Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Bachelor's Thesis

GPS game for mobile framework Locify

David Vávra

Supervisor: Ing. Míkovec Zdeněk, Ph.D.

Study Programme: Software Technology and Management

Field of Study: Software Engineering

June 10, 2009

Acknowledgements

I would like to thank a lot following people:

- My colleague Ing. Jiří Mlavec for his hard work on Locify and helping me with implementation in Locify
- My friend Veronika Dvořáková, who created a wonderful logo for the game
- My supervisor Ing. Míkovec Zdeněk, Ph.D. for support, valuable advises and help with organizing tests
- All students of the subject Y36TUR who helped me to test the game in real conditions

Declaration

In Prague on June 10, 2009 ....

Abstract

In this thesis I describe a creation of GPS Game called Catch&Run. It is a multiplayer game running on mobile devices connected to the internet. Players can see each other on the map and when they get close enough, the chase starts. They are rewarded for their performance by virtual money. This game is an example implementation of service for Locify framework, which is also described in this thesis.

Abstrakt

V této práci popisuji tvorbu GPS hry jménem Catch&Run. Je to hra pro více hráčů běžící na mobilních zařízeních připojených k internetu. Hráči se vidí na mapě a když se k sobě přiblíží, začnou se navzájem chytat. Podle jejich výkonu jsou odměněni virtuálními penězi. Tato hra je ukázkovou implementací služby pro framework Locify, který je v této práci rovněž popsán.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goal . . . 1

2 Analysis **3**
2.1 GPS games . . . 3
2.2 My idea . . . 3
2.3 Catch&Run . . . 3
2.4 Functional requirements . . . 4
2.4.1 Web part . . . 4
2.4.2 Mobile part . . . 4
2.5 Locify . . . 4
2.5.1 Locify from user's point of view . . . 5
2.5.2 Locify from developer's point of view . . . 5
2.5.3 Examples . . . 6
2.6 Developers's guide . . . 6
2.6.1 Prerequisites . . . 6
2.6.2 Welcome page . . . 7
2.6.3 XHTML possibilities and limitations . . . 7
2.6.3.1 Supported element types . . . 7
2.6.3.2 Unsupported element types . . . 7
2.6.4 Acquiring location . . . 7
2.6.5 Internal functions . . . 7
2.6.6 KML files . . . 8
2.6.7 Advanced functions . . . 8
2.6.8 Example service . . . 8
2.6.8.1 Welcome page . . . 8
2.6.8.2 Acquiring location, azimuth and distance . . . 9
2.6.8.3 Calculation of the new location . . . 9
2.6.8.4 Screenshots . . . 9
2.7 Usage of Locify for Catch&Run . . . 10
2.7.1 What can be used . . . 10
2.7.2 What is missing (in Locify 1.0) . . . 10
2.8 Technical requirements . . . 10
2.8.1 Server . . . 10
2.8.2 Client . . . 11

3 Design **13**
3.1 Use cases . . . 13
3.1.1 Web part . . . 13
3.1.2 Mobile part . . . 14
3.2 Chase progress . . . 15
3.3 Data model . . . 16

3.4 GUI design . . . 17
3.4.1 Limitations . . . 17
3.4.2 Result . . . 17

4 Implementation 19
4.1 Extensions to Locify framework . . . 19
4.1.1 Refreshing map repeatedly . . . 19
4.1.2 Sound and vibration . . . 19
4.1.3 Map text overlay . . . 20
4.2 Graphics . . . 20
4.3 Server-side part . . . 21
4.3.1 CakePHP . . . 21
4.3.1.1 Example . . . 22
4.3.2 PostGIS . . . 23
4.3.2.1 Examples of location-based queries . . . 23
4.4 Problems and their solutions . . . 24
4.4.1 PostGIS cannot be installed to older Unix distributions . . . 24
4.4.2 How to deal with multiple players sending their location? . . . 24
4.4.3 How to implement user disconnecting? . . . 25
4.4.4 How to debug game situations? . . . 25
4.4.5 How to create an area shape? . . . 25
4.4.6 Google Static Maps API has URL limit for displaying routes . . . 26
4.4.7 Areas are not visible in Google Maps API when zoomed out . . . 27

5 Testing 29
5.1 Preliminary testing . . . 29
5.2 Unit testing . . . 29
5.2.1 CakePHP Unit testing . . . 29
5.2.2 Usage in Catch&Run . . . 29
5.3 Usability testing of low-fidelity prototype . . . 30
5.3.1 Prototype features . . . 30
5.3.2 Prototype creation . . . 30
5.3.3 Results . . . 30
5.3.4 Changes after the test . . . 30
5.4 Testing of final prototype . . . 31
5.4.1 Organizing test . . . 31
5.4.2 Results . . . 31
5.4.3 Changes after the test . . . 31

6 Conclusion 33
6.1 Goal fulfilment . . . 33
6.2 Future work . . . 33
6.2.1 Performance . . . 33
6.2.2 Web . . . 33
6.2.3 Mobile application . . . 34
6.2.4 Game . . . 34

Bibliography 35

A Database in SQL 37

B Contents of attached CD 39

Chapter 1

Introduction

1.1 Motivation

Penetration of mobile phones in the population is enormous and unexpected. There are much more cell phones than PCs in the world. Currently more and more cell phones are capable connecting to the internet. Prices go down and network coverage goes wider. Phones are getting smarter and they are changing from the call-making tool to universal devices, gateways to the internet. Phones are equipped with more sensors then PCs do. One of this sensors is used for obtaining phone's location - GPS. GPS is becoming broadly available - for example Nokia is planning to have 50% GPS-enabled phones in 2010-2012[1]. These observations are the reason why I see a great opportunity here - cell phones, internet and location together can create very useful applications for everyday life.

These technologies can save lives, make money or make life easier. I want to demonstrate possibilities of these technologies in a application used just for entertainment. I will demonstrate a creation of a GPS Game. This game will be played in the real world - a city center, park or forest can become the new playground. My game will be unique by actual interaction between players. People will be catching each other and meet together after the game. I hope the game will bring players a sport activity, new social connections and lot of fun!

1.2 Goal

The goal of this thesis is a creation of GPS game based on Locify framework. There is a list of steps I want to follow:

- analyze a concept of GPS game
- describe framework used for creating location-enabled applications - Locify
- create a Developer's guide for Locify framework
- test early ideas and design with users
- implement a web part of GPS game
- implement a mobile part of GPS game based on Locify framework
- test final application with users
- propose changes to framework itself based on experience from implementation and testing

Chapter 2

Analysis

In this chapter I will propose rules of the GPS game and describe framework Locify. I have chosen a name ”Catch&Run” for the game.

2.1 GPS games

There are number of GPS games in the today’s world. The most popular GPS games are:

- **Geocaching**: Players seek ”caches” hidden all over the world and log their visits online[2].
- **Wherigo**: Players can create and play ”cartridges” for Windows Mobile. Every cartridge contains another adventure - series of task player should complete. Game itself is offline and then players share their experience online[3].
- **GPS Mission**: A game similar to Wherigo with easier creation of cartridges and availability for more platforms[4].
- **NavBall**: Experimental multiplayer GPS game where players move with a virtual ball trying to score a goal[5].

2.2 My idea

I have played some of these games and have found some limitations of them. They are (excluding NavBall) just singleplayer and playing application is available for very limited number of devices. In my GPS game, I want to use internet capabilities of today’s phones and and create a multiplayer game. Players will be able to actually meet in the game. This will bring them sport, adventure and also socializing. I will also target most of mobile platforms by using framework Locify (section 2.5).

2.3 Catch&Run

Catch&Run is a multiplayer GPS game. Basic principle is an interaction of the players in the real world - they are chasing each other. Game can be played anywhere: for example parks, city centres, forests, ... A player can have three roles in the game - *catcher*, *runner* and *idler*. Catcher’s goal is to catch runner. Runner’s goal is to escape. Idler is just broadcasting his location and can turn into catcher or runner anytime.

When a player logs into a game, becomes an idler. If two idlers are close enough (concrete numbers have to be determined during a testing), one of them becomes catcher and the other runner. This choice is random. In this moment begins one minigame called *chase*. Until the chase ends, these two players cannot play with other players but themselves.

One chase can end with three possible results:

- Catcher catches runner and wins. It means getting close to runner in matter of meters.

- Runner escapes and wins. It means that runner is escaping for some time or is distant from catcher distant enough.
- Catcher or runner stops broadcasting his location repeatedly for any reason. Nobody wins and the player who stopped broadcasting first is mildly persecuted.

Chases are happening inside bordered *areas*. Every player can create an area, but area is limited by maximum square meters. It is possible to be idler just inside some area. If catcher or runner leaves area during the chase, looses the chase.

Players are making virtual money while playing the game. Every idler is making small amount of money for the time of idling. Idler does not make money when alone in the area or close to another idler. When chase is over, winner takes part of looser's money. When a player stops broadcasting his location, looses a small amount of money. See Figure 3.3 for schematic explanation.

2.4 Functional requirements

The game will have two parts - web and mobile. Web pages will contain game management and statistics. Game itself will be played on mobile devices.

2.4.1 Web part

Web pages will contain:

- Information about the game
- Registration, login and personal profile management
- Map of all areas in the world and players inside them
- Area management system - users can create, edit and delete areas
- Statistics for the user, area and global statistics

Server will also manage game logic for mobile clients. See related usecase in Figure 3.1.

2.4.2 Mobile part

Mobile application will be able to:

- View areas around current location
- Join selected area and show area and users inside on a map
- Repeatedly send location of the user to the server and refresh location of other players
- Maintain sound, vibration and visual alert when chase is starting
- Navigate player to the other player using map or compass during a chase
- View information about chase result

See related usecase in Figure 3.2.

2.5 Locify

Locify is a framework which makes creating mobile location-based applications easy. It is a mobile application with ready-to-use location features for the user. It can be expanded by services from 3rd party developers. This reasons make Locify the best candidate for my GPS game.

