The LIMO Environment for Learning Programming using Interactive Map Activities

Ruby Y. Tahboub Jaewoo Shin Walid G. Aref Sunil Prabhakar
Department of Computer Science, Purdue University, West Lafayette, IN, USA
{rtahboub, shin152, aref, sunil}@cs.purdue.edu

***Abstract*—Advances in geographic information, interactive two- and three-dimensional map visualization accompanied with the proliferation of mobile devices and location data have tremendously benefited the development of geo-educational applications. In this paper, we introduce LIMO; a web-based programming environment that is centered around operations on interactive geographical maps, location-oriented data, and the operations of synthetic objects that move on the maps. LIMO materializes a low-cost open-ended environment that integrates interactive maps and spatial data (e.g., OpenStreetMap). The unique advantage of LIMO is that it relates programming concepts to interactive geographical maps and location data. LIMO offers an environment for students to learn how to program by providing 1. an easy to program library of map and spatial operations, 2. high-quality interactive map graphics, and 3. example programs that introduce users to writing programs in the LIMO environment.**

I. Introduction

In recent years, the interest in learning computer science among high-school students has significantly declined compared to other STEM subjects. According to the Computer Science Teaching Association CSTA [1] and the College Board [2], enrollment in introductory computer science courses has decreased in number by 17% from 2005. Despite the slight 4% increase in the most recent enrollment trend, academic institutions are challenged to develop innovative methods to improve retention and reverse the declining enrollment trends.

The key to inspire students to learn computer science lies in adopting an active learning approach [3]. In active learning, students use computations and programming to solve real-world problems. Many academic institutions have redesigned introductory computer science courses to include a visual component (e.g., Scratch [4] and Alice [5]), a hardware component (e.g., Finch robot [6] and Scribbler [7]), or a context (e.g., media computation [8] and graphics [9]). Although these approaches have successfully raised the interest in computing and made programming accessible, none of the environments provide a platform for maps as a pedagogical tool for learning programming. Interactive maps (e.g., based on OpenStreetMap, Google Maps, Bing Maps, or Google Earth) and location-based services provide an alternative option to learning a programming language by studying and applying abstract concepts. Furthermore, interactive maps offer a unique opportunity to add excitement to students while learning programming by relating programming concepts to locations and places that students are familiar with. Map activities, e.g., finding directions with certain properties, scale conversion, coordinate systems, distance computations, shortest paths in road networks, and spatial analysis are rich in semantics and help students develop strong computational thinking skills [10].

In this paper, we introduce LIMO; a web-based environment for learning computer programming using Interactive Map Operations. LIMO is centered around activities on interactive two- or three-dimensional geographical maps. Writing programs in LIMO is simple. First, the program scene is set as a map that is zoomed at a default location. Next, the user writes a standard Python script that is enrished by the LIMO library to encode the actions of the program's default moving object that is referred to as the *Commuter*. Finally, on running the program, the animation of the movements of the *Commuter* is displayed on the map and the program textual output is displayed on a dedicated output area.

Figure 1 demonstrates a "Hello World" program that is comprised of displaying "Hello World" on address '305 N University St, West Lafayette, IN 47907". This "Hello World" program is realized using a $read\_address$ function that takes as input the location/address where the *"Hello World"* message needs to be placed, and then the $display\_message$ function that displays the message on the map in the designated location.

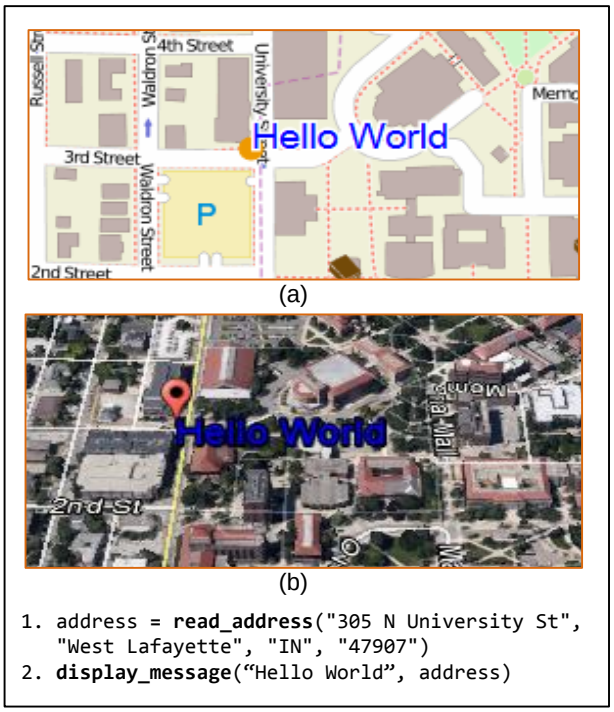


Fig. 1. The visualization of the "Hello World" program in LIMO using (a) OpenStreetMap, and (b) Google Earth.

