# Rapid backend prototyping for a geolocation-based mobile game

**With OpenResty, Redis and Docker**

LogicEditor

Alexander Gladysh <ag@logiceditor.com>
@agladysh

FOSDEM 2017

## Talk plan

## About me

- Developing software since 2002
- Most of the time in gamedev
- Beyond that: high-load internet solutions, enterprise software etc. etc.
- Organizer of meetup.com/Lua-in-Moscow.
  Come to our Lua-related conference in Moscow on March 5th!

## Mobile games with geolocation

## The Goals

- To try out a number of approaches to the gameplay, to generate new ideas, to figure out what is fun and what is not.
- To get as cheaply as possible the framework that would allow to iterate over gameplay variants as fast as possible.
- To figure out technical limitations of the genre in practice.

## Results

A geolocation game server-side prototype along with a rudimentary client-side was developed in less than 100 man-hours (2 calendar months)

We’re rapidly iterating over the gameplay options and develop the technology part of the project.

## What is this talk about?

- This talk is about the technology part of the project,
- not about game-design
- or monetization.

It now is easier when ever to develop geolocation-based games.

I will show where to begin.

## First prototype gameplay

- The player is searching for the mobs placed on a map by walking around;
- Player has a set chance to catch the found mob.
- Caught mobs increase a stats counter in player characteristics.
- Caught mobs disappear from the map, but respawn at the same place after a set time.
- Admin users may add new mobs on the map.

## The Stack

- Server:
    - Redis,
    - OpenResty,
    - Docker.
- Client:
    - A single-page web application (in browser),
    - HTML5.

## Redis

- A reliable, proven solution.
- Supports geoposition out of the box:
  - `GEOADD key longitude latitude member`
  - `GEORADIUS key longitude latitude radius m`
- Useful set of primitives to store game objects in.
- ”Stored procedures” in Lua.

## OpenResty

- An nginx distributive supporting Lua, Redis and many other things out of the box.
- Very fast, rather friendly and well-maintained.
- Useful both for quick prototyping and production environment.

## Docker

- Reproducible cross-platform development environment.
- Efficiently relieves the development environment pain set up.
- Can be quickly turned into a prototype of production environment (it is arguable if it is suitable for real production).
- Has to be updated to a sufficiently recent version.

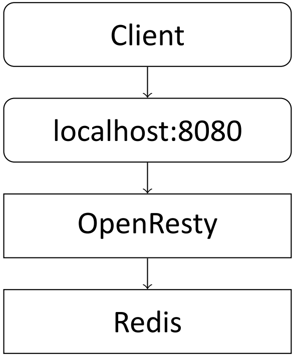
## Browser, HTML5

- Server-side during early stages of development is more important.
- Don’t implement what can be imagined while play-testing, like augmented-reality combat and other frills.
- HTML5 is good to implement ”quick and dirty” client.
- NB: Geolocation data access is limited, but sufficient.

## Docker: how to install on Ubuntu

- Traditionally, Ubuntu version is outdated.
- Don’t do `wget | sh`.
- Install `docker` и `docker-machine` from docker’s apt-repository (see manual).
- `docker-compose` is to be installed by `pip install`.
- Read the manual for the installation on other platforms.

## Docker on the developer’s machine



## docker-compose.yml for development: Redis

```
version: "2"
services:
  redis:
    image: redis
    volumes:
      - ./redis:/data
    command: redis-server --appendonly yes
  openresty: <...>
```

## OpenResty: interesting parts of nginx.conf (reduced)

```
error_log logs/error.log notice;
http {
  include resolvers.conf;
  lua_package_path "$prefix/lualib/?.lua;;";
  lua_code_cache off; # TODO: Enable on production!
  server {
    listen 8080;
    include mime.types;
    default_type application/json;
    location / { index index.html; root static/; }
    location = /api/v1/ { content_by_lua_file 'api/index.lua'; }
  }
}
```