Figure 2.1: Locify logo

2.5.1 Locify from user's point of view

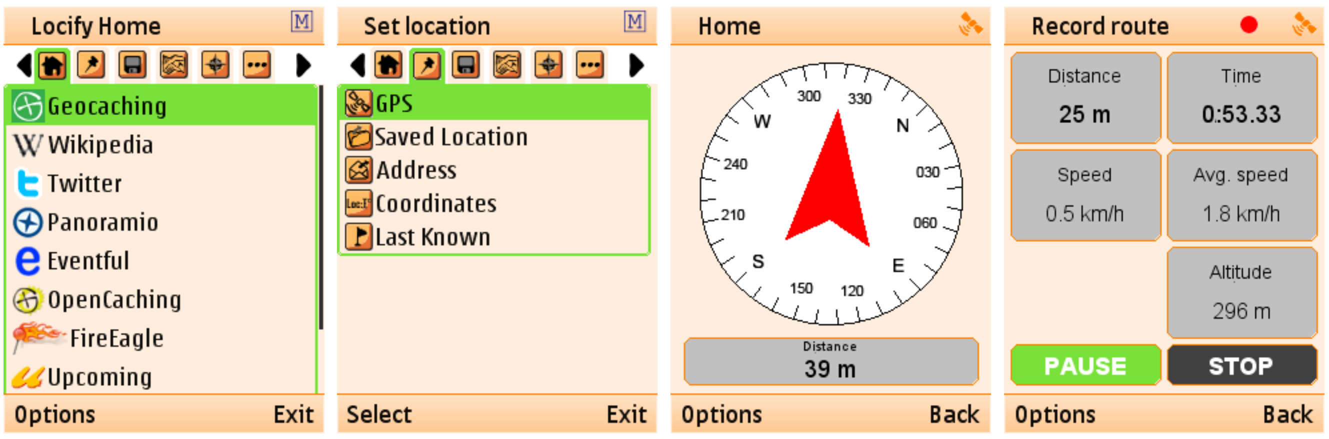
Locify is a mobile application written in mobile Java[6]. With this application, the user can:

- connect GPS (internal or Bluetooth) or input location manually
- create new places and routes
- browse on-line maps and view places and routes on them
- navigate to places and along routes
- use various online services which are providing interactive content

This application can be installed on various platforms which contain Java (MIDP 2.0, CLDC 1.1)[7]. Application was tested on this platforms:

- Nokia S40 and S60
- Sony Ericsson (without operation system or UIQ)
- Windows Mobile 6
- BlackBerry

Application is available in English and Czech language, it is free and open-source.



(a) Home screen with services (b) Possibilities for obtaining location (c) Navigation (d) Route recording

Figure 2.2: Selected application screens

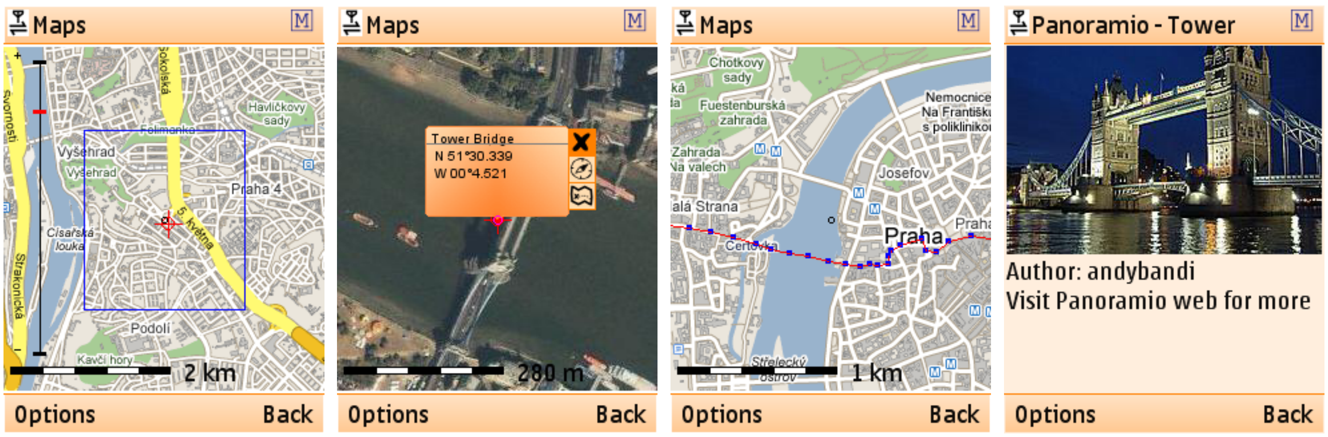
2.5.2 Locify from developer's point of view

For a developer Locify is acting like **XHTML browser**. It can view simple XHTML pages. These pages can contain some specific extensions, which allow developer to obtain user's location and use Locify internal functions like maps, navigation, offline data etc. It is very easy to write Locify service, because basic language is well-known XHTML.

2.5.3 Examples

For proper understanding it is better to list some examples of Locify services. These services are currently working and they are fully documented including XHTML source codes[8]:

- **Wikipedia** shows Wikipedia articles near user's location and opens selected article in external browser.
- **Geocaching** searches geocaches around user's location, views more information about a cache, shows it on map or navigates to it.
- **Twitter** enables user to update his Twitter status with his actual location.
- **Panoramio** searches nearby Panoramio photos and views them - see figure 2.3d.
- **Eventful** displays nearest events from Eventful database and shows more information about selected event.
- **FireEagle** sends actual location to the Fireeagle service, which shares the location to other web services.
- **AccuWeather** shows current weather in city nearest user's location. There is also 3-days weather forecast.
- **GeoMail** can send e-mail to anyone with link to online maps displaying user's location at the end.



(a) Map zooming (b) Place and satellite view (c) Route on map (d) Selected photo in Panoramio service

Figure 2.3: Working with map and Panoramio service

2.6 Developers's guide

After reading this guide, anyone should be able to create a simple Locify service. This guide is a simplified version of full developer documentation[8].

2.6.1 Prerequisites

It is important to understand that Locify works like XHTML browser. Then it is clear that developer needs the same tools as for regular web development. But just XHTML is not enough for dynamic services - in most cases some scripting language needs to be used on the server. It is up to the developer which scripting language will use - for example PHP, ASP.NET, J2EE, CGI, ... It is also useful to install phone emulator[9] for service testing.

2.6.2 Welcome page

Every service has to contain *welcome page*. This page describes service itself - it contains name, description, icon and URL of the service. When a user type URL of welcome page in the application (`Options->Add->Service By Link`) (s)he can install and use this service. The format of the welcome page is described on the web[10] or in the subsubsection 2.6.8.1.

2.6.3 XHTML possibilities and limitations

Every screen on mobile device is a XHTML page. Locify does not support full XHTML specification. It supports just the most important element types. But it is enough for the creating mobile web pages.

2.6.3.1 Supported element types

- Structure of the page: `<html>`, `<head>` and `<body>`. `<title>` defines screen title
- Forms (`<form>`, `<input>`(type=text, type=submit, type=hidden), `<button>`,`<textarea>`, `<select>`, `<option>`, `<label>`, `<fieldset>`)
- Tables (`<table>`, `<tr>`, `<td>`, `<th>`)
- Text formatting (`<strong>`, `<em>`, `<i>`, `<b>`)
- Other (`<img>`(in PNG format), `<a>`, `<p>`, `<br>`, `<div>`, `<span>`)

2.6.3.2 Unsupported element types

- `<input type="radio">` and `<input type="checkbox">` - developer can use `<select>` and `<select multiple>` instead
- Ordered lists (`<ol>`), Headings
- Media (`<object>`, `<embed>` etc.)
- CSS, JavaScript (`<style>`, `<script>`), Frames

2.6.4 Acquiring location

The strength of Locify is ability to react to user's location. But how to obtain it? It is simple: Every URL in Locify can contain *variables*. Using variables can provide lot of interesting information about client[11], most important variables are `$lat`(contains current latitude in decimal format) and `$lon`(contains current longitude in decimal format). So you can create for example this link: `http://myservice.com?lat=$lat&lon=$lon` and your page know user's location.

But this method has a disadvantage - user cannot change his location, connect GPS etc. before it is send. For this purposes the developer can create page for acquiring location. It is in fact XHTML form with special element types `<locify:where>` and `<locify:variables>`.
`<locify:where>` is a visual component - it shows currently set location and has a button to change it. With `<locify:variables>` is possible to set variables to the values of form fields. Developer can create form fields of type `hidden` and set variables `$lat` and `$lon` to them. You can find example in subsubsection 2.6.8.2.

2.6.5 Internal functions

Locify has interesting internal functions like maps, navigation, route recording etc. All internal functions are available to the developer via *internal URLs*. Internal URL always starts with `locify://` instead of common `http://`. These URLs can be used in links or redirections. The list of all internal URLs is on the web[12], here are some examples:

- `locify://maps?lat=50.1234&lon=14.1234&name=Prague` - Views place with coordinates N 50.1234 E 14.1234 and name Prague on map

- `locify://navigation?lat=50.1234&lon=14.1234&name=Prague` - Navigates to place with coordinates N 50.1234 E 14.1234 and name Prague
- `locify://external/http://www.fel.cvut.cz` - opens external browser and loads webpage `http://www.fel.cvut.cz`.
- `locify://mainScreen` - redirects user to the home screen

2.6.6 KML files

Locify stores all location data in KML format[13]. This format is used for example in a program Google Earth - so files from Google Earth are compatible in Locify and vice-versa. Locify can work just with subset of KML specification - it understands three types of files: *place*, *route* a *cloud of places*. A service can send this KML file to user - user can save it and then use offline. User can do actions based on a file type - all files can be viewed on the map, user can launch navigation to the place or along the route. The format of KML files is described on the web[14] or in the subsubsection 2.6.8.3.

2.6.7 Advanced functions

In the web development just XHTML is not enough. Locify also supports this features:

- **Authentication** - full support of HTTP Basic Auth[15] for user authentication
- **Cookies** - full support of Cookies standard. You can save for example session id to cookies.
- **Caching** - all pages are cached by default - if a developer do not want the page to be cached, a HTTP header `Pragma:no-cache`could be send.

2.6.8 Example service

Consider this: In some area there little papers with azimuth and distance on it. If player finds one of these papers, azimuth and distance can guide him to the next paper. This is repeated until players finds the final paper and wins. This game can be played with busola or more comfortable - with mobile phone, GPS and Locify service. We will now demonstrate how to create this kind of service.