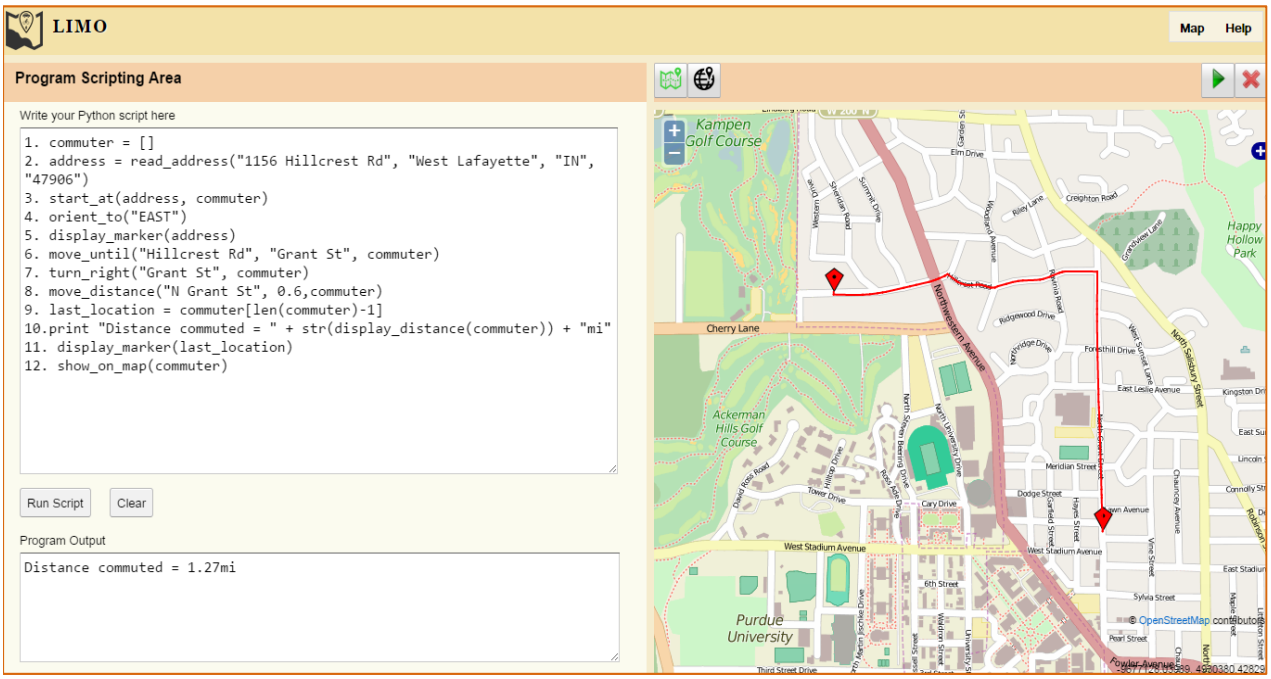


Fig. 2. The LIMO programming environment consists of a *program scripting area*, *a program output area*, and an *interactive map*.

The main contributions of this paper are as follows.

1) We introduce LIMO, a web-based programming environment centered around activities of interactive two- and three-dimensional maps. LIMO provides easy-to-program library of map and spatial operations with high-quality output graphics.
2) We introduce the LIMO library; a high-level Python-based map library that integrates *map* operations and *spatial* functions.
3) We provide example programs that demonstrate how the LIMO environment and library are used to write map-visualized programs. Furthermore, the examples showcase how LIMO can be used to demonstrate a variety of introductory programming concepts.

The rest of the paper proceeds as follows. Section II discusses the related work. Section III introduces the LIMO user interface, LIMO library, and system design. Section IV provides example LIMO programs and demonstrate its utility, and Section V contains concluding remarks.

II. Related Work

The traditional approach to learn a programming language involves studying and applying abstract concepts (e.g., syntax and semantics) that, by itself, represents a challenge for the young learners. In contrast, Visual programming environments e.g., Alice [5], Scratch [4], and Greenfoot [11] provide intuitive graphical interfaces. Hence, the learner focuses on programming rather than mastering the tedious programming constructs. A typical program in these visual environments is comprised of a personalized simulation with characters (i.e., actors) that tell a story. Program development in LIMO is similar to Scratch, Alice and other visual programming environments. However, the LIMO environment is specialized on the activities of interactive two- and three-dimensional geographical maps and locations as the program's primary story. Furthermore, LIMO is distinguished by incorporating interactive maps, location-oriented data, and spatial functions into programming constructs. Lastly, programs in LIMO are based on Python which makes LIMO reusable beyond being a motivational environment for introductory programming.

