EECS 22L: Project 2 (Taxi Cab Management System)

Prepared by: Nistha Tandiya, Shang Ma, and Prof. Rainer Dömer

Feb 25, 2015

The City of New Irvine is soliciting proposals for the development of a new fully-automated taxi cab management service that can optimally coordinate a number of taxi cabs serving the individual public transportation needs of the population and visitors of the city.

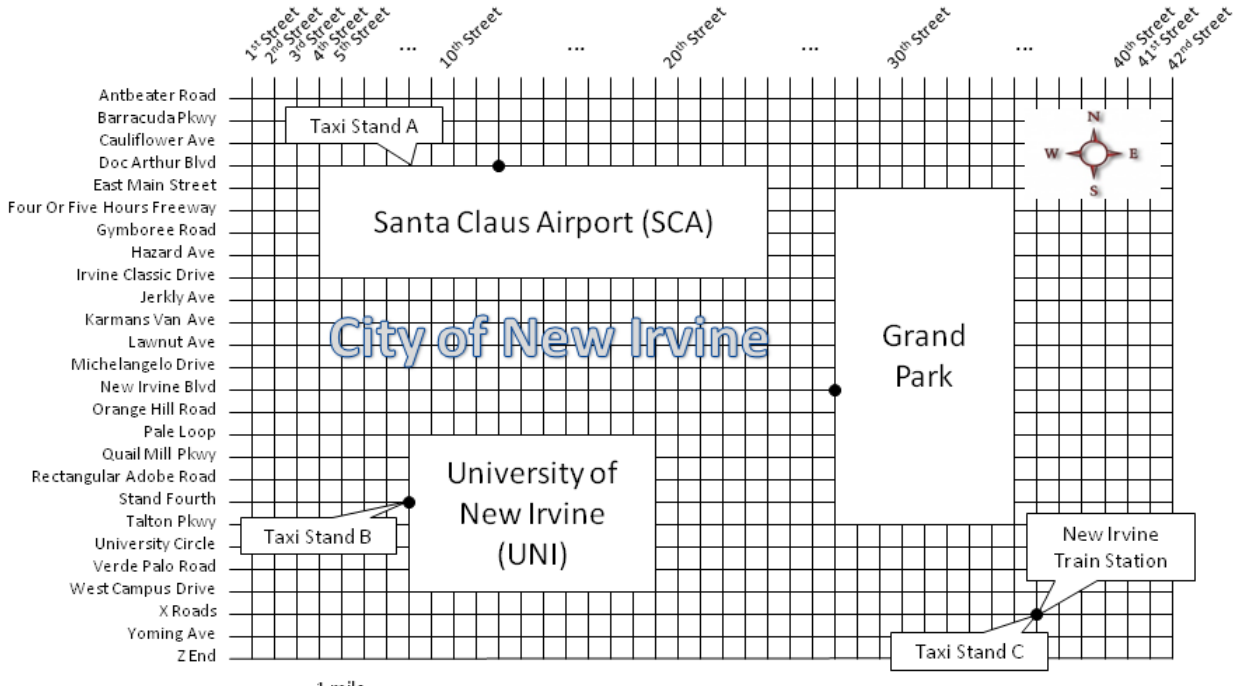

Figure 1: Map of the City of New Irvine.

As shown in the map detailed in Figure 1 above, the City of New Irvine is planned as a grid of 26 by 42 roads intercepted by three major land marks, namely the University of New Irvine (UNI), the international Santa Claus Airport (SCA), and the Grand Park. For public transportation, there is also the New Irvine Train Station and three dedicated taxi cab stands A, B, and C, strategically co-located at major destinations.

1 Envisioned Public Transportation

Due to its ingenious Manhattan-style street plan with blocks a quarter of a mile apart, the City of New Irvine expects transportation by taxi to be very flexible and efficient for individual people and small groups.

1.1 Goals and Requirements

Taxi cab customers may request rides from and to every street corner within the city limits, to be served at the time of request (i.e. as soon as possible) or at a defined later time as specified by the customer. As a goal, the majority of taxi rides within the city limits should take less than 20 minutes, including the time the customer waits for the cab to arrive. Of course, the posted speed limit of 45 miles per hour in the city must be strictly observed.

