Güber

S.T.A.C.K. LABS

Güber (Version 0.1)
Software Specification

William Scotten
Brian Truong
Desmond Chan
Hikaru Kasai
Jay Patel

The Henry Samueli School of Engineering
University of California, Irvine

Table of Contents

Glossary

API - application program interface

Outlines the set of protocols and explains how software components interact. APIs provide the necessary tools and building blocks for a developer to work with the provided software components.

GUI - graphical user interface

A type of user interface that lets user interact with electronic devices through graphical icons, imagery and text.

Data type

A classification of data which tells the compiler or interpreter how the programmer intends to use the data.

Data Structure

A specialized format for organizing and storing data.

Syntax

The input method, rules and symbols that define programming language.

Software Architecture Overview

Main Data Types and Structures

```
char *map[ ]

typedef struct TaxiStandList
{
        struct TaxiStandEntry *first;
        struct TaxiStandEntry *last;
        int totalTaxis;
        int totalStands;
}TaxiStandList;

typedef struct TaxiStandEntry
{
        struct TaxiStandList *list;
        struct TaxiStandEntry *next;
        struct TaxiStand *stand;
}TaxiStandEntry;

typedef struct Taxi
{
        int passenger;
        int rideLength;
        double fare;
        double expense;
        int type;
        int taxiID;
        int location;
        int pickup;
        int destination;
}Taxi;
```

```
typedef struct TaxiStand
{
        int location;
        int numTaxis;
}TaxiStand;

typedef struct map_obj
{
        char *arr;
        int x;
        int y;
        int num_taxis;
} Map;
```