2.6.8.1 Welcome page

Developer uploads this static page to his web. User inputs the URL of this page inside Locify (`Options->Add->Service By Link`) and installs this service:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1
    /DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:locify="http://client.locify.com/
    locify_ns/1.0" xml:lang="en" lang="en">
  <head>
    <title>GeoHunt</title>
  </head>
  <body class="serviceInfo">
    <div>
      <p class="description">This service can guide users through number of papers
          with azimuth and distance to the next paper.</p>
      <img src="http://www.fel.cvut.cz/~vavrad1/geohunt/icon.png" class="icon" alt=""
          />
      <a href="http://www.fel.cvut.cz/~vavrad1/geehunt/acquire.html" class="
          firstScreen">Start</a>
    </div>
  </body>
</html>
```

2.6.8.2 Acquiring location, azimuth and distance

This static page need to be uploaded to the server with filename `acquire.html`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1
    /DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:locify="http://client.locify.com/
    locify_ns/1.0" xml:lang="en" lang="en">
  <head>
    <title>GeoHunt</title>
  </head>
  <body>
    <form action="http://www.fel.cvut.cz/~vavrad1/geohunt/process.php" method="post">
      <locify:variables>
        <locify:element name="latitude" value="$lat" />
        <locify:element name="longitude" value="$lon" />
      </locify:variables>
      <fieldset>
        <locify:where />
        <label>Azimuth: <input type="text" name="azimuth" /></label>
        <label>Distance: <input type="text" name="distance" /></label>
        <input type="hidden" name="latitude" value="" />
        <input type="hidden" name="longitude" value="" />
        <input type="submit" value="Count!" />
      </fieldset>
    </form>
  </body>
</html>
```

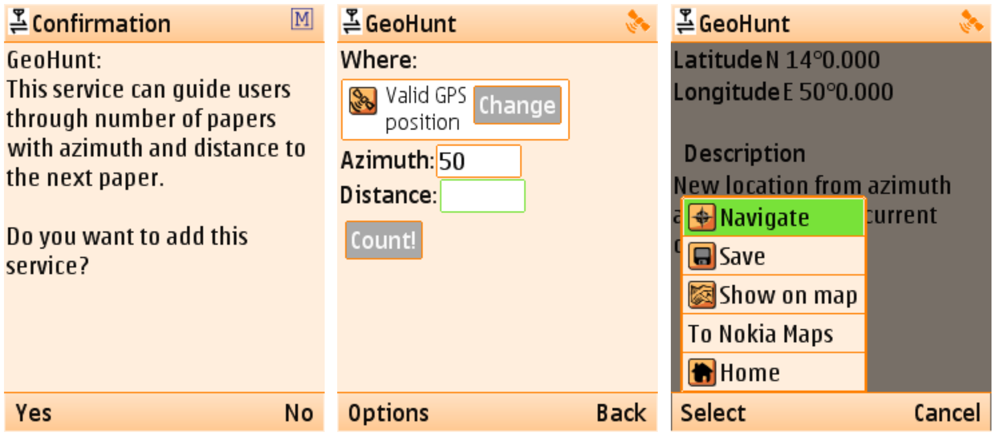
2.6.8.3 Calculation of the new location

There is a need of some server-side script to calculate the new location. Here is implementation in PHP. Script computes new coordinates with algorithm named Vincenty's direct formulae[16] and returns new place in KML format:

```
<?php
$coordinates = $_POST["latitude"].",".$_POST["longitude"];
$azimuth = $_POST["azimuth"];
$distance = $_POST["distance"];
$newCoords = directVincenty($coordinates, $azimuth, $distance);
echo '
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Placemark>
    <name>GeoHunt</name>
    <description>New location from azimuth and distance from current coordinates</
       description>
    <Point>
      <coordinates>'.$newCoords.'</coordinates>
    </Point>
  </Placemark>
</kml>
';
?>
```

2.6.8.4 Screenshots

And that is everything! We have now working Locify service. Screenshots from this service are shown in Figure 2.4.


Confirmation

GeoHunt:
This service can guide users through number of papers with azimuth and distance to the next paper.

Do you want to add this service?

Yes No

GeoHunt

Where:
Valid GPS position Change
Azimuth: 50
Distance:
Count!

Options Back

GeoHunt

Latitude N 14°0.000
Longitude E 50°0.000

Description
New location from azimuth

(a) Installation of service (b) Acquiring location, azimuth and distance (c) Incoming KML file

Figure 2.4: Screenshots from example service GeoHunt

2.7 Usage of Locify for Catch&Run

Locify will simplify creation of the mobile application a lot. But still are required some modifications to the framework itself to meet Catch&Run specification.

2.7.1 What can be used

- GPS connection and obtaining location
- List graphical component for nearest areas
- authentication
- viewing KML type *place cloud* for players in the area
- downloading, zooming, moving and navigation on the map
- XHTML pages for game information

2.7.2 What is missing (in Locify 1.0)

- Repeated online refresh of the map
- Ability to play sounds and vibrate

These missing features will have to be added to Locify framework.

2.8 Technical requirements

For successfull implementation of both web and mobile part of GPS game, these technical requirements has to be fulfilled:

2.8.1 Server

Server should meet this requirements:

- Unix-based operating system with SSH access
- web server Apache with PHP module
- PostgreSQL database
- addon to PostgreSQL - PostGIS[17]
- web database management system

2.8.2 Client

Client application will be build on Locify framework. User should have:

- mobile device compatible with Locify (section 2.5)
- internal GPS or Bluetooth GPS module
- data plan from network carrier

Games can be played only in areas with good GSM and GPS coverage.

Chapter 3

Design

In this chapter I will present Catch&Run functionality using UML diagrams and design system architecture. I will also design a prototype of user interface.

3.1 Use cases

Use cases shows functional requirments for web and mobile part of application.

3.1.1 Web part

UML diagram depicted in Figure 3.1 shows use cases for the user on the web part of application. User access this interface using desktop browser.

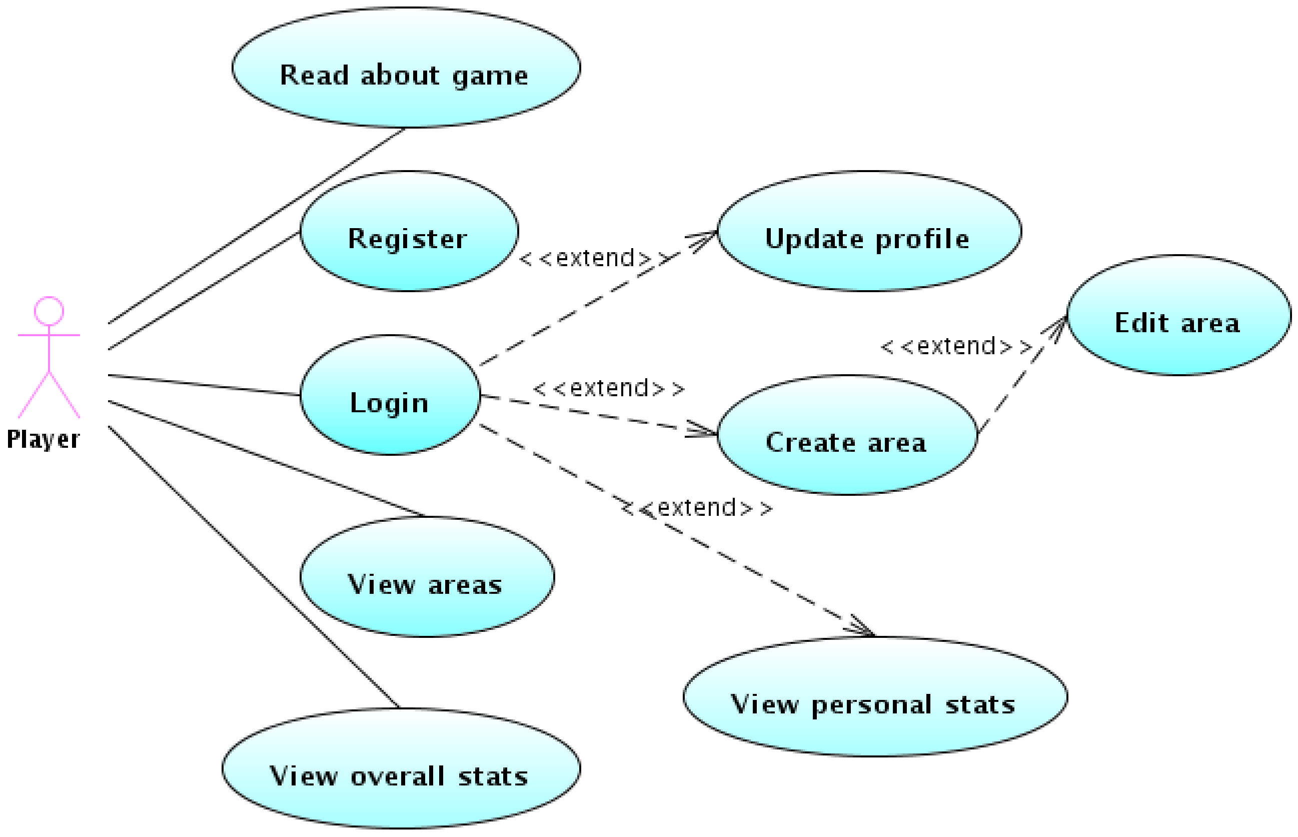


Figure 3.1: Web part usecase

3.1.2 Mobile part

UML diagram depicted in Figure 3.2 shows use cases for the user on the mobile part of application (using mobile phone). Proximity watcher is not a real person, it is a system watching proximity of users in the area.