Teaching programming in context (i.e., using domains relevant to learners) has shown promising results in bringing interest into learning computing [12]. Several mature works have redesigned the introductory programming courses to incorporate real-world contexts (e.g., media computation and graphics). Guzdial [8] provides a media computation platform that introduces programming concepts through the manipulations of images, audio, and video. The work of [9] uses the context of graphics applications (e.g., photon mapping) to teach computer science topic e.g., data structures and algorithms. LIMO can also be viewed as an in-context programming environment. LIMO is distinguished by incorporating interactive maps, location-oriented data, and spatial functions into programming constructs.

Several works have integrated geographic datasets and interactive maps into geo-education. Teresco [13] utilizes highway data and Google maps to illustrate graph algorithms e.g., Dijkstra. GI@School and GeospatialLearning@PrimarySchool [14], [15] initiatives have developed educational modules that integrate Geographical Information (GI) science into high-school curricula. The GI initiatives cooperate with high schools to hold practice-oriented classes that raise interest in computing and GI as an emerging topic. One of the most popular modules is a Geocaching (i.e., locating hidden caches using a Global Positioning System GPS) application for the XO-Laptop based on OpenStreetMap. The GI initiatives and LIMO meet in integrating computing and rich

GI capabilities into learning programming. Furthermore, map integrated platforms raise interest in learning computing and nurture computational thinking skills.

III. The LIMO Programming Environment

We have realized a prototype for LIMO (Please see http://ibnkhaldun.cs.purdue.edu:8181/limo/). In this section, we present a first view on the LIMO environment including its user interface and programming library. Also, this section covers an overview of the LIMO system design.

A. LIMO User Interface

LIMO provides a web-based environment that integrates Python scripting and interactive maps to facilitate creating map-visualized programs. The LIMO user interface, illustrated in Figure 2, is comprised of three main parts: a *program scripting area*, a *program output area*, and an *interactive map*. Users can create programs by writing Python scripts that utilize the LIMO library to write map-centric programs. Furthermore, the LIMO interface integrates OpenStreetMap that facilitates exploring the various map features, e.g., the roads and the parks, and visualizing the output of the executed programs, e.g., an animation of the *Commuter* as it follows moving directions on the map. Finally, the program output area is dedicated to displaying textual output, e.g., computational results performed by the executing program.

Figure 2 gives an example program in LIMO that provides the commuting directions between two locations. The functions $start\_at$ and $orient\_to$ (Lines 3-4) specify the start location and direction of *Commuter*. Next, the $display\_marker$ function (Line 5) adds a marker at the current location of *Commuter*. The constructs $move\_until$, $move\_distance$, and $turn$ (Lines 6-8) control *Commuter's* movement on the map. Finally, the $display\_distance$ (Line 9) function prints in the program output area the *total distance* commuted so far. On running the program, the function *show_on_map* (Line 12) displays the animation of *Commuter's* movement on the map.

B. The LIMO Programming Library

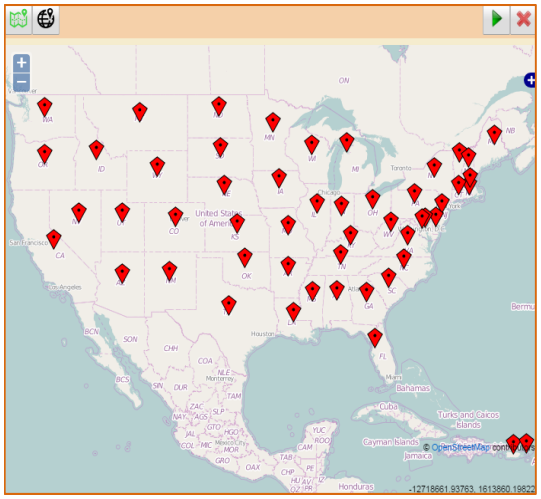
LIMO integrates interactive maps and rich location-oriented data and spatial data types into an easy to use programming library. The LIMO library functions are designed to enable users to process, customize, and visualize location data on the map. The LIMO library offers two categories of functions: *map* basics, and *spatial*.

