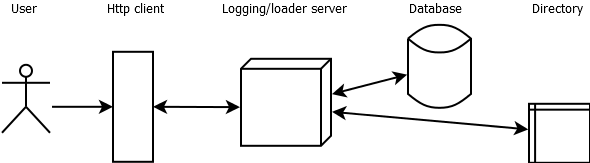
## Login/bot loading process

### Overview

Components interaction in logging/bot loading process described on the diagram. The user takes only steps 1-2 if just logging.

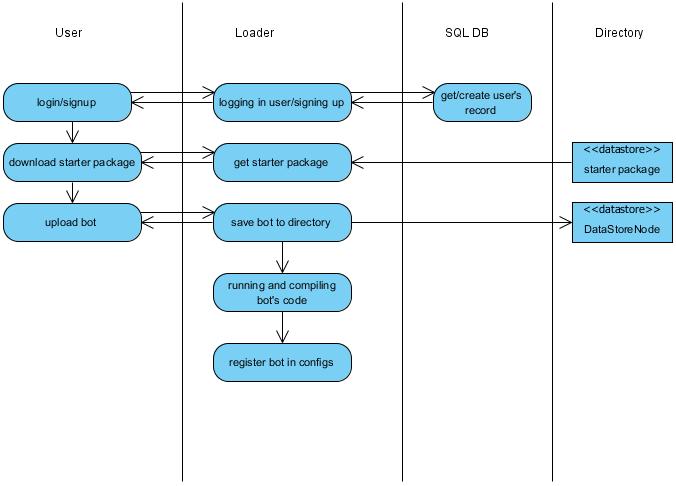
Steps:

1. User send a login request
2. Server checks the database and authorize user
3. User download the starter package
4. User upload the bot
5. Server saves the bot to directory
6. Server check the bot, choose compiler and compile the code, run the code, register it in a messaging queue



## 

### Loading starter package subprocess



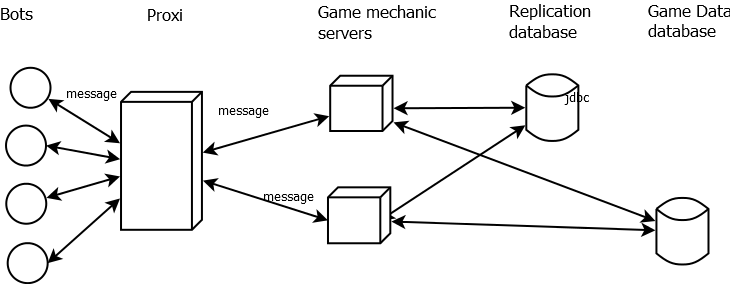
1. User sends a login/signup request
2. Loader requests user record from database, authorizes the user
3. User downloads the starter package, develops the bot
4. User uploads the bot
5. Loader saves the bot to directory
6. Loader checks the bot, chooses compiler, compiles and runs the code
7. Loader registers it in a messaging queue

## Game process

* is the process where game is playing. Animals are born, breed, die, make different actions. “World” calculates results of actions and send them to all participants.

### Overview

Components interaction in the game process described on the diagram.



Steps:

1. Bots send messages to proxi.
2. Proxi receives a message, checks it for cheating, possibly packs/unpacks it, checks if previous action is done, and sends message to game mechanic server
3. Game mechanic server receives a message, checks possibility, sends request to Game Data DB if needed, fulfills it and sends return message if needed.
4. Game mechanic saves game state every n-minutes to replication database. It’s a resource consuming process, sometimes a replication app server needed for this operation.

### Detailed description

GM has 2 loops:

