Crime statistics mobile application

Software development project

Jose Ricardo Buitron de la Vega - 595356

Supervisor: Rui Zhang
Applications

Crime prevention
The government and security institutions could use the information provided by this application. The government can geographically identify the places where social programs need to be implemented to reduce crime and violence. The police could identify the regions where more units and security is needed.

Crime avoidance
People could use this application to plan common activities in their daily life. For instance, when planning a trip they could choose the best places to visit based on the crime rate, and even the best routes to reach a specific place. Additionally, it could be used when looking for a new place to live as an extra indicator to make an appropriate decision.

Requirements

- Gather data that allows to outline the boundaries and data with crime statistics about these places.
- Link the gathered data and create a data structure that can be accessed from the android application.
- Display crime data and boundaries of the identified places in a mapping component taking into account performance and usability.
- Identify an appropriate user interface to interact with the crime statistics and the mapping component.
Data sources

Google maps
The first option to retrieve the geometric boundaries for each region, due to its popularity, was the Google Maps API. Unfortunately, even though they display the boundaries data when using their website, they don't provide it through their API. As a result, a different data source had to be found.

Overpass API
Other option was Overpass API. It is an API that can provide location information through a web service. It uses a query language called Overpass QL, that allows to create complex requests. This approach was sufficient to extract the geometric boundaries of the regions. However, the associated data did not have all the required information needed to identify the regions for posterior matching with the crime data and interaction with the user.

Data files
Finally, due to the problems found to retrieve the data from a dynamic source, like a webservice, data files were used. They were retrieved from the Australian Bureau of Statistics. To Gather the required information, 2 groups of data files were used.

Boundaries:
These files contains the geometric boundaries, name, code and state code of each region. These files have a GeoJSON format and are categorized by division level, the division levels used for these application were:

- State level
- Statistical division 4
- Statistical division 3
- Local government areas
- Suburbs

Crime data:
These files contains the name, and crime rate by year. They are categorized by state. Their format is CSV. Their region code and division level are unknown, they have to be determined in the matching process, searching for the appropriate region from the boundaries files.
Data structures

Boundaries
The structure of each region is defined by a set of polygons. Each polygon data was stored as an encoded list of latitude and longitude points. The data was encoded using a Google Utility library. It converts this list of points to a String that uses less memory and can be easy decoded when needed. Additionally, a bounding box was stored in order to find this data easily.

```java
public class PlaceStructure implements Serializable {
    private PlaceBounds bounds;
    private List<String> encodedPaths;
    private PlaceInfo placeInfo;
    private String code;
    ...
}
```

Place information
The data for each region was stored along with their name, code and state code. Depending on the source, and due to the fact that the crime data will be associated by name, all the possible names were stored in a list.

```java
public class PlaceInfo implements Serializable {
    private List<String> names;
    private String code;
    private Integer stateCode;
    private CrimeData crimeData;
    ...
}
```

Crime statistics
The crime statistics were stored with the related name and the crime indicators by year in a map.

```java
public class CrimeData implements Serializable{
    private String name;

    // Crime values by year <year,crime>
    private HashMap<Integer, Double> values;
    ...
}
```
Data matching

After gathering and storing the data in the appropriate data structures, the data was linked together. The data linking was done in the following order:

1. Load crime data
2. Load places information and boundaries
3. Match crime data with places without name errors
4. Match crime data with places with name errors
5. Identify the appropriate level to perform the crime match
6. Complete data in all levels different from the identified match
7. Simplify the geometric boundaries.
8. Persist data structures.

Taking into account that the crime data is indexed by place name, some problems raised:
- Names can have different order.
- Word can have misspellings.
- Names do not have an appropriate structure for matching.

To solve these problems a matching process was implemented:

1. Finding exact matches, without taking into account the order of the words. This solved cases like:

   "north carlton" ⇔ "carlton north"

2. Then, iteratively using the Edit Distance algorithm with a define threshold of errors to match the missing data. This help to match places with different spellings.