The *map* functions enable users to interact with map and perform primitive activities (e.g., displaying a message or adding a marker on map). Location in the basic constructs is represented as an actual address that can be explicit (e.g., intersection of two streets) or implicit (e.g., *commuter's* current location). Moreover, the basic map functions are suitable for beginners with no map programming experience. For instance, the semantics of *display*, *move*, and *turn* are intuitive and have bases in reality. Hence, a wide variety of simple programs can be written to describe the various movements of *Commuter* while keeping track of time and distance.

The *spatial* functions encapsulate complex location-based data and operations into easy-to-use functions. Location is represented as a geo-coordinate point (e.g., using latitude and longitude). Spatial functions enable creating programs that incorporate real-world location data and apply it in creative ways. For instance, $get\_location$ seamlessly converts a textual address to its equivalent geo-coordinates, get_all is another rich spatial function that provides a list of locations (in the form of geo-coordinates or shapes) that match a place *description* parameter (e.g., all airports in Indiana). Consequently, given the current location of *Commuter*, one program may iterate over the list of interesting locations and find the closest

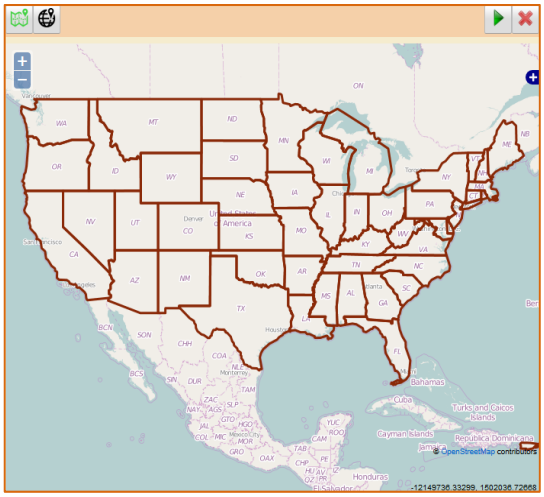
TABLE I. Sample LIMO Programming Library.

Category	Function Name	Description/ Options
Map (Basics)	start_at(address, commuter)	Sets Commuter's start location to address
	orient_to(direction)	Direct Commuter towards a certain direction, e.g., East, West, North, or South
	move_distance(street, distance, commuter), move_until(street1, street2, commuter)	Move Commuter for certain distance or until a clear intersection
	turn_[right|left](street, commuter)	Re-orient Commuter towards a new direction, e.g., right or left
	display_message (message, address| location)	Place a text message, e.g., at given address or at geo-location
	display_marker (address| location)	Place a marker on map at given address or at geo-location
	display_[distance| time](commuter)	Display total distance/time commuted so far
	compute_distance (add1, add2)	Return the distance between two addresses or geo-locations
Spatial	get_location(address)	Return a point that represents the geo-coordinate of address
	get(name, description, geometric shape)	Return the location (as geometric shape) of the place that matches place-name and discription
	get_all(description, geometric shape)	Return a list of locations (as geometric shape) for places that match discription
	overlaps| touches| intersects| contains (shape1, shape2)	Boolean operators that test whether two shapes: overlap, touch, intersect, or contain one another
	display_shape (geometric shape)	Display geometric shape (e.g., lake boundary) on map



```
1. all_states = get_all("STATE", "POINT")
2. for i in range(len(all_states)):
3.  display_marker(all_states[i])
```

(a)



```
1. all_states = get_all("STATE", "POLYGON")
2. for i in range(len(all_states)):
3.  display_shape(all_states[i])
```

(b)

Fig. 3. Sample program using get_all construct with ”POINT” and ”POLYGON” parameter options, (a) displays a marker on each state, (b) displays a polygon around each state.

(or farthest) location to the *Commuter*'s location. Another program may simply visualize all the qualifying locations on a suitably scaled map. Figure 3a-b illustrates two sample programs that use the *get_all* construct to retrieve a list of states. The second parameter in *get_all* determines the type of the returned shape, e.g., POINT (Line a-1) returns a geocoordinate point and POLYGON (Line b-1) returns a polygon shape. Moreover, spatial predicates, e.g., *intersect* or *contains*, further enrich the programming library by enabling spatial tests, e.g., does the park contain a lake? Refer to Table I for a list of sample LIMO programming library.

C. System Design

Figure 4 gives the process flow of a LIMO program inside the LIMO programming environment. In the editing phase, the user creates and edits her program. Upon program execution, the LIMO backend analyzes the program script and invoke the proper LIMO library functions to carry out map and spatial operations. For example, the ***display_marker****(“1001 Hillcrest Rd ...”)* requires a spatial query that obtains the geo-coordinates of the address parameter. The geo-coordinates are used to visualize the construct on the map. The backend performs the actions specified by the program. Finally, in the execution phase, the program output is visualized on the map and the textual output is displayed on the output area.