Three designated taxi stands are planned at strategically placed locations: Stand A at the local airport (corner of Doc Arthur Blvd and 8th Street), Stand B at the university (Stand Fourth and 8th Street), and Stand C at the train station (X Roads and 36th Street), as indicated in the map above. Each taxi stand provides parking for up to 12 taxi cabs. Cab cars may be limousine or van type with an occupancy limit of 3 or 6 passengers, respectively.

The vision of the New Irvine Great Plan is to facilitate individual transportation as fairly and as easily as possible. Specifically, the plan is to establish a central computer server where all incoming transportation requests from customers are automatically processed and taxi drivers can fairly compete for trips and compensation. At the same time, customers'expectations of punctuality and quick trips are to be met to the best extent possible.

The City of New Irvine has strictly regulated the cost of taxi rides. Unless a customer is picked up or dropped off directly at a taxi stand, a base fee of $3.75 is charged. In addition, a distance dependent charge of $2.00 per mile is applied. No tips are expected nor allowed. For calculating a fare, the minimal Manhattan distance is applied between the start and end points of the customer's ride.

For taxi drivers, every block driven as instructed by the central server earns $0.20. This includes driving to customer pickup and return from drop-off locations. However, waiting at a taxi stand parking lot does not earn any compensation.

All charges for taxi rides are centrally processed at the management server. Revenue may be earned from customer payments after taxi drivers are compensated. Thus, to maximize revenue, taxi rides need to be optimally scheduled and routed. Note that combining multiple taxi rides can increase revenue significantly as each customer pays their ride, but drivers are compensated only for distance driven.

1.2 System Specification

In order to ensure interoperability between software solutions provided by different vendors, the City of New Irvine has conducted an early research study to determine the overall system of planned computer components and the associated communication protocols needed to facilitate the taxi cab coordination services. A hired professional consultant has created a proposal for the overall taxi management system which the City of New Irvine has now approved for adoption. The consultants proposal consisting of three computer-based components is outlined below in Figure 2.
The following sections detail the specification of the client app Call Taxi, the server program Manage Taxis, and the server app Taxi Cab.

1.2.1 Client App "Call Taxi"

In order for customers to call for or reserve a taxi ride, an application "Call Taxi" is envisioned that may be implemented on a variety of devices, including stationary computer terminals, portable computers, and mobile phones. The application may be web-based or come in form of specially prepared software apps, but is expected to communicate with the central server via standard internet protocol.

The "Call Taxi" application will guide the customer in specifying the starting point and destination of the desired taxi ride, as well as the time for pickup and any other special requests. The complete customer request will then be sent to the central server for processing. The generated response by the server will then be displayed to the customer in order to confirm the trip reservation and indicate expected costs, or decline the request (e.g. when there is no taxi cab available).

In summary, the "Call Taxi" app will serve as an easy-to-use customer interface to the central taxi management system.

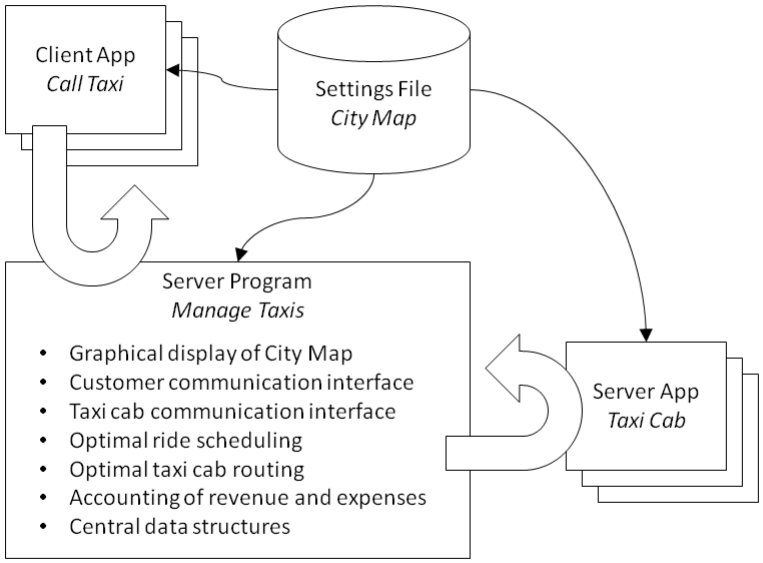

Figure 2: Overview of the planned Taxi Management System.