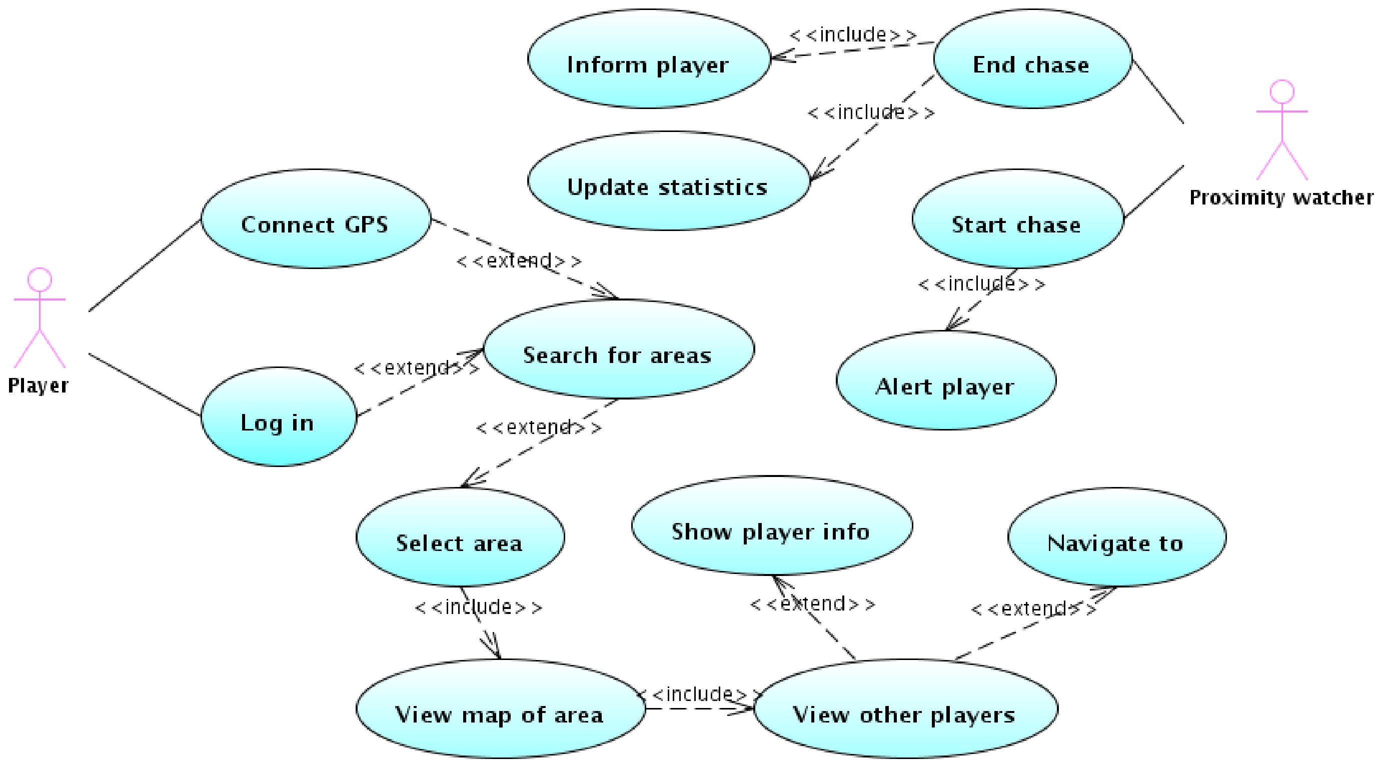


Figure 3.2: Mobile part usecase

3.2 Chase progress

UML state diagram depicted in Figure 3.3 shows progress of one individual chase. It shows three roles of the user - Catcher, Runner and Idler and three endings of the chase - winning, loosing and connection lost. It shows also situations where money are made.

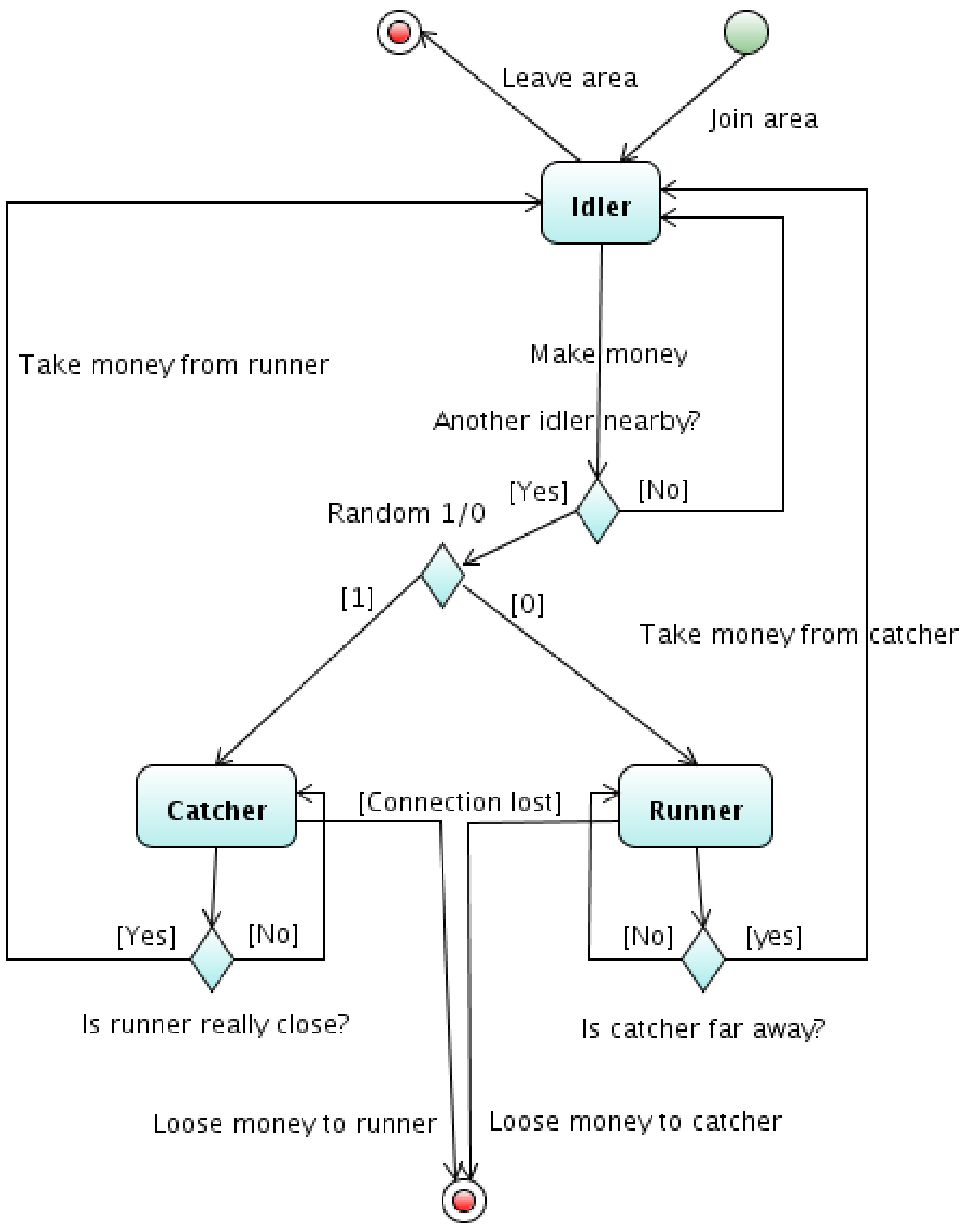


Figure 3.3: Chase state diagram

3.3 Data model

UML class diagram shown in Figure 3.4 models data in proposed system. It defines basic entities, their attributes and relations. This model will be transfered into relational database - entities will become tables in the database and attributes will become columns. There are just 0 to N typed relations, which will be transfered into foreign keys in database.

Implementation of data model consisting of SQL queries is shown in Appendix A.

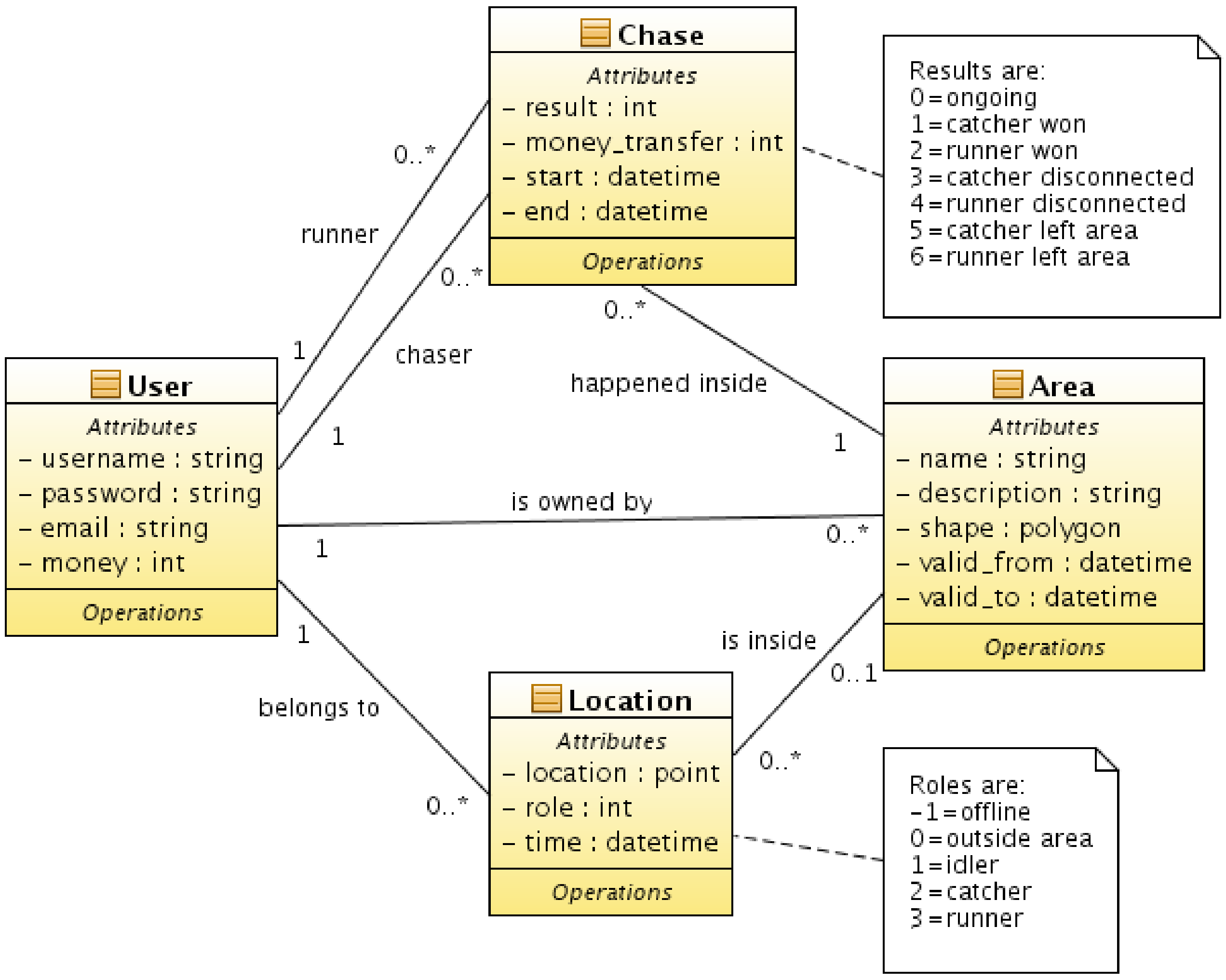


Figure 3.4: Data model diagram

3.4 GUI design

I did not have a free hand in GUI design because of Locify framework. But I was able to prepare a working applet for purposes of GUI design.