The LIMO programming environment is a web application that follows a multi-tier architecture as shown in Figure 5. LIMO's Presentation Layer embodies a web-based user interface that integrates a scripting area and an interactive map. The middle layer (i.e., Logic) consists of a Jython interpreter, the LIMO library and a customized map visualization library. Finally, the Data Layer uses a relational database that handles spatial datasets. We discuss the technical details of each layer next.

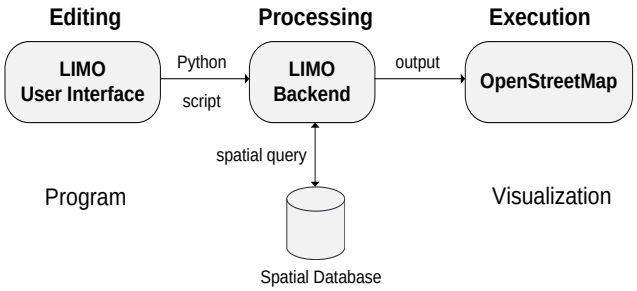


Fig. 4. Process flow diagram of a program in LIMO.

1) Presentation Layer: LIMO's user interface is built using Google Web Toolkit (GWT) [16]: an open-source web tool for developing JavaScript front-end applications. GWT is appealing for LIMO due to its cross-browser compatibility, efficient testing and debugging, and not mandating prior background in web languages (e.g., JavaScript or HTML).

An interactive map based on OpenStreetMap (OSM) is integrated into LIMO's user interface. OSM provides a free, editable map of the world. Moreover, OSM offers open access to map datasets. Finally, we use OpenLayers API [17] (open source library for map development) for realizing visualizations on LIMO's interactive map.

2) Logic Layer: The Logic Layer (also termed the Application Layer) plays a key role in executing programs in LIMO including performing computations and communicating with the Data Layer. The Jython interpreter processes the python script and utilizes the LIMO library to carry out spatial and map operations. On program execution, the textual output is displayed on the program output area and the map related data is moved to a customized map visualization library to

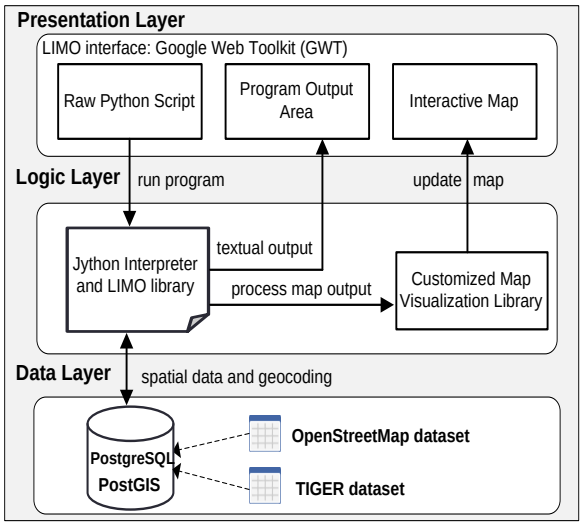


Fig. 5. The architecture of the LIMO programming environment.

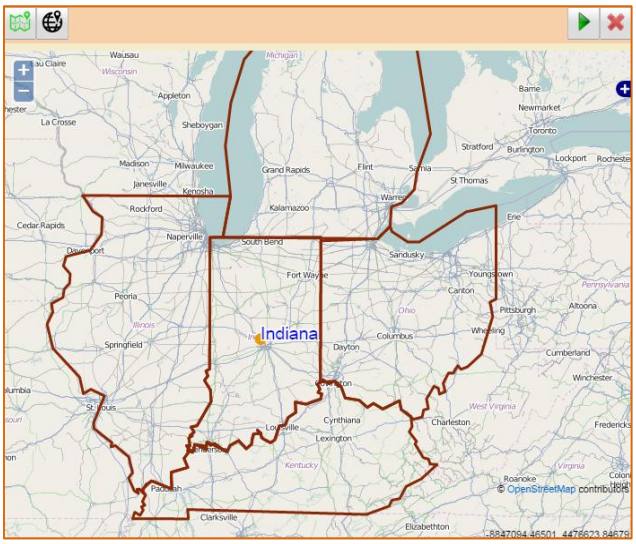


Fig. 6. The output of Program 2. The map gives the boundaries of the states that have shared borders with Indiana.

create object movement animation. Finally, the interactive map is updated with program visualizations.