1.2.2 Central Server Program "Manage Taxis"

A central server program called "Manage Taxis" will receive and process all incoming customer requests via a standard internet-based communication protocol. Every request is expected to be processed fully automatically and an instantaneous response (within a few seconds time) shall be generated. To maximize customers'satisfaction, requests shall be serviced quickly and trips shall be efficiently routed.

To service customers'rides, the central server program will communicate automatically with autonomous devices located in the taxi cab cars. The cab cars are location-aware (GPS based) and will respond with their current position in the city whenever requested. It is up to the central server, however, to route the taxis to their pick-up and drop-off locations, so that trips can be centrally optimized and possibly combined. Unless a customer has specifically requested a first-class ride (no other parties in the cab), taxi rides may be combined (multiple parties in the cab, up to the maximum occupancy of the cab car) at any time in order to minimize energy consumption, save costs, and maximize taxi management and cab driver profit.

Taxi cabs may be routed via any street within the city limits, but must not drive through private land (i.e. the blocks occupied by the landmark locations). To avoid traffic congestion, taxi cabs are supposed to wait only at the taxi stands (in the designated parking lots for taxis). Unless driving to service customers ride requests, taxi cabs may not wait (idle parking) on the city streets.

Generally it is important to note that taxi driving routes are to be optimized by the central server (to use the shortest path). However, planned routes may be revised (updated) at any point in time, for example, when another customer ride lies "on the way". To facilitate such online scheduling and routing optimization, the central server can relay optimized directions to the taxi cabs at any time (any block), indicating to continue into any direction including U-turns (at every street crossing, the server can tell the taxi to go either North, South, East, or West).

The central taxi management system shall provide a graphical user interface (GUI) that shows the current locations of taxi cabs and customers in real-time. The management system must also handle the accounting needed for income

(customers pay with credit cards at the time of reservation), expenses (taxi drivers must be compensated for driving, but not for waiting), and resulting revenue (income minus expenses; see taxi revenue regulations by the City of New Irvine above).

1.2.3 Server App "Taxi Cab"

Each taxi cab car is equipped with a GPS-based mobile computer system that automatically tracks the current position and can autonomously respond to location requests by the central server. The taxi cab device also keeps track of routing information (path selected by the server) and displays it to the driver who can then focus on safe driving. Taxi drivers will follow the instructions requested by the central server unless traffic conditions dont allow it. For example, a taxi will not drive through the Grand Park which is reserved exclusively for recreation activities of City of New Irvine residents.

Customer pickup and drop-off will also be performed only upon instruction by the central server. Thus, with the exception of observing local traffic situations, taxi drivers will not make any decisions but solely rely on the optimal planning by the central management service.

2 Program Specification

In this project, we want a software program which can manage a taxi company in a city. Its major task would be to earn profit by catering to the incoming customer requests.

There are several features that we expect the program to have. We distinguish between the minimum requirements for this project, and more advanced options that are goals and optional features (bonus points).

2.1 Basic functions that are required:

1. Ability to pick and drop customers at their specified place and time (now or at a time later the same day)
2. Route and schedule taxis in a city considering its roads and landmarks
3. Understand and implement a given map of any city according to the settings file
4. Graphical user interface (GUI) to display the map along with current customer and taxi locations
5. Maintain accounting for the company (income and expenses)
6. Unit test for all major modules in the project.
7. Well structured user/software deliverable and detailed documentation which meets the requirements specified in the grading criteria.

2.2 Advanced options that are desirable (but optional): (Bonus)

In addition to the above minimum requirements, additional features may be added:

- Clients can specify the number of passengers for each cab request (default: 1 rider)
- A clock specifying current time can be displayed on the map
- Profit earned by company can be displayed
- There can be a provision for first class ride (the passenger does not share the ride with anyone)
- There can be a GUI for clients to input requests

Of course, any other features are also welcome which can make taxi management software more efficient and easier to use.

3 Software Engineering Approach

In the design and implementation of this project, we will follow basic principles of software engineering in a close-to-real-world setting. You will practice the major tasks in software engineering to build your own software product. As for project 1, we will not provide detailed instructions on how to design the program. Instead, your team needs to come up with your own choices and practice designing the software architecture of a medium-size program and document it.

3.1 Team work

The software design and programming in this course will be performed by student teams. Teams of 6-8 students will be formed at the beginning of this project. Team work is an essential aspect of this class and every student needs to contribute to the team effort. While tasks may be assigned in a team to individual members, all members eventually share the responsibility for the project deliverables.