3.4.1 Limitations

In designing a GUI, I am limited by capabilities of Locify framework. On the other hand, GUI in Locify is easy to write (it is just XHTML), so the framework can be used for quick prototyping. I can use following GUI elements in Locify:

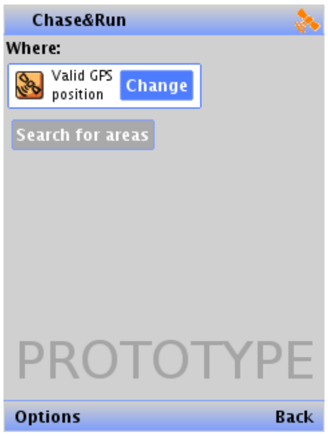
- element type `<locify:where>` for acquiring location (subsection 2.6.4)
- buttons
- list
- map
- icons of points and routes on map
- info bubble for a point on map
- map navigation visible as straight dotted line from location to point
- compass used for compass navigation
- other supported XHTML elements (subsection 2.6.3)

3.4.2 Result

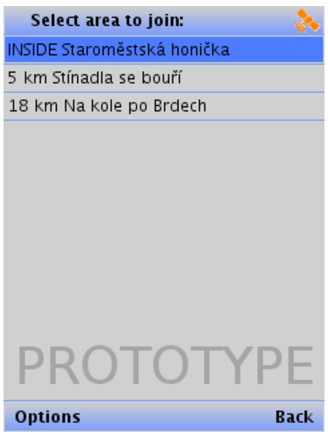
For purposes of the GUI design I have prepared a working prototype of Catch&Run application. Read more about it in section 5.3.

Figure 3.5 shows prototype screens which are visible after start of Catch&Run service in Locify. In these pictures the player is in the role *idler*, exploring his neighbourhood.

Figure 3.6 shows screens which are visible when a chase starts, player becomes catcher or runner and starts chasing or running. Player can use map or compass navigation to help him.

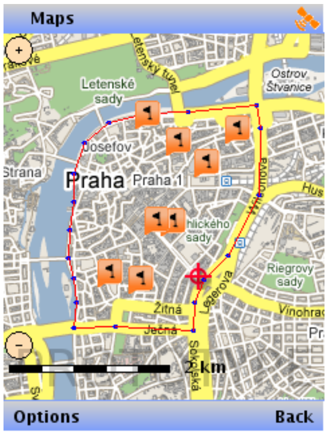


(a) Acquiring location

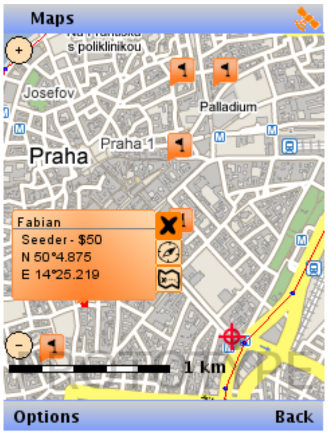

Select area to join:

INSIDE Staroměstská honička
5 km Stínadla se bouří
18 km Na kole po Brdech

(b) List of available areas

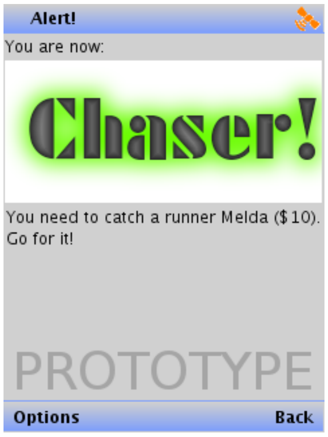


(c) Area overview


Fabian
Seeder - $50
N 50°4.875
E 14°25.219

(d) Player detail

Figure 3.5: Prototype screens

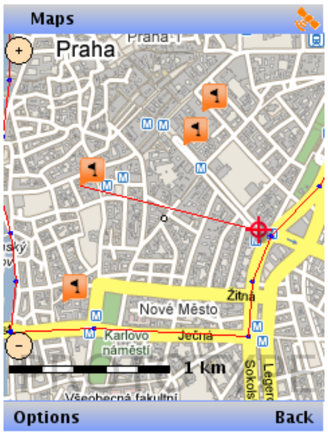

Alert!

You are now:

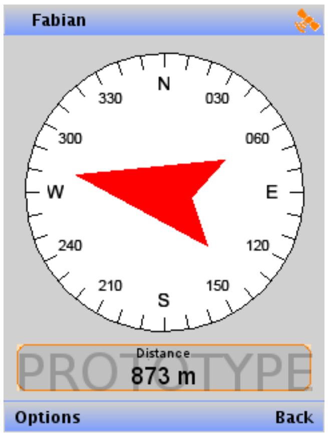
Chaser!

You need to catch a runner Melda ($10). Go for it!

(a) Chase started, player became idler



(b) Map navigation



(c) Compass navigation

Figure 3.6: Prototype screens

Chapter 4

Implementation

In this chapter I will cover implementation on both mobile and web part. I will describe technologies I have used and what problems I encountered.

4.1 Extensions to Locify framework

Locify framework did not have all required features when I started to work on Catch&Run. But Locify is an open-source project[18], so I could implement several improvements.

4.1.1 Refreshing map repeatedly

I needed the map to refresh automatically in given interval. I have been searching for the standard solution and find format **NetworkLink** described in KML specification[13]. NetworkLink is a KML file which is used mostly for dynamic loading of other KML files. NetworkLink specifies URL of the dynamic KML file and frequency of the repetition.

Implementation of NetworkLink itself was not so hard. It is just another KML file. But Locify was not prepared for updating some places or routes. I solved this issue by replacing KMLs on the map with same name. But what if user has navigation running, detail opened or map navigation running? I had to implement support for IDs of places inside KML. Then remember currently selected ID (player on the map), replace whole KML and update map navigation, compass navigation or opened detail according to the ID.

4.1.2 Sound and vibration

A player does not have to watch the screen while idling. There has to be a significant alert about start of the chase. I propose to make a sound alert, vibration alert and visual alert (display will blink). Sound playback can be done using JSR 135[19], vibration and blinking is part of MIDP 2.0 specification [20].

I have proposed following extensions to Locify XHTML browser:

```
<embed src="http://path/to/a/sound.wav" />
```

Element type *embed* works in the same way in desktop browser - it downloads a sound and plays it. I have implemented a support for this element type into Locify framework. It has a limitation of playing sounds only in WAV format - because this sound format can be played on any mobile device. There is also a default caching of sounds - a sound is downloaded only for the first time, for the second time it is played from cache.

There is no suitable element type for vibrations and blinking in XHTML. So I extended Locify namespace for this two element types: `<locify:vibrate>` and `<locify:blink>`. Example usage:

```
<locify:vibrate duration="800" />
<locify:blink duration="800" />
```

If this element type is present on the page, vibration or blinking is performed for a time specified in `duration` attribute in milliseconds.

4.1.3 Map text overlay

I did not plan this feature, but it became necessary after first prototype testing (section 5.3). User should be informed about his role and amount of money while playing. I was looking again to KML specification[13] and found element type `<screenOverlay>`. This element type is used for image or text map overlay. I have implemented just the text overlay, which is perfectly enough for me.
For example this code snippet:

```
<screenOverlay>
  <description>
    Idling... ($130)
  </description>
</screenOverlay>
```

will look like this:

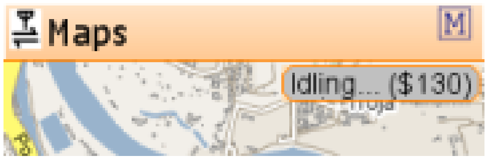


Figure 4.1: Part of screenshot displaying custom text screen overlay

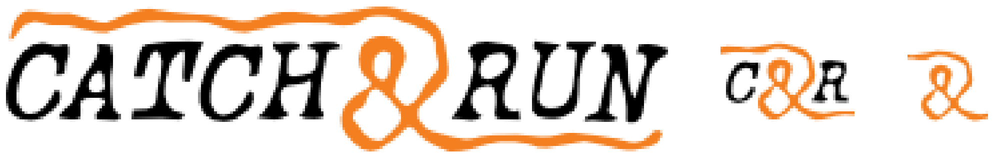
4.2 Graphics

I had to create some representative graphics for the game itself, the web and mobile application.

Logo shown in Figure 4.2 was created by Veronika Dvořáková[21]. Full logo is used on web, small logo is used as service icon in Locify and favicon is used also on web.

Player icons shown in Figure 4.3 are used in mobile application as map icons and on the web for explanation the rules. Orange icons are normal icons visible on map and red variant is used when for user in chase with current user.

Figure 4.4 shows a website layout. It is a screenshot from webpage showing available areas.



(a) Full logo (b) Small logo (c) Favicon

Figure 4.2: Logo

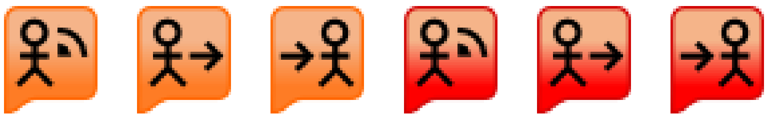
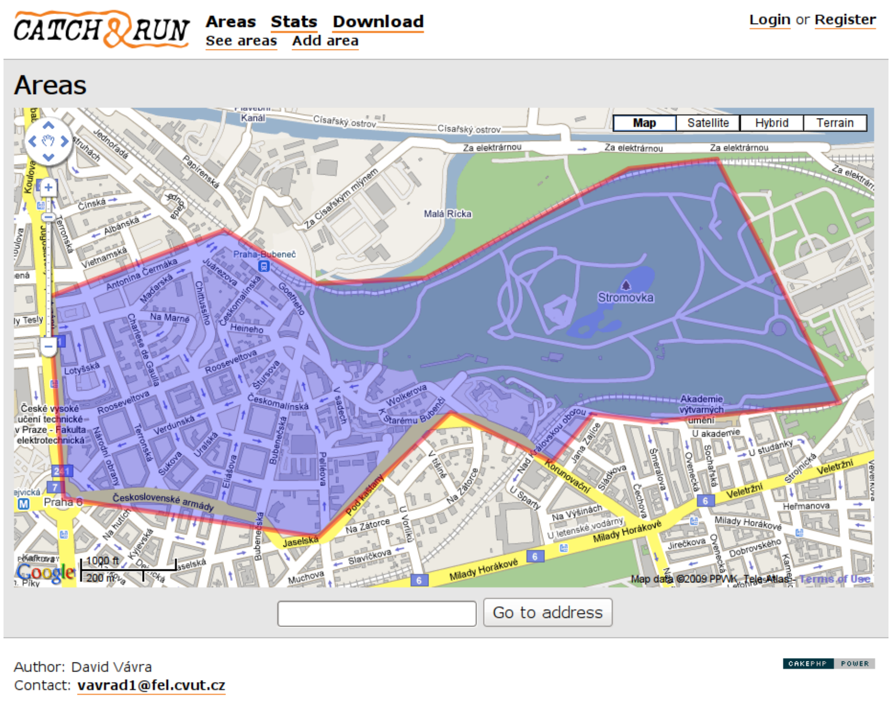

(a) Idler (b) Catcher (c) Runner (d) Idler (e) Catcher (f) Runner

Figure 4.3: Player role icons



Author: David Vávra
Contact: vavrad1@fel.cvut.cz

Figure 4.4: Website layout

4.3 Server-side part

Due to my previous experience I wanted to use language PHP for server-side part of application. I want the code to be clean and extendable, so I used a PHP framework CakePHP[22]. For database storage I needed PostgreSQL, because it supports great extension PostGIS[17] for storing location data. For presenting location data on the web I have used Google Maps API[23]. Server-side part is running at `http://edux2.felk.cvut.cz/car`.