3) Data Layer: The Data Layer is comprised of a relational database management system (DBMS) and stored datasets. LIMO deploys PostgreSQL [18]: an open-source relational DBMS along with PostGIS [19]: an open-source extension that adds support for spatial data to PostgreSQL.

The spatial database in LIMO stores TIGER [20] and OpenStreetMap datasets. TIGER is a public dataset administrated by the U.S. Census Bureau [21]. TIGER consists of geometric data that features roads, railroads, rivers, in addition to legal and statistical geographic areas. LIMO uses TIGER to support geocoding, e.g., translating between address and geo-coordinates. Geocoding is crucial for LIMO in order to visualize the actions of constructs on the map. OpenStreetMap contains rich datasets about the locations of points of interests (e.g., tourist attractions), lines (e.g., roads) and areas (e.g., lakes).

D. The LIMO Prototype

The LIMO project is being developed in Purdue University [22]. The live LIMO environment and sample programs can be accessed using http://ibnkhaldun.cs.purdue.edu:8181/limo/

IV. Sample Programs

Figures 1 and 2 present the *Hello World* and the *Commuting Directions*, respectively. This section provides additional sample LIMO programs to demonstrate the various programming library functions and showcase how the LIMO environment can be used to create stimulating open-ended programming exercises.

A. Basic Programs

Program 1. This program presents an example of utilizing *display*, *compute-distance*, and conditional structure *If ... Else*. The goal of the program is to determine whether the user should walk or bike from home to office. The user is willing to walk *only if* the distance to office is less than 3 miles. Otherwise, she prefers to bike. Moreover, the program adds a marker and a message on the locations of home and office.

```
1. Home = read_address("1001 Hillcrest
   Dr.", "West Lafayette", "IN", "47906")
2. Office = read_address("Airport Dr.",
   "West Lafayette", "IN", "47907")
3. display_marker(Home)
4. display_message("Start", Home)
5. display_marker(Office)
6. display_message("Destination", Office)
7. distance = compute_distance(Home,
   Office)
8. if distance < 3 :
9.  print "I'll be walking!"
10. else:
11.  print "I'll be biking"
```

A simple extension to Program 1 is to add a third option for driving in case the distance between home and office is greater than 5 miles.

Program 2. This program presents an example of utilizing *get*, *get_all*, *display_shape*, the spatial operator *touches*, conditional and looping structures. The goal of the program is to iterate over a list of geometries that represent the boundaries of states. It is required to identify the states that share borders with *Indiana* and display a boundary around each one of them.

```
1. Ind_pol= get("Indiana", "STATE",
   "POLYGON")
2. Ind_pnt= get("Indiana", "STATE",
  "POINT")
3. display_message("Indiana", Ind_pnt)
4. display_shape(Indiana)
5. all_states = get_all("STATE",
   "POLYGON")
```

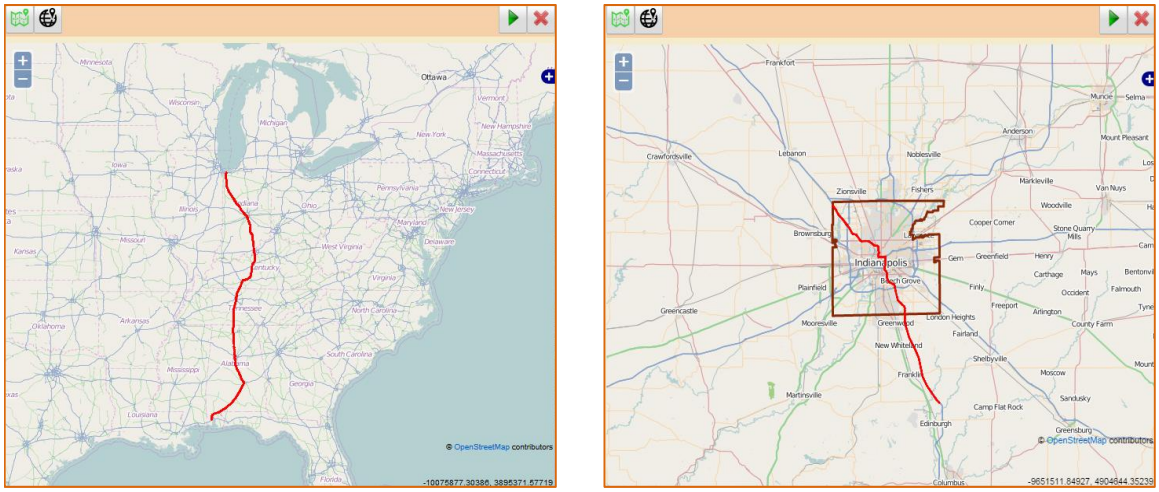


Fig. 7. The output of Program 3. The map gives (a) I-65 Highway segments spanning multiple states (b) I-65 Highway segments that intersect the boundary of Indianapolis.

```
6.  for state in all_states:
7.    if touches (state, Ind_pol):
8.      display_shape(state)
```