The overall tasks of software design, implementation and documentation should be partitioned among the team members, for example, to be performed by individuals or pairs (pair programming). A possible separation into tasks or program modules may include:

- main server
- graphical user interface
- communication module (server to customer, server to taxi)
- routing and scheduling taxis
- managing finances (accounting of expenses and income)
- documentation
- testing

When planning the team partitioning, keep in mind that certain tasks depend on others and that some tasks are best handled by everyone together. However, for this taxi project in particular, there is a good opportunity to exploit the fact that the overall structure of the program is component-based. The communicating components can easily be implemented and tested separately. Also, dummy components, e.g. an automatic customer request generator, may be used to test the other system components.

A team account will be provided on the `crystalcove` and `zuma` server for each team to share source codes, data and document files among the team members. Since teams will compete in the projects, sharing of files across teams is not permitted.

Every student is expected to show up and participate in team meetings. Attendance of the weekly discussion and lab sections is mandatory for the sake of successful team work.

3.2 Major project tasks

As for the previous project, we again go through several steps to approach this medium-sized programming project, with **some changes**.

- *Design the software application specification*: work as a team to decide the functionalities of the program, the *input* and *output* of the program, and other things that describe the *features* of the program for the users. Again, we will write an user manual to document this.
- *Design the software architecture specification*: work as a team to design the data structure, program modules, application programming interface (API) functions between modules, and basic algorithms that will be used to solve the problem. This will be documented again in a software specification document, but special attention should be given to the **testing plan**.

- *Build the software package*: write the source code and implement the program. Each team member may be in charge of their own module(s) and ideally work in parallel on implementation and module testing. Use Makefile for rule-based compilation to integrate the modules from different owners. Here, will use **unit testing** for each module before integration into the entire software system.
- *Version control and collaboration*: use a version control application, i.e. CVS (introduced in Lecture 3) to maintain the team project documentation and source code files. Team members can synchronize their own work with the others through the team repository located in the team account. In contrast to Project 1, **the use of CVS in Project 2 is mandatory**.
- *Test and debug the software*: work as a team to decide the testing strategies, write automated test programs or scripts, and debug the program when some of the test cases fail. Both unit tests and **system test** are needed during software development and delivery.
- *Software release*: release the software package with the executable program and documentation, e.g. the README file, user manual, etc. Release also the source code as a package for the further developers or maintainers. In contrast to Project 1, Project 2 has 3 software releases, **alpha**, **beta**, and **final release**.

3.3 Deliverables

Each team needs to work together and submit one set of deliverables each week. Here is the checklist of the files the team needs to submit and the due dates (hard deadlines).

Table 1: The Taxi Cab Project Deliverables

Week	File Name	File Description	Due Date
1	`Taxi_UserManual.pdf`	The application specification	02/16/15 at 12:00pm
2	`Taxi_SoftwareSpec.pdf`	The software architecture specification	02/23/15 at 12:00pm
3	`Taxi_Alpha.tar.gz` `Taxi_Alpha_src.tar.gz`	The alpha version of the Taxi program, including the program source code and documentation	03/02/15 at 12:00pm
4	`Taxi_Beta.tar.gz` `Taxi_Beta_src.tar.gz`	The beta version of the Taxi program, including the program source code and documentation	03/09/15 at 12:00pm
5	`Taxi_V1.0.tar.gz` `Taxi_V1.0_src.tar.gz`	The release software package for the Taxi program and the program source code and documentation	03/16/15 at 12:00pm

Note that we do require these exact file names. If you use different file names, we will not see your files for grading.

We will separately provide detailed templates (document skeleton, table of expected contents) for the textual documents and a detailed list of contents (directory structure and expected files) for the file archives. These grading criteria will be provided at the Projects tab of the course webpage.

3.4 Submission for grading

To submit your team's work, you have to be logged in the server `zuma` or `crystalcove` by using your **team's account**. Also, you need to create a directory named `taxi` in your team account, and put all the deliverables in that directory. Next, change the current directory to the directory containing the `taxi` directory. Then type the command:
`% /ecelib/bin/turnin22`
which will guide you through the submission process.

For each deliverable, You will be asked if you want to submit the file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.