4.3.1 CakePHP

CakePHP[22] is a great framework for PHP. It has same principles as well known framework Ruby On Rails, but it is still PHP, so it is much easier to learn and there are no problems with hosting. Main features of this framework are:

- Model-View-Controller pattern
- Strict naming conventions for everything. Code is clean and after that lot of things work automatically
- Command-line tool `cake` for generating models, views and controllers
- Built-in support for layouts, localization, authentication, pagination, AJAX, ...
- Helpers for building views like FormHelper or HtmlHelper
- Database abstraction and automated building of SQL queries

And that is just the most important features. CakePHP is simple to learn and really powerful. It has also strong community - when I had problems, all I need to do was to ask Google and some CakePHP forum came up. I would really recommend this framework for every PHP developer.

4.3.1.1 Example

I will present an example of registration, login and logout in CakePHP. We have database table called `users` and fields `username`, `password` and `email`. Then we need to create three files - model, controller and view.

First file described here is the **model** named `models/User.php`. In this file we define validation rules: username can be only alphanumeric, has to be unique, email has to be valid, password has to be minimum 5 characters long and password and confirm password has to match:

```php
<?php
class User extends AppModel {
    var $name = 'User';
    var $validate = array(
        'username' => array(
            'alphaNumeric' => array(
                'rule' => 'alphaNumeric',
                'required' => true,
                'allowEmpty' => false,
                'message' => 'Alphabets and numbers only'
            ),
            'unique' => array(
                'rule' => 'isUnique',
                'message' => 'This username has already been taken.'
            )
        ),
        'password' => array(
            'rule' => array('minLength', '5'),
            'required' => true,
            'allowEmpty' => false,
            'message' => 'Mimimum 5 characters long'
        ),
        'email' => array (
            'rule' => 'email',
            'required' => true,
            'allowEmpty' => false,
            'message' => 'E-mail is not valid'
        ),
        'confirm_password' => array (
            'rule' => array('confirmPassword'),
            'message' => 'Passwords do not match'
        )
    );
    function confirmPassword($data) {
                $valid = false;
                if ($this->data['User']['password'] == Security::hash(Configure::read
                    ('Security.salt') . $data['confirm_password'])) {
                        $valid = true;
                }
                return $valid;
        }

}
?>
```

Second file we need to create is a **controller** named `controllers/users_controller.php`. We need to define three methods for login, logout and register:

```php
<?php
class UsersController extends AppController {
    var $name = 'Users';
    function login() {
    }
    function logout() {
        $this->redirect($this->Auth->logout());
    }
    function register() {
        if ($this->data) {
            $this->User->create();
            if ($this->User->save($this->data))
            {
                $this->flash("Your account has been created", "/");
            }
        }
    }
}
?>
```

Last file we need to create is a **view** named `views/register.ctp`. It renders HTML form for registration:

```php
<?php
    echo '<h1>Register</h1>';
    echo $form->create('User');
    echo $form->input('username');
    echo $form->input('password');
    echo $form->input('confirm_password');
    echo $form->input('email');
    echo $form->end('Register');
?>
```

And that is all. Now these URLs works: `/users/register`, `/users/login` and `/users/logout`. Registration form corrects user according to the validation rules and users can login or logout. Result can be seen at `http://edux2.felk.cvut.cz/car/users/register`.

4.3.2 PostGIS

PostGIS[17] is an extension to database PostgreSQL. It helps developer to work with location-based data types. In PostgreSQL database table, all PostGIS data have type `geometry`. But there are several geometries PostGIS can work with:

- POINT (single point in the world)
- LINESTRING (route created from multiple points)
- POLYGON (closed area)
- MULTIPOINT (Several points)
- MULTILINESTRING (Several routes)
- MULTIPOLYGON (Several areas)
- GEOMETRYCOLLECTION (collection of geometries of different type)

Developer can work with this data using PostGIS SQL functions. There are functions for intersecting geometries, merging geometries, creating geometries, transforming them into different coordinate format, ...

4.3.2.1 Examples of location-based queries

- Is location N50, E14 inside area with ID 5?

```sql
SELECT ST_within(GeomFromText('POINT(14 50)',4326), shape) AS within FROM areas
    WHERE id='5';
```

- Return all users who are in distance 100-200 m from point N50, E14 inside area with ID 5:

```
SELECT
DISTINCT ON (user_id) user_id
FROM locations
WHERE area_id='5'
AND ST_distance(ST_transform(location,utmzone(location)), ST_transform(
    GeomFromText('POINT(14 50)',4326),utmzone(location))) > 100
AND ST_distance(ST_transform(location,utmzone(location)), ST_transform(
    GeomFromText('POINT(14 50)',4326),utmzone(location))) < 200
ORDER BY user_id, time DESC;
```

- Return user who is a catcher of runner ID 1 in area ID 5, distance towards him from point N50 E14 and chase duration:

```
SELECT * FROM (
SELECT
DISTINCT ON (l.user_id) l.user_id
ST_distance(ST_transform(location,utmzone(location)), ST_transform(GeomFromText(
    'POINT(14 50)',4326),utmzone(location))) AS dist,
EXTRACT(MINUTES FROM AGE(NOW(),c.start)) AS duration,
c.start
FROM locations AS l
JOIN users AS u
ON u.id = l.user_id
JOIN chases AS c
ON u.id = c.catcher
WHERE l.area_id = '5'
AND c.runner = '1'
ORDER BY l.user_id, l.time DESC, c.start DESC) AS v ORDER BY v.start DESC;
```

4.4 Problems and their solutions

I have encountered several unexpected problems during implementation and here I describe their solutions.

4.4.1 PostGIS cannot be installed to older Unix distributions

When I was installing PostGIS to school Debian server 'faraon', I came across following problem: Newest PostGIS requires PostgreSQL 7.2 or higher. This version of PostgreSQL needs library `libc6`. This library is a part of Linux kernel 2.6 and higher. There was kernel 2.4.2 on the school server.

Possible solution would be to upgrade Linux kernel, which is not an easy task. Better solution would be reinstalling whole server. I was not allowed to do that so I had to switch to newer school server 'edux2'.

4.4.2 How to deal with multiple players sending their location?

In Catch&Run, there are several players in the same time in one area broadcasting their location. Server receives these HTTP requests in a queue. How to implement a good application logic?

All location updates are handled by one method in location controller. I save every location of all users in the game into database. First I query database about last location and ask for last player role. Then i behave according to users' role:

- Offline - if player is inside area, becomes *idler*, otherwise is *outside area*
- Outside area - if player is inside area, becomes *idler*
- Idler - if there is another nearby idler, chase starts. If not, i have to ask if chase has not already been started by some other player. If not, idler will make money.

- Catcher - if user is outside area, chase ends. Then I ask database for runner. If runner is close enough, chase ends and user becomes *idler*. If not, I have to ask if chase has not been ended by runner.
- Runner - if user is outside area, chase ends. Then I ask database for catcher. If catcher is far enough, chase ends and user becomes *idler*. If not, I have to ask if chase has not been ended by catcher.

4.4.3 How to implement user disconnecting?

Inactive users are disconnected in Catch&Run and they loose some money if they are in the chase. I need to check for disconnected users periodically. How to do it?

First I thought that query for disconnected users will be performed with every location update on the server. But this would be really time-consuming operation. I used Unix background process `cron`. I put this line into the file `/etc/crontab` on server:

```
* * * * *  lynx -dump http://edux2.felk.cvut.cz/car/chases/check &> /dev/null
```

It will call my script every minute using command-line browser `lynx`.

4.4.4 How to debug game situations?

Game cannot start when there is just one player in the area. How to debug game situations when I need at least two players broadcasting their location at the same time and closing to each other?

This was quite a hard problem at first. I had to test and debug several game situations. First thing I needed was a movement. Locify has built-in simulator of GPS. It can read NMEA sentences[24] from file and move with the player. GPS data was taken in Dejvice, Prague. I have created testing area there.

First I set up one player's location statically, I turned off users' disconnecting and launched Locify in WTK emulator[9]. I was observing application behavior in emulator and when something was not right, I have copied last location update URL in a regular web browser. With the help of CakePHP's `debug()` function, I could see content of variables and I could fix the problem. CakePHP also printed out all performed SQL queries and errors in them.

But this was not enough. I needed two simultaneous devices for testing more complex situations. I have created an applet version of Locify and embedded it into webpage using Microemulator[25]. The other player was still in WTK emulator.

Then I needed to test application behavior on the real device. Some issues occurs only on real devices. Now I could not use localhost server. I uploaded server application into production server and launched WTK emulator, applet and application on the real phone at the same time.

But this still was not enough. I had to test application behavior while actually moving with the cellphone. I have turned on users's disconnecting and create virtual player using `cron`:

```
* * * * * lynx -dump http://edux2.felk.cvut.cz/car/locations/update/5?lat=50.104158\&
    lon=14.401510\&test=1 &> /dev/null
```

There were more ways to debug: I have tested some situations by changing game's rules on the fly. I have also setup log files for logging all cron's actions. I could also use CakePHP's error logs. But I have to admit - this debugging phase was the hardest part in the whole project.

4.4.5 How to create an area shape?

This concerns the web part of application - users should be able to create their own areas. How to do this in user-friendly way?

I used a Google Maps Javascript API[23] for convenient area creation right on a world map. I have modified one example[26] of creation a polygon. Result can be seen in Figure 4.5.