A related program finds the states through which the Wabash river flows. The program first gets the geometry of the river then uses the spatial function *intersects* to test against each state boundary in the list as follows.

```
1.  Wabash-river = get("Wabash",
    "RIVER", "POLYLINE")
    ...
4.    if intersects(state, Wabash-river)
    ...
```

Program 3. This program presents an example of utilizing *get*, *display_shape*, the spatial operator *intersects*, conditional and looping structures. The goal of the program is to display the I-65 Highway segments that intersect the boundary of the city Indianapolis. The *get* constucts (Line 1) obtains a list termed *I65_segments* that contains all of the I-65 Highway segments. Refer to Figure 7a for illustration, the *I65_segments* spans multiple states (e.g., Michigan, Indiana, ...). Next, the boundary of Indianapolis city is obtained (Line 2) and the spatial operator *intersects* is used inside a looping structure to filter out the unqualified segments (Lines 4-6). Figure 7b gives the output of the program.

```
1.  I65_segments = get("I- 65",
    "PRIMARY-ROAD", "POLYLINE")
2.  Indy_pol = get("Indiana", "CITY",
    "POLYGON")
3.  display_shape(Indy_pol)
4.  for i in range(len(I65_segments)):
5.    if intersects (I65_segments[i],
      Ind_pol):
6.      display_shape(I65_segments[i])
```

A related exercise on lists processing finds the states where I-65 Highway passes. Next, for each state in the list the program prints 1. cities and towns 2. finds the state with the mimimum and the maximum number of the Highway segments. Moreover, *add_marker* and *display_message* functions can be used annotate the answers on map.

Program 4. This program presents an example of utilizing *get-all*, *compute_distance*, *for* loop, lists and the list sort utility. The goal of the program is to obtain a list of the geo-coordinate locations of airports in a given area (e.g., state of Indiana) and find the closest five airports to a *Commuter's* location. The sample program uses the list sort utility function to identify the closest airports. Next, the user iterates over closest airports to (1) compute and display the distance between her location and each airport (2) at each airport location on map, add the airport name and marker. Figure 8 gives the output of the program.

```
1.  address = read_address("1156 Hillcrest
    Rd", "West Lafayette", "IN", "47906")
2.  my_location = get_location(address)
3.  display_message("HOME", my_location)
4.  airport_list = get_all("AIRPORT",
    "POINT")
5.  for i in range(len(airport_list)):
6.    airport_loc = (airport_list[i][0],
      airport_list[i][1])
7.    distance = compute_distance(my_location,
      airport_loc)
8.    airport_list[i].append(distance)
9.  airport_list = sorted(airport_list,
    key =lambda x : x[3])
10. for i in range(5):
11.   airport_loc = (airport_list[i][0],
      airport_list[i][1])
12.   airport_name = airport_list[i][2]
13.   display_marker(airport_loc)
14.   display_message(airport_name,airport_loc)
15. print airport_name,
```

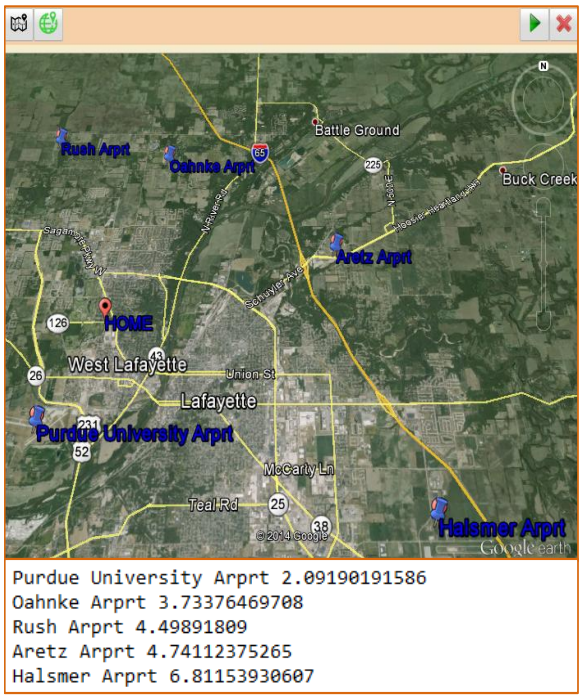


Fig. 8. The output of Program 4. The map gives the locations of the five closest airports to the HOME location.

```
airport_list[i][3]
```

B. Advanced Problems

Interactive maps can be used to construct open-ended programming problems, i.e., ones that can have multiple potential solutions. In the following, we present ideas for some open-ended problems relevant to the LIMO programming environment.