1. GM action loop
2. GM main loop

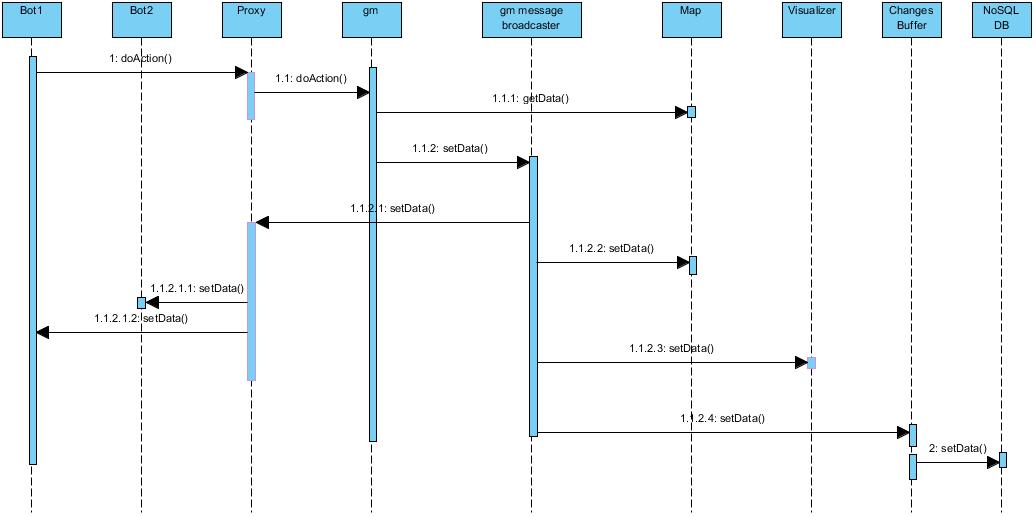
### GM action loop

Animals send actions to the server. Each step –one action only. Actions can be:

* 1. Move
  2. Attack
  3. Eat
  4. Breed
  5. (Make noise?)
  6. Wait
  7. More??

GM processes the messages with actions from bots one by one. Every cycle has phases :

* 1. receiving message from bot
  2. calculation of the result
  3. sending changes all subscribers. Data to be send depends on action type and subscriber(see the description below)



## 

Methods:

1.doAction() - bot sends a message with action to be done to Proxy. If bot don’t have unfinished action, Proxy resend message to GM. Types of action are described below.

1.1 doAction() - Proxy send a message from bot to GM.

1.1.1 getData() - GM gets data from Map and calculates the result of an action.

1.1.2 setData() - GM sends result of an action to GM message Broadcaster

1.1.2.1 setData() - Broadcaster sends the result of an action to Proxy

1.1.2.1.1 setData() - Proxy sends the result of an action to all influenced bots

1.1.2.1.2 setData() - Proxy sends the result of an action to all influenced bots

1.1.2.2 setData() - Broadcaster sends the result of an action to the Map

1.1.2.3 setData() - Broadcaster sends the result of an action to the Vizualizer

1.2.3 setData() - GM message broadcaster sends the result of an action to all bots influenced by the action

1.1.2.4 setData() - Broadcaster sends the result of an action to the Changes buffer.

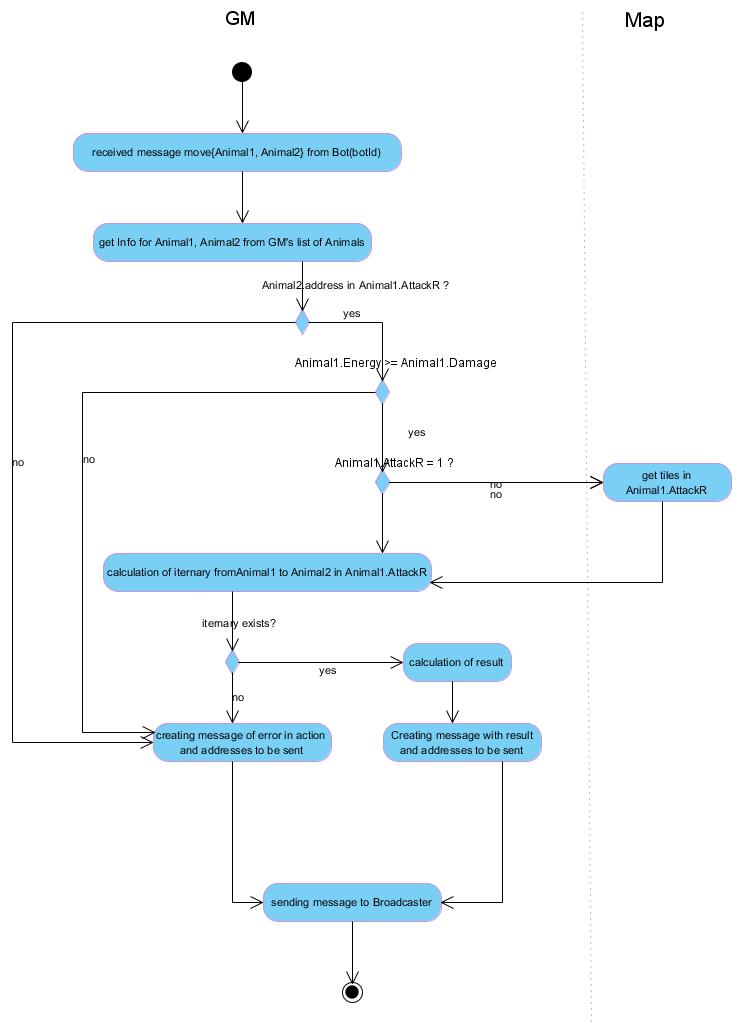
2 setData() - Changes buffer has a capacity. It’s a param in a database. When changes buffer is full it stores changes to the database.