## OpenResty: Dockerfile

```
FROM openresty/openresty
COPY bin/entrypoint.sh /usr/local/bin/openresty-entrypoint.sh
COPY nginx/conf /usr/local/openresty/nginx/conf
COPY nginx/lualib /usr/loca/openresty/nginx/lualib
COPY nginx/lua /usr/loca/openresty/nginx/lua
COPY nginx/static /usr/loca/openresty/nginx/static
ENTRYPOINT /usr/local/bin/openresty-entrypoint.sh
```

## OpenResty: entrypoint.sh

```
#!/bin/sh
grep nameserver /etc/resolv.conf \
  | awk '{print  "resolver " $2 ";"}' \
  > /usr/local/openresty/nginx/conf/resolvers.conf
/usr/local/openresty/bin/openresty -g 'daemon off;' "$@"
```

## docker-compose.yml for development: OpenResty

```
<...>
openresty:
  build: .
  ports:
    - "8080:8080"
  volumes:
    - ./nginx/lualib:/usr/local/openresty/nginx/lualib:ro
    - ./nginx/api:/usr/local/openresty/nginx/api:ro
    - ./nginx/static:/usr/local/openresty/nginx/static:ro
  links:
    - redis
```

# API calls

- Player calls:
    - `/` get game world state,
    - `/go/:go-id/` get game object state,
    - `/go/:go-id/act/:action-id` do action.
- System calls:
    - `/register` create a player,
    - `/reset` factory-reset the DB,
    - `/patch` upgrade the DP to the current version.
- NB: We’re not implementing a back-office UI to save effort, but using in-game mechanics for the in-game management.

## Game Object

- From the server’s point of view, game world consists of game objects.
- A game object has numeric characteristics and a list of actions.
- A game object may have coordinates.
- Position-less game objects must either belong to other objects or be used as their prototypes.

## Prototype chain

- A game object might have the prototype.
- A prototype game object might also have the prototype.
- A game object derives its characteristics and actions from its prototypes.

## Characteristics

- A characterisic is a named numeric property of a game object.
- If a game object does not have given characteristic set, its value is taken from the closest prototype in chain that has it set. If no such prototype found, the value is 0.

## Actions

- Game object action is a key from the Lua table of action handlers.
- A player may initiate action from a game object if he has sufficient permissions.

## A Green Toad Mob

```
{
  id = 'proto.mob.collectable';
  chrs = { respawn_dt = 10 * 60 };
  actions = { 'mob.collect' };
};
{
  id = 'proto.mob.toad.green';
  proto_id = 'proto.mob.collectable';
  chrs = { escape_chance = 0.25 };
};
{
  id = 'fa2eb7bca46c11e6be447831c1cebc82';
  proto_id = 'proto.mob.toad.green';
  geo = { lat = 55.7558, lon = 37.6173 };
};
```

## Action: Catch the Mob

```
ACTIONS['mob.collect'] = function(target, initiator)
  if
    math.random() * initiator.chrs.collect_skill >
    target.chrs.escape_chance
  then
    -- Inc number of catches for this mob type
    go_inc_chr(initiator.id, target.proto_id, 1)
    go_schedule_action_initiation( -- Schedule respawn
      target.chrs.respawn_dt, 'mob.spawn',
      { proto_id = target.proto_id, pos = target.pos },
      initiator.id
    )
    go_remove(target.id) -- Mob is caught, remove
  end
end
```

# Delayed Action Execution

```lua
local timestamp = os.time()
local action_ids = redis:zrangebyscore('da', '-inf', timestamp)
for i = 1, #action_ids do
  -- Execute action_ids[i] action
end
redis:zremrangebyscore('da', '-inf', timestamp)
```

## Player

```
{
  id = 'proto.user';
  chrs = { vision = 100, reach = 50 };
};
{
  id = 'user.1';
  geo = { lat = 55.7558, lon = 37.6173 };
  chrs = { collect_skill = 0.5 };
};
```

## Item: Admin Hat

```
{
  id = 'proto.item.wearable';
  actions = {
    ['item.don'] = { enabled = true };
    ['item.doff'] = { enabled = false };
  };
}
{
  id = 'proto.item.admin-hat';
  proto_id = 'proto.item.wearable';
  grants = { 'user.admin' };
  chrs = { collect_skill = 0.25 };
};
```