Program 5. **The design of a 5K race route**. This problem has a restriction on the race distance in addition to the start and end locations. In contrast to the *Commute Directions* program, there might be multiple or even *no* possible solutions.

Program 6. **Planning a road trip**. This problem involves choosing the points of interest and allocating the gas budget that is determined by the total miles to be commuted. Many compelling questions can be addressed in this setting. For instance, finding a potential trip route that allows the user to visit as many places as the budget and miles restrictions allow.

Program 7. **Data visualization activities**. There are many interesting real-world activities that involve visualizing data on maps, e.g., keeping track of social media friends and followers around the country. Another example is monitoring the local weather e.g., storms, snow accumulation, and other relevant weather data.

Program 8. **The design of a map game**. Hunting for hidden treasures is a very intriguing game for all ages. Markers on the map can be used to represent treasures and the goal is to use the LIMO library to encode the path to each treasure. In this game, user-defined metrics (e.g., the distance or the number of turns) can be used as scoring criteria i.e., the value of a hidden treasure may increase when the user finds a path that minimizes the total number of turns.

V. Concluding Remarks

The computer science education community has placed tremendous resources to raise interest in computing and improve retention. A substantial share of these efforts relies on developing innovative programming environments that target the young learners. Along this line, we introduce LIMO- a web-based environment for learning programming using activities related to interactive two- and three-dimensional geographical maps and the operations of moving objects on the maps. An essential aspect of LIMO is that it provides a diverse programming library that enables users to process, customize, and visualize location data on the map. In our view, the integration of interactive maps, location-based data, and delivering seamless spatial constructs within an educational programming environment is a very promising direction. We plan to develop LIMO further by extending the LIMO library with additional spatial and visual functions. Moreover, we plan to conduct a usability study that involves novice users to assess the efficacy of using LIMO while learning programming.

VI. Acknowledgements

This research is supported in part by the National Science Foundation under Grants III-1117766 and III-0964639.

References

[1] Computer science teaching association. [Online]. Available: http://csta.acm.org/Research/sub/HighSchoolSurveys.html

[2] College board. [Online]. Available: http://www.collegeboard.org

[3] J. J. McConnell, "Active learning and its use in computer science."

[4] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond., "The scratch programming language and environment." *TOCE*, vol. 10(4), 2010.

[5] C. Kelleher, R. Pausch, and S. Kiesler, "Storytelling alice motivates middle school girls to learn computer programming." in *CHI*, 2007, pp. 1455–1464.

[6] T. Lauwers and I. Nourbakhsh., "Designing the finch: Creating a robot aligned to computer science concepts." in *AAAI*, 2010.

[7] Scribbler. [Online]. Available: http://www.parallax.com/product/28136

[8] M. Guzdial, "A media computation course for non-majors." *SIGCSE Bulletin*, vol. 35, pp. 104–108, 2003.

[9] S. Matzko and T. A. Davis, "A graphics-based approach to data structures," in *ITiCSE*, 109-113, p. 2008.

[10] J. M. Wing., "Computational thinking." *CACM*, vol. 49(3), pp. 33–35, 2006.

[11] M. Kölling, "The greenfoot programming environment."

[12] S. Cooper and S. Cunningham, "Teaching computer science in contex," *ACM Inroads*, vol. 1(1), 2010.

[13] J. D. Teresco, "Highway data and map visualizations for educational use," in *SIGCSE*, 2012, pp. 553–558.

[14] T. Bartoschek, G. Gundelsweiler, and C. Brox., "Gi@ school: Gi education and marketing at high schools." in *AGILE*, 2007.

[15] T. Bartoschek, H. Bredel, , and M. Forster., "Geospatiallearning@ primaryschool: A minimal gis approach." in *GIScience 2010 extended abstracts*, 2010.

[16] Google web toolkit. [Online]. Available: http://www.gwtproject.org

[17] Open layers. [Online]. Available: http://openlayers.org

[18] Postgresql. [Online]. Available: http://www.postgresql.org

[19] Postgis. [Online]. Available: http://postgis.org

[20] Census tiger. [Online]. Available: http://www.census.gov/geo/maps-data/data/tiger.html

[21] Us census bureau. [Online]. Available: http://www.census.gov

[22] The limo environment. [Online]. Available: http://mps.cs.purdue.edu/wiki/index.php/LIMO