### Data(changes) to be send:

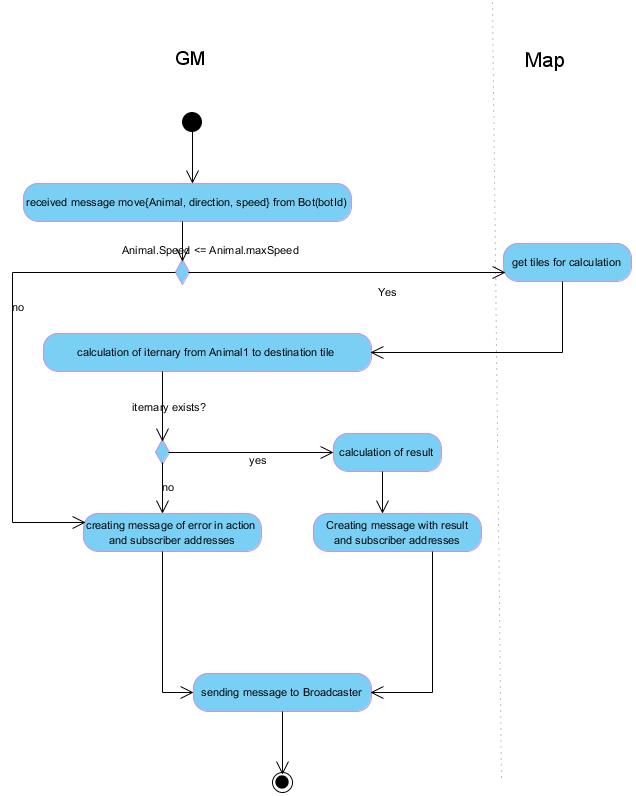
1. Attack
   1. messaging bot. Attacking animals were moved, it means their visibility and hearing radius changed. Format: json
      1. map in visibility radius of a bot (only the part not visible before)
      2. addresses of bot’s attacking animals
      3. enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      4. enemy’s animals and there’s addresses in visibility radius of bot’s animals
      5. food in visibility radius of bot(only food not visible before)
   2. all other bots. Enemy’s animals were moved, they may become visible or audible. Format: json
      1. all enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      2. all enemy’s animals and there’s addresses in visibility radius of bot’s animals
      3. bot’s dead animals, survived animals with updated health
   3. Сhanges buffer, Visualizer Format: json
      1. addresses of attacking animals
      2. dead animals, survived animals with updated health
2. Move
   1. messaging bot. Format: json
      1. map in visibility radius of a bot (only the part not visible before)
      2. addresses of bot’s moved animals
      3. enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      4. enemy’s animals and there’s addresses in visibility radius of bot’s animals
      5. food in visibility radius of bot(only food not visible before)
   2. all other bots. Format: json
      1. all enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      2. all enemy’s animals and there’s addresses in visibility radius of bot’s animals
   3. changes buffer. Format: json
      1. addresses of moved animals
3. Eat.
   1. messaging bot. Format: json
      1. map in visibility radius of a bot(only the part not visible before)
      2. bot’s eating animals and their addresses
      3. enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      4. enemy’s animals and there’s addresses in visibility radius of bot’s animals
      5. food in visibility radius of bot(only food not visible before)
      6. amount of food left in food source after eating
   2. all other bots. Format: json
      1. all enemy’s animals’ addresses in hearing radius of bot’s animals(rules applied described in [Physics section of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))
      2. all enemy’s animals and there’s addresses in visibility radius of bot’s animals
   3. changes buffer, visualizer Format: json
      1. addresses of moved animals
4. Breed
   1. messaging bot, changes buffer. Format: json
      1. pregnant animal