Below is an example of how you would submit your team work:

```
zuma% ls # This step is just to make sure that you are in the correct directory that contains taxi/
taxi/
zuma% /ecelib/bin/turnin22
============================================================
EECSL 22L Winter 2015:
Project "taxi" submission for team1
Due date:  Mon Feb 16 12:00:00 2015
* Looking for files:
* Taxi_UserManual.pdf
============================================================
Please confirm the following:  *
"I have read the Section on Academic Honesty in the *
UCI Catalogue of Classes (available online at *
http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) *
and submit original work accordingly." *
Please type YES to confirm.  y
============================================================
Submit Taxi_UserManual.pdf [yes, no]?  y
File Taxi_UserManual.pdf has been submitted
============================================================
Summary:
============================================================
Submitted on Sun Feb 9 18:55:31 2015
You just submitted file(s):
Taxi_UserManual.pdf
zuma% _
```

You may want to verify your submission. This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:

```
zuma% /users/grad2/doemer/eecs22/bin/listfiles.py
```

This command lists your submitted files. More importantly, be sure to double-check that the submitted packages can be cleanly extracted and used (graded!).

For a binary package, we expect the user to read the documentation and run the executable program as follows:

```
% gtar xvzf BinaryArchive.tar.gz
% evince taxi/doc/Taxi_UserManual.pdf
% taxi/bin/Taxi
```

For a source code package, we expect the developer to read the documentation and build the software as follows:

```
% gtar xvzf SourceArchive.tar.gz
% evince taxi/doc/Taxi_SoftwareSpec.pdf
% cd taxi
% make
% make test
% make clean
```

Again, please ensure that these commands execute cleanly on your submitted packages.

4 Design and Implementation Hints

For this second project in EECS 22L, we provide in the following a few suggestions towards effective software design and implementation.

4.1 System Management Settings

For future maintenance and extension, the automated coordination service should be customizable to accommodate for changes in the street grid, new landmarks, closed roads, or similar needs due to the evolution of the City of New Irvine. The hired consultant has specified a draft of a settings file (specification of the city map) that the software system to be developed should support.

Based on the current map of the City of New Irvine, an example of such a settings file is provided below as a reference:

```
STREETS EAST-WEST:
Antbeater Road, Barracuda Pkwy, Cauliflower Ave, Doc Arthur Blvd, East Main Street,
Four Or Five Hours Freeway, Gymboree Road, Hazard Ave, Irvine Classic Drive, Jerkly
Ave, Karmans Van Ave, Lawnut Ave, Michelangelo Drive, New Irvine Blvd, Orange Hill
Road, Pale Loop, Quail Mill Pkwy, Rectangular Adobe Road, Stand Fourth, Talton Pkwy,
University Circle, Verde Palo Road, West Campus Drive, X Roads, Yoming Ave, Z End.

STREETS NORTH-SOUTH:
1st Street, 2nd Street, 3rd Street, [···street names omitted for brevity···], 42nd
Street.

LANDMARKS:
University of New Irvine (UNI) AT Stand Fourth AND 8th Street BOUNDARIES NORTH Pale
Loop EAST 19th Street SOUTH West Campus Drive WEST 8th Street,
Santa Claus Airport (SCA) AT Doc Arthur Blvd AND 12th Street BOUNDARIES NORTH Doc
Arthur Blvd EAST 24th Street SOUTH Irvine Classic Drive WEST 4th Street,
Grand Park AT New Irvine Blvd AND 27th Street BOUNDARIES NORTH East Main Street
SOUTH Talton Pkwy EAST 35th Street WEST 27th Street, New Irvine Train Station AT
X Roads AND 36th Street,
Taxi Stand A AT Doc Arthur Blvd AND 8th Street,
Taxi Stand B AT Stand Fourth AND 8th Street,
Taxi Stand C AT X Roads AND 36th Street.

PARKING:
Taxi Stand A CAPACITY 12 CABS LimoA1(3), LimoA2(3), [...], LimoA6(3), VanA1(6),
VanA2(6), [...], VanA6(6);
Taxi Stand B CAPACITY 12 CABS LimoB1(3), LimoB2(3), [...], LimoB6(3), VanB1(6),
VanB2(6), [...], VanB6(6);
Taxi Stand C CAPACITY 12 CABS LimoC1(3), LimoC2(3), [...], LimoC6(3), VanC1(6),
VanC2(6), [...], VanC6(6).
```

It is expected that the above is only an example of a settings file. In addition to the city map with street names and landmarks, additional specifications may include settings for accounting (e.g. trip charges, driver compensation), parking and cab car capacity, visualization settings, etc. Here, the City of New Irvine welcomes flexible and creative software solutions!