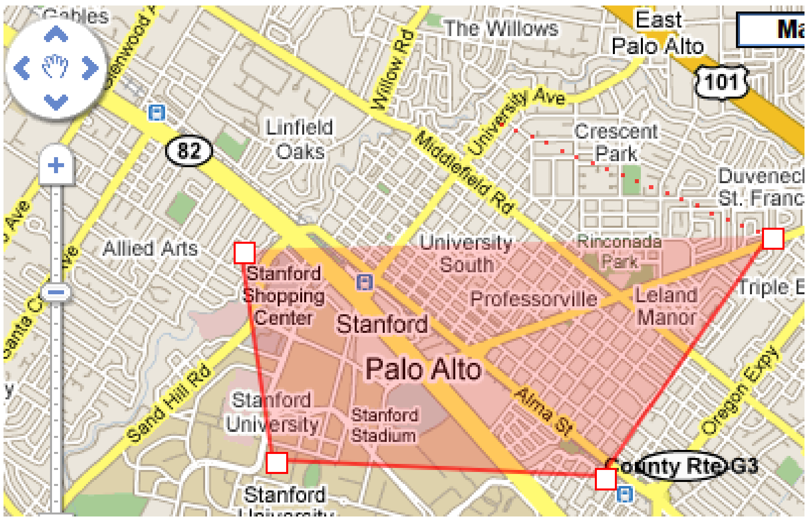


Figure 4.5: Creation of area online

4.4.6 Google Static Maps API has URL limit for displaying routes

In the personal statistics on the web I have a list of all user's chases with map of the user and his opponent. It is possible because I save all locations into database. This approach has also its limits - table with locations will soon be enormous and server will not be able to process it. So I came with the solution - table locations will be (in the future) deleted each day and all player routes will be archived in map images from Google Static Maps API[27]. But in Static Maps API, all parameters are sent via HTTP GET. And HTTP URL has a limit about 2048 characters. Most of the routes were larger than that. What now?

First I reduced numbers in each coordinate to 7. It helped, but it was not enough. Hopefully PostGIS have function for reducing routes into less points. User will not see the difference and URL will have much less coordinates. Here is a SQL-snippet which solves that:

```sql
SELECT
ST_Collect(
ST_simplify(ST_MakeLine_garray(ARRAY(SELECT location FROM locations WHERE user_id = c
    .catcher AND time>=c.start AND time<=(c.end + interval '1 second'))),0.0001),
ST_simplify(ST_MakeLine_garray(ARRAY(SELECT location FROM locations WHERE user_id = c
    .runner AND time>=c.start AND time<=(c.end + interval '1 second'))),0.0001)
) AS progress
FROM chases AS c
```

Result can be seen in Figure 4.6.

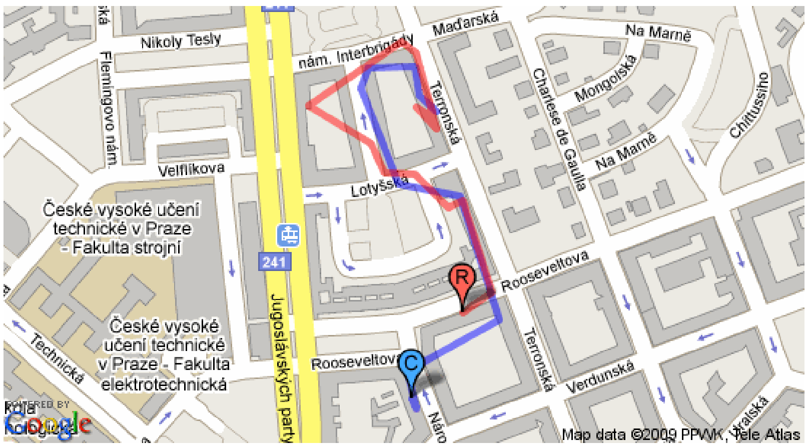


Figure 4.6: Chase progress on map

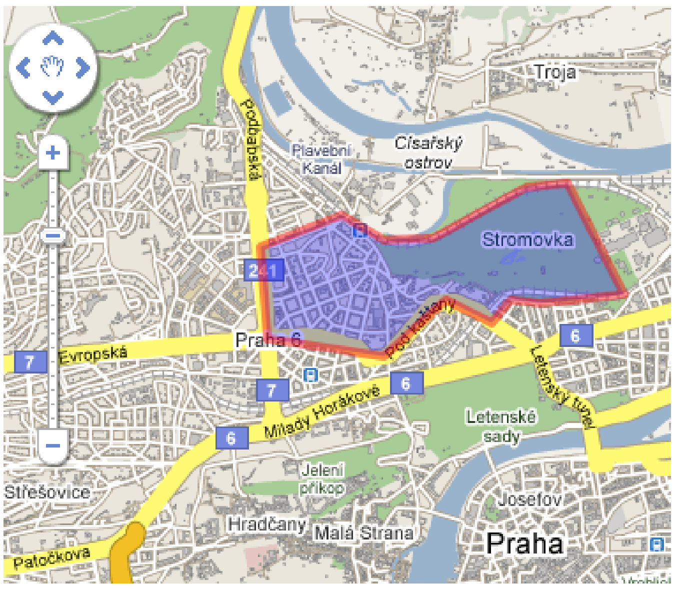
4.4.7 Areas are not visible in Google Maps API when zoomed out

I need to have a list of available areas all over the world. Most user-friendly way is to use Google Maps again. Displaying polygon on Google Map is easy but when I zoomed out, polygon was smaller, smaller and then dissapeared. What now?

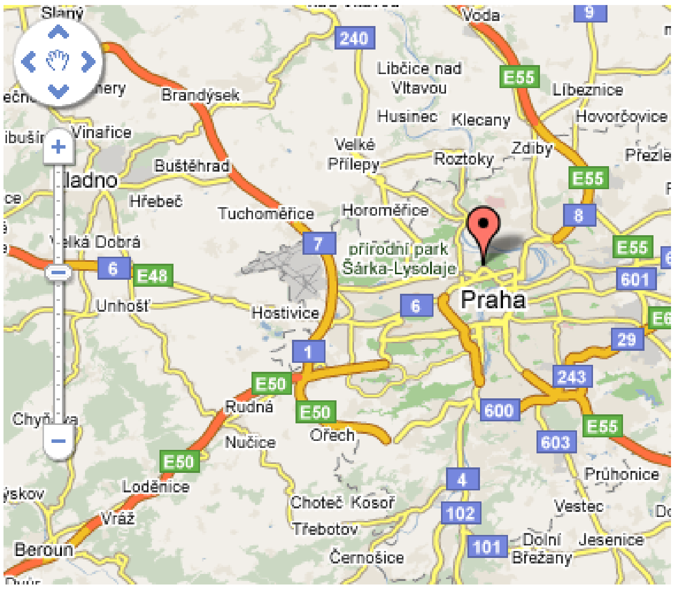
Solution was quite simple. I exchanged all polygons on Google Map with markers at some zoom level. Here is a Javascript snippet:

```
function zoomChanged(oldLevel, newLevel, start)
    {
        if (newLevel > 12) {
            //polygons
            if (markersVisible || start) {
                map.clearOverlays();
                for ( var i=0;i<polygons.length;i++ )
                {
                    map.addOverlay(polygons[i]);
                }
            }
            markersVisible = false;
        }
        else
        {
            //markers
            if (!markersVisible || start) {
                map.clearOverlays();
                for ( var i=0;i<markers.length;i++ )
                {
                    map.addOverlay(markers[i]);
                }
            }
            markersVisible = true;
        }
    }
```

Result can be seen in Figure 4.7.



(a) Zoomed in



(b) Zoomed out

Figure 4.7: Area on map

Chapter 5

Testing

I have been testing my application in all stages of project cycle.

5.1 Preliminary testing

I have began testing in the first phase - when I was just defining the game concept. I have sent a first draft of the game rules to several friends and colleagues. I have received a valuable feedback. According to the feedback I changed following:

- When catcher cathes runner, catcher will take money from runner. Firstly I thought cacher will be rewarded and runner will not lose anything, but this would enable cheating from "friendly runners"
- Chases will take place just inside bordered areas. If anyone leaves area during chase, looses. This will remove some unexpected problems (outside area does not have to be network coverage) and create new strategies (runner in the corner).
- Actual money taken when the chase ends will take player's experience in mind (measured by money). This will prevent experienced players chasing newbies and vice versa. Most profitable will be to chase someone with similar experience.

5.2 Unit testing

I have also tried to test Catch&Run with CakePHP's unit tests.

5.2.1 CakePHP Unit testing

There is a great support for unit testing in CakePHP[22]. This include following features:

- unit tests for models, controllers and views
- automated generation of test skeleton classes using tool `cake bake`.
- fixtures - a way to create database tables and fill them with test data just for tests
- web interface for launching and evaluating tests
- various asserts for many situations

5.2.2 Usage in Catch&Run

I have been studying CakePHP's unit tests for a while and created a few tests for a few models and controllers. It works well for a simple tasks like login, registration etc. But what I need to test most - game logic - appeared to be very hard to test using unit tests. There has to be multiple users sending their location simultaneously and Cake's unit tests are not ready for that. I believe there will be a solution (because CakePHP is highly extendable), but it will take more time than actual testing described in subsection 4.4.4. So I have decided not to use unit tests for testing game logic.

5.3 Usability testing of low-fidelity prototype

I have created a working prototype of Catch&Run game and tested it remotely with students. Prototype is running as an applet in a web browser. Applet logs all actions of the user and saves it into database. Test also contains questions for the tester about usability of the application. Screenshots of this prototype can be found in section 3.4. Prototype is running at `http://edux2.felk.cvut/char/test`

5.3.1 Prototype features

- Launching of the service, choosing area nearby and login - with static data
- Connecting GPS and simulated movement in virtual area
- Google Map of the virtual area, zooming and panning is possible
- Virtual players in the area moving randomly
- Possibility of viewing player details in a bubble
- Simulated start of the chase (after fixed time)
- Possibility of map navigation or compass navigation to the target

5.3.2 Prototype creation

What I had to do to create the prototype:

- create a specific build of Locify - remove all references to JSR 75[28] and other tweaks
- create an applet from midlet using Microemulator[25] and put it into website
- create a NMEA[24] log for GPS simulator in Locify - I prepared route in Google Earth[29] and then converted into NMEA format using Route Converter[30]
- create a prototype service using static data for first screens
- create a PHP script which emulates a movement of virtual players and returning an output as KML with route (area boundary) and place cloud (players)
- create another Locify service for saving log into database when test is over

5.3.3 Results

Test was performed by **90 people**, mostly students of subject Y36TUR[31]. 59 of them completed all steps in the test, so success rate was **66%**. A problem reported most times was: It was not possible to click whole area of icon on map, just bottom left corner. About 20% of participants proposed better terminology for the game. Students were generally excited about game's concept.