#### Action types

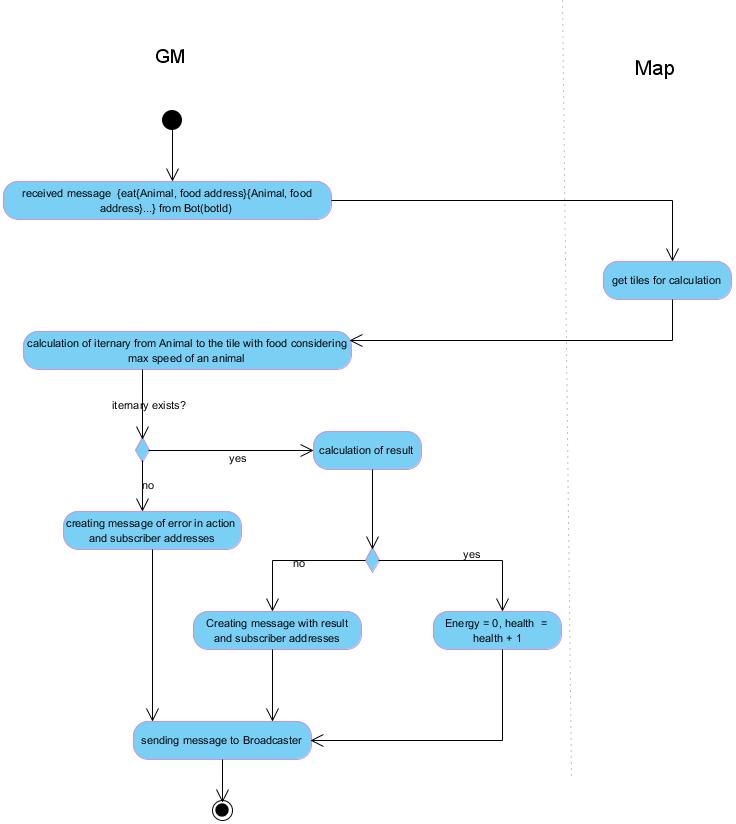
1. attack. Message format: {attack{AttackingAnimal1,AttackedAnimal1}Attack{AttackingAnimal2,AttackedAnimal2}...}.There could be more than one pair of animal. [Formulas](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4) for result calculation are in First version PRD



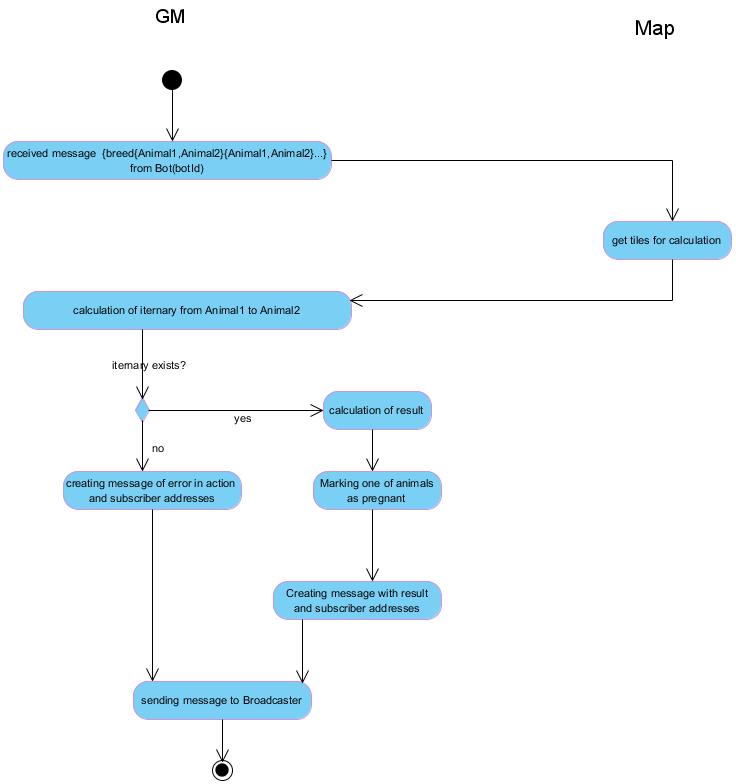
1. move. Message format:{move{Animal, destination tile, speed}{Animal, destination tile, speed}...}. [Moving rules](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4) are in FirstVersion PRD