   "Bundoora" ⇔ "bundora"

3. Clean string a fix name structures. This fixed problems like:

   "carlton+north" ⇔ "carlton","carlton north"

Even with these three approaches some crime data was left without match, due to some places not having their associated boundaries or some places having completely different names. Below the result of the matching process will be presented.
## Division match

<table>
<thead>
<tr>
<th>State</th>
<th>Division level</th>
<th>Match percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSW</td>
<td>local_goverment_areas</td>
<td>98%</td>
</tr>
<tr>
<td>VIC</td>
<td>suburbs</td>
<td>92%</td>
</tr>
<tr>
<td>QLD</td>
<td>suburbs</td>
<td>44%</td>
</tr>
<tr>
<td>SA</td>
<td>local_goverment_areas</td>
<td>92%</td>
</tr>
<tr>
<td>WA</td>
<td>statistical_area_3</td>
<td>25%</td>
</tr>
<tr>
<td>NT</td>
<td>suburbs</td>
<td>57%</td>
</tr>
<tr>
<td>ACT</td>
<td>statistical_area_3</td>
<td>78%</td>
</tr>
<tr>
<td>TAS</td>
<td>states</td>
<td>100%</td>
</tr>
</tbody>
</table>

## Missing places

<table>
<thead>
<tr>
<th>State</th>
<th>Missing places</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSW</td>
<td>Lord Howe Island, Prisons</td>
</tr>
<tr>
<td>VIC</td>
<td>Avon Plains/Redbank, Ensay/Docters Flat, Ovens, Yeodene/Alvie, Molesworth,</td>
</tr>
<tr>
<td></td>
<td>Kinypanial/Korangvale, HMAS Cerebus, Awonga/Booropki/Kangawall, Ballangeigh,</td>
</tr>
<tr>
<td></td>
<td>Tempy, Prairie/Miloo/Piavella/Tennyson, Bolinda, Dookie College, Tabilk,</td>
</tr>
<tr>
<td></td>
<td>Buldah/Chandlers Creek, Black Heath/Brooksby/Bugalall, Berrybank, Keotong,</td>
</tr>
<tr>
<td></td>
<td>Lascelles, Nullawil, Cowangie, Borlands/Bullarto/Bullarto South, Barkly/Callawaddy, Gellibrand River/Kennedys Creek, Poolaigelo/Chetwynd, Macorna, Bendigo Roadside, Kernot/Korrine, Leeor, Glen Forbes, RAAF Laverton, Swan Hill Suburbs, Lillimur, Durham Ox, Tambo Crossing, Hedley, Speed, Pullut, Deakin University, Tintaldra, Tankerton, Booroolite/Gaffneys Creek, Stony Creek/Carmichael, Longerenong, Ballarat PO Boxes, Kotta, Darlington/Pura Pura, Bylands</td>
</tr>
<tr>
<td>QLD</td>
<td>Southern Region, Central Region, NorthBrisbane, Northern Region, PineRivers,</td>
</tr>
<tr>
<td></td>
<td>Mountisa, BrisbaneCentral, MetropolitanNorth Region, SouthEastern Region,</td>
</tr>
<tr>
<td></td>
<td>North Coast Region, SunshineCoast, Far Northern Region, Metropolitan</td>
</tr>
<tr>
<td></td>
<td>SouthRegion</td>
</tr>
<tr>
<td>SA</td>
<td>Orroroo/Carrieton (DC), Naracoorte and Lucindale (DC), Clare and Gilbert</td>
</tr>
<tr>
<td></td>
<td>Valleys (DC), Total, Port Pirie City and Dists (M), Berri and Barmera (DC)</td>
</tr>
<tr>
<td>WA</td>
<td>Peel District, South West District, Central Metropolitan District, Great</td>
</tr>
<tr>
<td></td>
<td>Southern District, REGIONAL WA REGION, North West Metropolitan District,</td>
</tr>
<tr>
<td></td>
<td>West Metropolitan District, East Metropolitan District, South East</td>
</tr>
<tr>
<td></td>
<td>Metropolitan District, METROPOLITAN REGION, South Metropolitan District</td>
</tr>
<tr>
<td>NT</td>
<td>NT balance</td>
</tr>
<tr>
<td>ACT</td>
<td>Canberra, Molonglo Valley</td>
</tr>
</tbody>
</table>
Preprocessing

Taking into account the large amount of data that needs to be processed, all the parsing, matching and indexing of the data were done in a preprocessing operation. This was done with a JAVA application and it was performed in 2 stages:

1. **Data parsing, matching and storing**
   In this stage all the data is read, parsed, stored in the appropriate data structure and linked together as previously described in the data matching section.