5.3.4 Changes after the test

- It is now possible to click whole icon on map, not just bottom left corner
- Players have icons according to their role (Figure 4.3)
- Icon of an opponent in the chase has different color
- It is possible to start navigation directly from the chase announcement page
- Status line on the map about role and money of current player was added (subsection 4.1.3)
- Changes in game's terminology:
  - Chase&Run $\Rightarrow$ Catch&Run
  - Seeder $\Rightarrow$ Idler
  - Chaser $\Rightarrow$ Catcher
  - Hunt $\Rightarrow$ Chase

5.4 Testing of final prototype

Test in the real conditions was necessary to test the concept and programming of the game.

5.4.1 Organizing test

What I had to do to organize test:

- Create a virtual player in area so participants can test application before test (subsection 4.4.4)
- Find a suitable area for the game - area around park Stromovka in Prague won because of good GSM signal, wide streets (good GPS signal) and low traffic (security)
- Create a user-guide for installing Locify and Catch&Run service - available on attached CD (Appendix B)
- Lend some cell phones and GPS devices from CTU
- Organize an 'installation and troubleshooting' session for participants

5.4.2 Results

There were **two tests** - 12.5.2009 and 14.5.2009. Each test was lasting 2 hours. First test has **6 participants**, second test **7 participants**. All of them were able to play (few of them had to quit earlier due to technical problems). Most reported problem was: Application was falling down because of not enough memory. I have made a video clip from the testing. It is available on attached CD - see Appendix B.

5.4.3 Changes after the test

- During the first test server ran out of memory after one hour and half. I increased amount of memory for PHP from 16 MB to 64 MB. It helped.
- We have been working on memory issues on client side as well. We did some memory optimizations in Locify and created caching for maps - now map tiles are downloaded just once and then they are cached into filesystem, which reduces allocated memory and speeds user interface.
- Runner is informed about remaining time and distance in the status bar
- Catcher is informed about remainting distance in the status bar
- Distance to other players is visible also on map, not just in navigation
- Changes in rules:
  - Idler does not make money while is alone in the area
  - Idler does not make money while there is another idler very close (and chase cannot start)
  - Catcher wins in proximity less than 15 m
  - Runner wins in proximity more than 500 m or 10 minutes.

Chapter 6

Conclusion

The usability tests proved that the concept of GPS game Catch&Run is successfull. Framework Locify proved itself suitable for this kind of service. I hope Catch&Run has bright future ahead.

6.1 Goal fulfilment

Let's summarize steps I wanted to fulfil:

- I have analyzed concept of GPS Game (chapter 2, chapter 3)
- I have described a framework Locify (section 2.5)
- I have created a Developer's guide for Locify framework (section 2.6)
- I have tested early ideas and design with users using online prototype (section 5.3)
- I have implemented a web part of GPS game (section 4.3), but couple of features are missing for going public (subsection 6.2.2)
- I have implemented a mobile part of GPS game based on Locify framework (section 4.4), but there are still some client-side issues left (subsection 6.2.3)
- I have tested final application with users (section 5.4)
- I have proposed changes to framework itself and implemented them (section 4.1)

6.2 Future work

6.2.1 Performance

There is a lot of potential for performance improvement. Easiest way is to create spatial indexes on all location-based columns in database. PostGIS queries will be then much faster.

Currently all locations of all players are saved into database. This is unacceptable in long-term. Locations table should be erased every day, but before that chase routes will be generated using Google Static Maps API[27] and saved to disk. This way statistics will be archived and performance will stay on appropriate level.

Next problem is a hosting. Best way for this kind of service will be the use of hosting as IaaS (infrastructure as service). I have experience with IaaS provider Amazon Web Services[32]. When hosting with Amazon, I would pay only for traffic and actual usage of the site. Then it is very simple to clone virtual servers and scale whole application.

6.2.2 Web

Most of the future work has to be done on Catch&Run web. Web needs lot of polishing and new features before going public. Most important feature is an attendance planning for the games. Every area have date of availability so users will be able to tell if they are going to participate the game in that area or not.

There should be discussions about areas and about game itself, it is really important for growing of the community. I am also thinking about possibility of virtual treasures inside area. Creator of the area could place a treasure inside area, which will give players additional money. Game will be more attractive afterwards.

Areas should be 'permanent' and 'one-time-only'. In permanent areas, user logs in with his current amount of money. He can loose a lot, earn a lot. In the one-time-only areas, all users starts with 0 money when they log in. It will be more fair and used for propagation events for beginners.

6.2.3 Mobile application

There is lot of to be done in Locify as well. We have been doing research about memory consumption and found out that most of memory is consumed by our GUI framework J2ME Polish[33]. This framework has also other issues and we decided to move to different framework - LWUIT[34]. It will be a lot of work, but after that application will be smaller, faster and much nicer.

I am also thinking about developing a native application for Google Android. I will not create an application for other platforms like Apple iPhone, but I am thinking about publishing an open API to game data, so everyone can create alternative application for gameplay.

6.2.4 Game

I am thinking about another reward for users - experience. Users will be measured by money and experience. They can loose money, but they can not loose experience. Experience will be given for winning chases, not idling. Experience will be used for dividing money after chase. Most profitable will be to win with someone on the same level of experience. This will protect beginners.

Money could be used for bonuses inside the game. I am thinking about bonuses like:

- Temporal 'invisibility' against others
- Role switch during the chase
- Longer catching distance for catcher
- Shorter running distance for runner

There is also a place for commercial application (buying these bonuses for real money).

Bibliography

[1] Nokia sees half of cellphones with GPS in 2010-12.
http://www.reuters.com/article/technologyNews/idUSL1442615920080514.

[2] Geocaching - the most popular GPS game.
http://www.geocaching.com.

[3] Wherigo - GPS game from Groundspeak.
http://www.wherigo.com.

[4] GPS Mission - GPS game similar to Wherigo.
http://gpsmission.com/.

[5] NavBall - virtual GPS football.
http://navball.wordpress.com/.

[6] Locify - web of the project.
http://www.locify.com.

[7] List of devices supporting Locify.
http://www.locify.com/phones.

[8] Locify developers documentation.
http://www.locify.com/documentation.

[9] How to install and use emulator for testing Locify services.
http://www.locify.com/documentation/testing.

[10] Format of the welcome page.
http://www.locify.com/documentation/describe-service.

[11] List of variables which can be used in Locify.
http://www.locify.com/documentation/variables.

[12] List of internal URLs which can be used in Locify.
http://www.locify.com/documentation/internal-urls.

[13] KML format specification.
http://www.opengeospatial.org/standards/kml/.

[14] KML formáts usable in Locify.
http://www.locify.com/documentation/filesystem.

[15] Basic Access Authentication.
http://tools.ietf.org/html/rfc2617.

[16] Vincenty Direct Formulae.
http://www.movable-type.co.uk/scripts/latlong-vincenty-direct.html.

[17] PostGIS - addon to PostgreSQL database to help with spatial data.
http://postgis.refractions.net/.

[18] Locify as open-source - source code of Locify is available here.
http://code.google.com/p/locify.

[19] JSR 135 - Mobile Media API.
http://jcp.org/en/jsr/detail?id=135.

[20] How to make phone vibrate in J2ME.
http://www.java-tips.org/java-me-tips/midp/how-to-make-phone-vibrate-in-j2me.html.

[21] Veronika Dvořáková. Grafic works.
v.dvorka@email.cz.

[22] CakePHP - the rapid development PHP framework.
http://cakephp.org/.

[23] Google Maps Javascript API.
http://code.google.com/apis/maps/.

[24] NMEA 0183 - standard for formatting GPS data from GPS devices.
http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp.

[25] Microemulator - implementation of Java ME inside Java SE.
http://www.microemu.org/.

[26] Google Maps JavaScript API Example: Editable Polylines.
http://gmaps-samples.googlecode.com/svn/trunk/poly/mymapstoolbar.html.

[27] Google Maps Static API.
http://code.google.com/apis/maps/documentation/staticmaps/.

[28] JSR 75 - File connection API.
http://jcp.org/en/jsr/detail?id=75.

[29] Google Earth - excelent tool for creating KML files and more.
http://earth.google.com.

[30] Route converter - a tool to convert routes into various formats.
http://www.routeconverter.de.

[31] Y36TUR: Subject at CTU FEE, translation of name: User Interface Design.
http://www.feld.cvut.cz/education/bk/predmety/01/89/p18916.html.

[32] Amazon Web Services - IaaS service for hosting.
http://aws.amazon.com/.

[33] J2ME Polish - GUI framework used in Locify.
http://www.j2mepolish.org/cms/.

[34] LWUIT - Future GUI framework used in Locify.
https://lwuit.dev.java.net/.

Appendix A

Database in SQL

This SQL queries shows the structure of actual Catch&Run database. It is possible to create the database in PostgreSQL using these queries:

```sql
CREATE TABLE areas (
    id integer NOT NULL,
    name character varying(128),
    description text,
    user_id integer,
    valid_from timestamp with time zone,
    valid_to timestamp with time zone,
    shape geometry
);

CREATE TABLE chases (
    id bigint NOT NULL,
    catcher integer,
    runner integer,
    result smallint,
    money integer,
    area_id integer,
    start timestamp without time zone DEFAULT now(),
    "end" timestamp without time zone
);


CREATE TABLE locations (
    id bigint NOT NULL,
    role smallint,
    area_id integer,
    user_id integer,
    location geometry,
    "time" timestamp without time zone DEFAULT now()
);

CREATE TABLE users (
    id integer NOT NULL,
    username character varying(64),
    email character varying(64),
    password character varying(128),
    money integer DEFAULT 0,
    attending integer
);
```

Appendix B

Contents of attached CD

An attached CD has following structure:

```
|-- exe          Builds of Locify
|   |-- BlackBerry          BlackBerry version
|   |-- Generic          Version for all other phones
|   |-- Nokia 6230i          Version for Nokia 6230i
|   |-- Windows Mobile          Version for Windows Mobile
|   `-- Windows Mobile big screen          Version for WM with displays bigger than 600px
|-- html          Installation guide
|-- index.html          Project overview and links
|-- readme.txt          Short info about files
|-- src          Source code
|   |-- Catch&Run          PHP source code of Catch&Run
|   `-- Locify          Java source code of Locify
|-- text          Documentation
|   |-- latex          LaTeX sources
|   `-- vavrad1.pdf          PDF output
`-- video          Videos from testing
```