1. eat. Message format: {eat{Animal, food address}{Animal, food address}...}

Eating [rules](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4) are in FirstVersion PRD

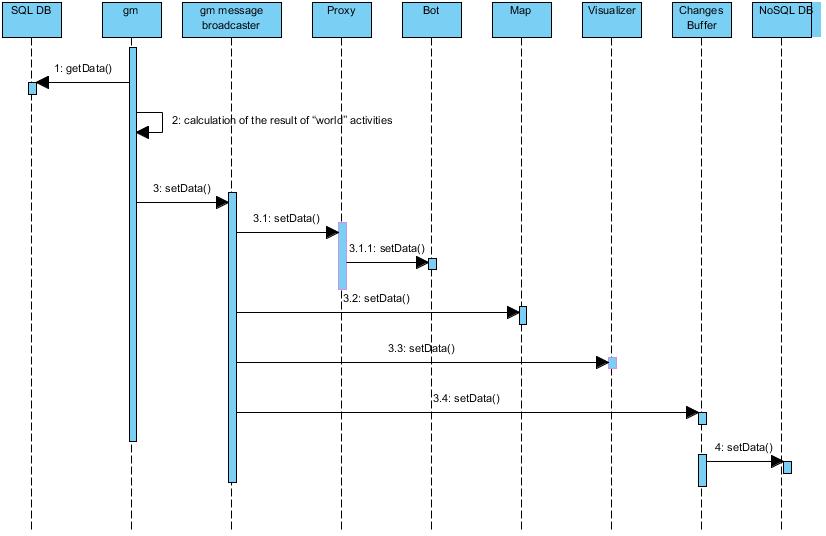
1. breed. Message format: {breed{Animal1,Animal2}{Animal1,Animal2}...}



### GM main loop

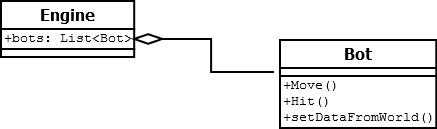
GM processes physics of the world(GM main loop). Every cycle of this loop called server step.

1. Phases of step :
   1. get data from database(optional)
   2. calculation of the result of “world” activities
   3. sending changes all subscribers
2. Types of “world” activities. Rules for activities described in [Physics section of First version PRD.](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4) Activities can be run simultaneously.
   1. pregnancy and newborns
   2. death of animals reached max age
   3. food managing
3. Types of data(changes) to be send:
   1. newborn animals
   2. dead animals
   3. addresses of new food
4. Server steps has numeration - “time of the world”.
5. Step runs every N milliseconds - configurable param in DB
6. Every N step GM gets config params from DB
7. Config params in DB:
   1. STEP\_SLEEP - milliseconds server sleep between steps
   2. STEP\_GET\_PARAMS\_FROM\_DB - number of steps to get updates param in DB
   3. STEP\_FOOD\_PER\_ANIMAL - pieces of food per Animal(for food source calculation)
   4. Table of size dependencies(described in [Game Analysis of First version PRD](http://drive.google.com/open?id=1TzBDARH73YmsVQTWS3HZnNSJ_n7de38L-NZgNLCSZy4))



### Alternative way of game process

Game mechanic could be done simpler if we don’t expect high load and no need in thorough checking for malicious code. We can put list of bots to game mechanic server and access them by reference.