2. **Tiles files creation**
   Based on the bounds processed in the previous steps, the data is retrieved from the stored data structures. Next, the crime intensities for each pixel of each tile is calculated. Finally, these crime intensities are stored in a data structure that will be used by the android application to create the appropriate tile images.

To manage the coordinates transformation between tiles positions to latitude and longitude positions and vice versa, Mercator projections were used. These was due to the fact that Google maps API works with this type of projection.

Tiles creation is not a simple task. To identify the data needed to render each tile image the value of each pixel has to be found. Therefore, to increase the performance of this task a Scanline algorithm was implemented. This algorithm loops horizontally or vertical identifying the ranges that are part of the polygon with the purpose of later using these ranges to set the values in the appropriate pixel.
Storage
To store the large amount of data, places are stored in sections along with an index file. The index file contains each section with its bounding box to improve the performance of the search.

Indexing and retrieval of the data
With the purpose of fast retrieval, an index file is persisted. The file has the following structure:

```java
public class PlacesStorageData implements Serializable {
    // <division_level,bounds,file_path>
    private Map<String, Map<PlaceBounds, Map<Double, String>>> sections;

    // <division_level,max_crime>
    private Map<String, Double> maxCrime;
    ...
}
```

With this index file the appropriate section file paths can be found using their bounds. Each section has the following structure:

```java
public class SectionProcessedData implements Serializable {
    private String filename;

    //<place_identifier,PlaceStructure>
    private HashMap<String, PlaceStructure> memoryPlaces;

    private double simplifyRatio;
    private String divisionLevel;
    private int stateCode;
    ...
}
```

It stores the information for all the places inside the defined bounds, it also contains the simplify ratio, division level and state code of the section.

An example of how to execute a request over the indexed data is the following:

```java
PlacesStorageManager.getInstance().getPlaces(divisionLevel, bounds, simplifyRatio);
```
Data displaying, rendering

Google maps API for Android
To display the crime information in a map, the Google Maps Android v2 API is used. This mapping API allows location data management and rendering information over a map.

Drawing boundaries polygons through the API.
Knowing that using the original functionality of the API will produce the best resolution possible it was the first option to be tested. It worked well in high zoom levels but in low zoom levels the amount of data to be displayed proven to be too much for the API to create a proper user experience. To solve this issue, three techniques were tested, simplifying the boundaries of the places, merging or joining the region boundaries and using tile images layer to reduce the amount of processing resources needed to render the data.

Polygons simplification
The basic idea behind polygon simplification is to reduce the complexity of the segments maintaining the overall structure of a polygon. To produce this result, three consecutive points were selected and then the distance between the one in the middle and the segment created by the other two was calculated, if this distances go out of the established range for the current zoom level the point stays if not the point is deleted.

Polygons merging
Polygon merging is another technique chosen because it could decrease the complexity of the boundaries removing unnecessary segments and making the rendering easier to execute. The basic idea is to find common segments, remove them and try to build an structure that will contain both polygons. This process has a great complexity, identifying common segments is really difficult when the similarity validation is approximate.

Polygon merging proved to be a process that could not solve the performance rendering problem, bringing more problems than solutions due to its complexity, the large number of validations and processing resources required to produce a consistent merging.

Tiles images overlay
A tile overlay is a layer displayed over a map. It consists of a set of images that will be render depending on the zoom level, and the current viewport. As described in the preprocessing section, a Scanline algorithm was used to compute the data needed to create the tile images. This approach proved to be the most effective one, greatly increasing the performance of displaying the boundaries in the map.
Therefore, for high zoom levels the simplification approach is used, and for low zoom levels the tiles approach is used.
**User interaction**

User interaction is divided in two, the output from the application that users receive and the input that the users give to the application to get better result. The application uses a map to show the crime statistics based on a defined color gradient, and an info box to display information of an specific place. Users can use the application in several ways. These ways are: Exploring the map, going to their current location, or searching for a specific place.

The defined gradient is from green to red, green being the place with low criminal rate and red being the place with high criminal rate. These color were chosen based on a traffic light, where green implies safety, yellow implies warning and red danger. Additionally, selecting a place in the map will display an info box with the detailed data of the selected place. These details contain the crime statistics by year with their related color from the defined gradient.