4.2 Taxi Management Communication Protocols

In order to ensure interchangeable software components, the City of New Irvine is aiming at standardized communication messages to be exchanged between the client and server software applications.

4.2.1 Protocol for "Call Taxi"

For calling a taxi or placing a reservation, the customer client software will communicate with the central management server by use of clear-text messages exchanged over TCP/IP protocol sockets.

The following simplified Backus-Naur Form (BNF) describes the syntax of possible customer requests for a taxi ride:

```
<request> ::= REQUEST <ride>
<ride> ::= RIDE FROM <location> TO <location> <special>*
<location> ::= CORNER <street name> AND <street name>
     | <landmark name>
<special> ::= [<time>]
     | [FOR <number> PERSONS]
     | [FIRST CLASS]
<time> ::= AT <hours>:<minutes>
     | NOW
```

The following server response then confirms the ride with its costs and estimated arrival time, decline the ride for a specified reason, or provide an error message, as follows:

```
<response> ::= OK BOOKING <code> PICKUP AT <estimated time> COSTS <amount>
     | DECLINED <reason>
     | ERROR <message>
<code> ::= <unique_reservation_code_or_number>
<amount> ::= $<dollars>.<cents>
```

The "Call Taxi" protocol may be extended as needed during software development.

4.2.2 Protocol for "Taxi Cab"

For the central server to stay in contact with the taxi cabs in the city, the server software will communicate with the taxis by use of short text messages exchanged over TCP/IP protocol sockets.

The following simplified Backus-Naur Form (BNF) describes the syntax of server requests and instructions sent to a taxi cab:

```
<message> ::= <request>
     | <booking>
     | <instruction>
<request> ::= REQUEST <taxi> POSITION
<booking> ::= BOOKING <taxi> <code> <ride>
<instruction> ::= GO <taxi> <direction> <direction>*
     | PICKUP <taxi> <code> <number> PERSONS
     | DROPOFF <taxi> <code> <number> PERSONS
<direction> ::= NORTH | EAST | SOUTH | WEST
<taxi> ::= <type><origin><number>
<type> ::= "Van" | "Limo"
<origin> ::= "A" | "B" | "C" | ...  | "Z"
<number> ::= ["1"| ...  |"9"]["0"| ...  |"9"]*
```

In response to a position request, a taxi cab will immediately answer with its current location. In response to instructions, the cab will confirm the receipt with an OK message or respond with an error message and corresponding reason.

```
<response> ::= OK <taxi> POSITION AT <location>
     | OK <taxi> BOOKING <code>
     | OK <taxi> <driving> <driving>*
     | OK <taxi> PICKUP <code>
     | OK <taxi> DROPOFF <code>
     | ERROR POSITION <message>
```

```
| ERROR BOOKING <message>
| ERROR DIRECTIONS <message>
| ERROR PICKUP <message>
| ERROR DROPOFF <message>
| ERROR <message>
```

The "Taxi Cab" protocol may be extended as needed during software development.

4.3 TCP/IP Communication via Sockets

Please refer to lecture 8 for a basic introduction to socket programming.
Online documentation is available at

- `http://www.linuxhowtos.org/C_C++/socket.htm`
- `http://www.linuxhowtos.org/data/6/client.c`
- `http://www.linuxhowtos.org/data/6/server.c`

An extended socket tutorial is available on our server at the location
`~eecs22/SocketTutorial.tar.gz`

- client2.c
- server2.c
- Makefile
- README

Handling of blocking I/O and multiple connections at the same time is discussed in lecture 9.
An extended clock server example is available on our server at the location
`~eecs22/ClockServer.tar.gz`

- ClockClient.c
- ClockServer.c
- Makefile
- README

4.4 Time

To consider travel time for taxis on the streets, assume that taxis take unit time (1 minute) to cover per block step. Additionally, 1 minute of real time is simulated as 1 second on screen, i.e.

$$\text{real time} = \text{simulated time} \times 60$$

Acknowledgements: The Taxi Cab project idea originated in a discussion with Tim Schmidt in December 2013. The EECS 22L team thanks Tim for his contribution to this project.