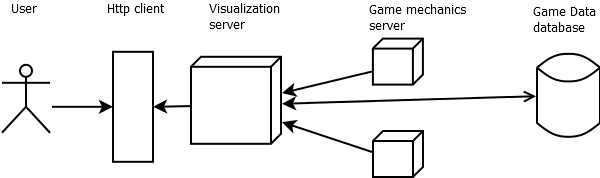


## 

## Visualization process

* is a process of visualizing the game on a website.

### Overview



Steps:

1. Visualization server gets the map data from replication database(optional)
2. Visualization server gets the current game state from the game mechanics server(via broadcaster)
3. Visualization server send data to http client(webpage)

### Detailed description

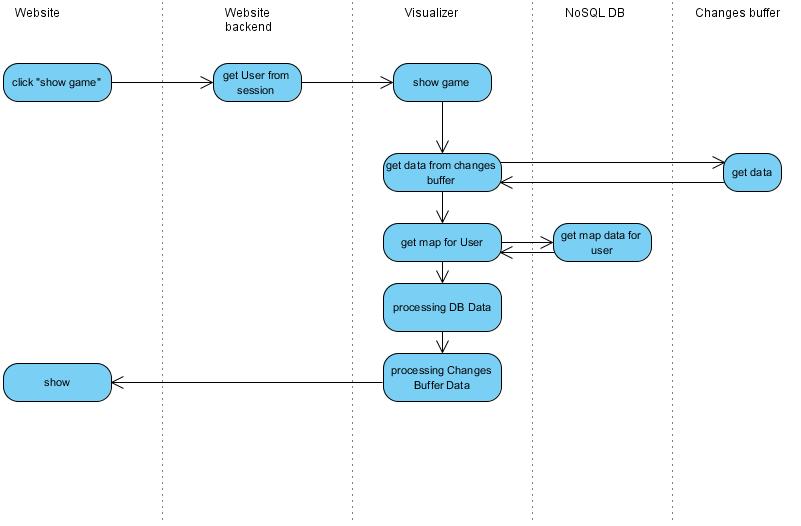
Main components of the process:

1. Webpage with rendering engine(javascript). Rendering engine provides graphics for the game.
2. Visualizer - provides data to be visualized.

Visualization principles

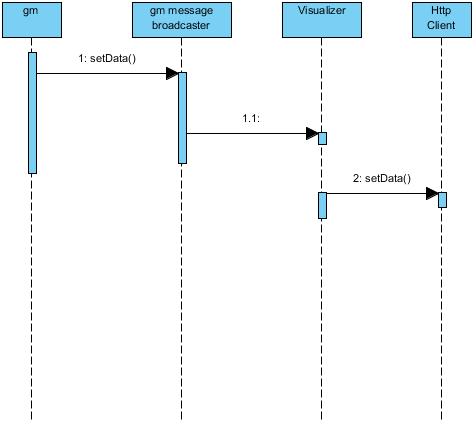
1. User can see only part of the map.
2. Zoom is 4x. Zoom makes map smaller/bigger, and maybe less detailed.
3. Visualizer is pushing data to frontend to visualize the game.
4. Visualizer pushes data on arriving (except of a starting process).
5. When Visualizer unpredictably restarts the starting process is using.

#### Starting visualization



1. User clicks “show game”, website sends request to backend.
2. Backend processes the request. Get User from session if exists. Send user and request to Visualizer.
3. Visualizer calculates part of the map to show. This part depends on user. (Above all user wants to see its bot )
4. Visualizer gets data from changes buffer.
5. Visualizer gets data from NoSQL DB according to the coordinates of needed part.
6. Visualizer makes DB data suitable for rendering engine.
7. Visualizer makes buffer data suitable for rendering engine.
8. Engine renders the game

#### Visualization process after starting



Methods:

1: setData() - GM sends changes to GM message broadcaster.

1.1 setData() - GM message broadcaster sends data to Visualizer.

2: setData() - Visualizer processes changes and sends them to the webpage. This step is not that easy because there’re a lot of threads on a visualizer component servicing different clients. Not all clients need arrived changes. Changes are a shared resource. But all this issues are solving on later maybe even development stage.

## 

## 

## 

# Testing

Loading test

* 1. We should write simulation to the game that includes:
     1. Bots
     2. Multi clients
  2. This needs to be planned.