The map component is provided by Google. It has well known capabilities of a map, zoom, dragging, rotating and tilting. Through this component the user is able to browse the map being able to locate the desire data. Additionally, depending on the zoom level and the current viewport of the map the crime data is adjusted to display more understandable information. For instance, the crime scale is modified depending on what places are currently being displayed.
Users will be able to go to their current location using the location service from Android. This service uses the location based on the current source available, making it not exclusive to GPS, and providing better results.

Search functionality was also added to improve the user interaction. Through this functionality, users are able to query for places by name getting suggestions and even working with an amount of errors thanks to the Google Geocode API. Additionally, the search history is persisted, helping the user to go back to previously searched places.

Finally, using the location service from Android, notifications are pushed to the user with the crime information related to their current location and, if supported, the phone led light will show the appropriate crime color level.
Design

Project structure
The android project was build with 7 main packages: model, preprocess, storage, services, tasks, ui and utils.

Model
This package contains the classes related to the main objects that will hold the data needed in the application.
Preprocess
The classes in this package are related to the preprocess operations. PreprocessMain is the main class through which the preprocess operations are executed. MissingCrimeDataFile contains the pending to match crime data and TilesProcessingInfo holds the information needed to avoid reprocessing already processed tiles.

Storage
This package contains the classes to manage the storage of data and also some data structures that need to be persisted.

The persistence manager is pluggable so there is a different implementation to be used when executing the preprocess operations and the actual execution of the Android application.

The place subpackage contains the classes for the persistence of data related to the places.
The search subpackage contains the classes related to the storage and management of the user search history.

![Search subpackage diagram]

Services
This package contains the classes to handle the services needed by the android application. These services are: tiles images creation, Overpass API loader and location tracking. The main difference between a service and a task in android is that services can run independently from the main application.

![Services diagram]

Tasks
These classes contain the functionality related to the asynchronous tasks needed by the application. These process are: Querying places, simplifying structures and processing search requests.

![Tasks diagram]
UI
The user interface is defined by the following classes. They are in charge of properly display all the crime data and interaction with the user.

Utils
The utilitarian classes needed in the project can be classify as follows:

Geometry and places utility classes.
Region data parsers for the different boundaries sources.

Functions needed in the rendering process, mainly needed by the tile overlay creation.

Functions needed to clean and process the strings. This functionality is mainly used by the matching process.
Libraries

**JTS Topology Suite**
In this project the JTS Topology Suite was used to handle the geometric operations. This operations are polygons intersection, distances, self intersections, building quadtrees and executing queries over these trees.

**Google Maps Android API Utility Library**
This utility library helped to add extra functionality to the original functionality of the Google Maps Android API v2. The functionalities used in this project were:

- **Polylines codification** as described in the Data Sources section, it was used to encode the paths of the polygons.
- Creating custom markers and infoboxes. This helped to improve the user experience displaying the appropriate information.

**Google Geocoding API**
To process the queries input by the user with the search functionality the Google Geocoding API was used. This API lets you call their web service with a set of parameters to get a list of places with their related information. This is a great API, it even works with a certain amount of misspellings produce a great user experience.

**File parsing libraries**
Two libraries were added to improve the parsing of specific file formats. These libraries are google-gson and opencsv. They helped to properly read json and csv files taking into account performance and one of their main objective is loading all the file data in memory.
Conclusions

- Indexing data based on location is important to improve the performance of the data retrieval. Storing the data that is geographically close together increases the speed to search of regions needed to be displayed. Additionally, storing the bounds of the polygons increases the performance of searching for regions.

- Rendering complex geometries and maintaining a proper performance is a challenge. In the case of this application, the built-in functionality of the Google Maps Android API was not enough for the required performance. Preprocessing the data, and using the scanline algorithm proved to be a proper solution for this problem.

- String matching is a complex process, and in the case of linking data I think it should be avoided as much as possible. Names and strings can produce a lot of errors, as shown in this project, and this problem can produce a lower quality in the results. A better way could be to try to index the data and linked them by a unique identifier.

- The user interface is crucial in any application. Therefore, the functionality should be built thinking on how the user is going to access the data. This approach will allow to identify the critical processes that should be optimized to produce a proper user interface.
References