## Let’s issue Admin Hat to the player

```lua
local hat = go_new('proto.item.admin-hat')

assert(go_get('user.1').stored[1] == nil)

go_store('user.1', hat.id)

assert(go_get('user.1').stored[1] == hat.id)
```

## Storing game objects (”storage”)

- A game object (”container”) might ”store” other game objects.
- Stored objects are not ”visible” outside of their container.
- Player (a game object too) may initiate actions from the game objects he directly stores.
- Stored game objects don’t affect characteristics of their containers.

## Admin Hat actions

```
ACTIONS['item.don'] = function(target, initiator)
  go_unstore(initiator.id, target.id)
  go_attach(initiator.id, target.id)
  go_disable_action(target.id, 'item.don')
  go_enable_action(target.id, 'item.doff')
end

ACTIONS['item.doff'] = function(target, initiator)
  go_attach(initiator.id, target.id)
  go_store(initiator.id, target.id)
  go_disable_action(target.id, 'item.doff')
  go_enable_action(target.id, 'item.don')
end
```

## Attaching / wearing objects

- A game object may be attached to another game object.
- Attached objects are visible outside of the parent object.
- Player (a game object too) may initiate actions from the game objects that are directly attached to it.
- Attached game object characteristics are added to the parent object characteristics.

## Item: Green Toad Spawner

```
{
  id = 'proto.item.spawner.toad.green';
  actions = {
    ['mob.spawn'] = {
      requires = { 'user.admin' };
      param = { proto_id = 'proto.mob.toad.green' };
    };
  };
};
```

## Action permissions

- An action is available only if `grants` list of the player contains all ids from `requires` list of the action.
- Items, attached to the player add their `grants` to it.

## Demo

- Current version of the application can be found here geo.logiceditor.com.
- Sources are linked to at the same page.
- The very first client is always `curl`. You can try out the API using it by appending `/api/v1` to the URL of the application.

## How does the client work?

- Geolocation and googlemaps are initalized.
- Current geoposition is sent to the server.
- Server returns a list of visible objects with their available actions.
- Markers are placed on the map for each object, and object details are rendered below the map by the HTML layout.
- Client awaits action initiation or coordinate change.

NB:

- Toad names are generated with change.js using object uuid as a seed.

## HTML5 Geolocation

- `navigator.geolocation.watchPosition(callback, options)`.
- Use developer panel tab Sensors to debug geolocation in Chrome.
- Chrome disables geolocation for HTTP sites (except localhost). Use HTTPS (e.g. with a free Let’s Encrypt certificate).

## Google Maps

```
new google.maps.Map(assert(document.getElementById('map')), {
  center: new google.maps.LatLng(pos.lat, pos.lon),
  zoom: 18,
  mapTypeId: google.maps.MapTypeId.ROADMAP,
  disableDefaultUI: true,
  disableDoubleClickZoom: true,
  draggable: false,
  scrollwheel: false,
  styles: [ { featureType: "poi",
    stylers: [ { visibility: "off" } ]
  } ]
});
```

## Problems

- Noisy geoposition data.
- Extremely low precision of the geolocation in the buildings.
- ...

## ~~Problems~~ Technical limitations

- Noisy geoposition data.
- Extremely low precision of the geolocation in the buildings.
- ...

There are no problems. There are technical limitations you have to consider while designing gameplay. Some of the limitations may be resolved technologically. If you need to spend time of it — is one of the questions for the preproduction stage.

## Missing mechanics in the current implementation

- Event system — the largest.
- Lots of small functions, like pedometer.

## A Way to Release

- ~~Throw away the code and write it from scratch.~~
- Create a new project and thoughtfully and manually move good parts of the prototype there.
- Rewrite from scratch bad parts.

## Questions?

@agladysh

ag@logiceditor.com

meetup.com/Lua-in-Moscow

Come to our Lua conference in March 5th in Moscow!