The Map

*char *map[ ]*

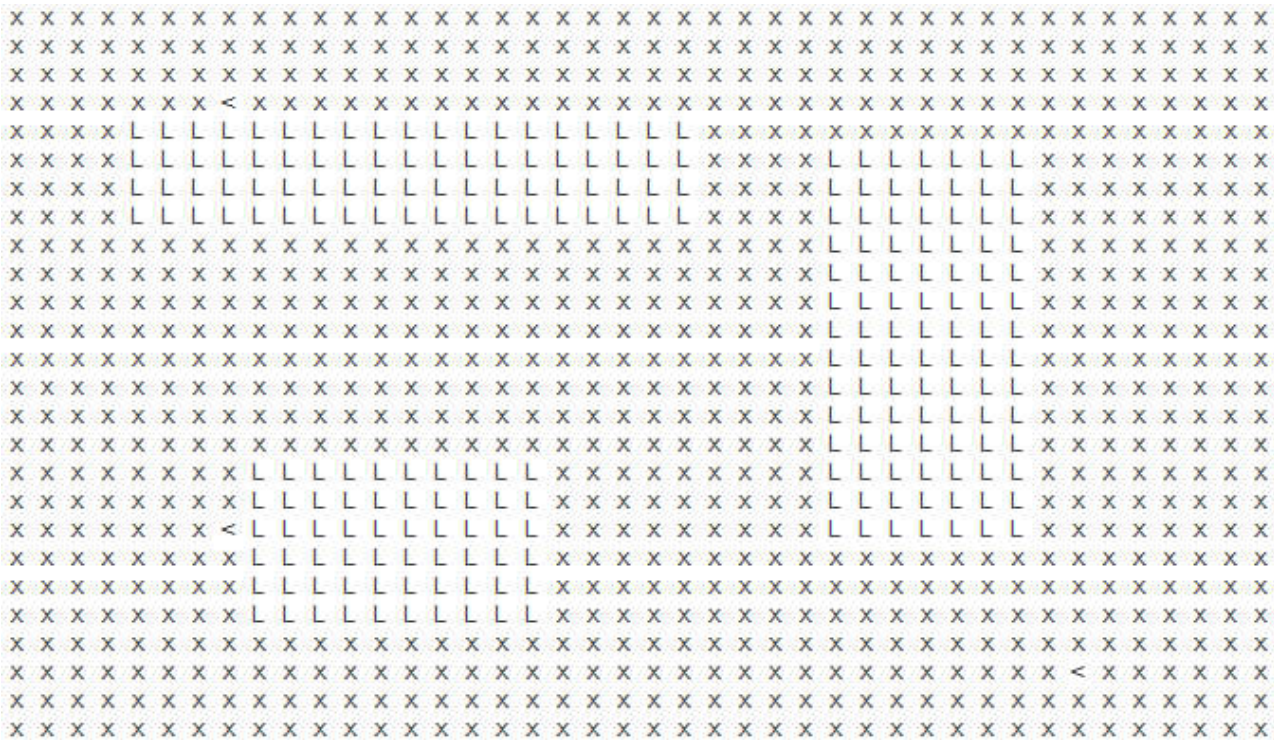
1 dimensional array of characters

Index 0 is at the top left, and final index is bottom right

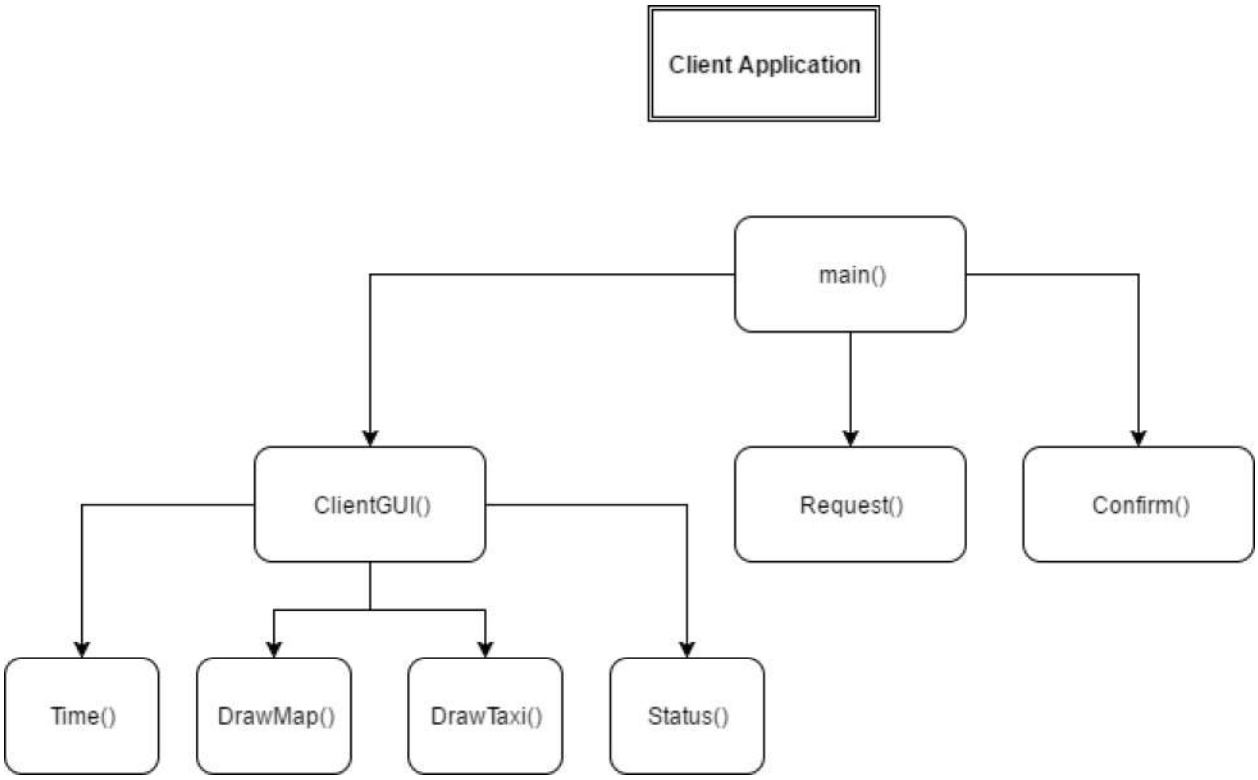
Example:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

Coordinates that are roads are marked with X's on the map, landmarks that aren't traversable are marked with L's, and coordinates that have taxis on them have the number of taxi's on that exact spot. In the example below, the initialized taxi stands have 12 taxis in each spot, which is 3 ASCII characters above the number 9, which is the symbol "<".



Major Software Components



Major Software modules:

GuberClient - Executable taxi Application designed for the ease of use for the clients. This module also holds the communication protocols and manages communications with the server(host) and sends requests accordingly and receive commands to process the requests made by users. The program projects the taxis available and the map of the city(New Irvine) on a window that users can interact with.

GuberServer - Executable taxi Application designed for the host and the developer to access smooth flow of the taxi Application. This module process the requests received from clients via communications protocols and accepts valid clients request for connections to establish a running taxi software.

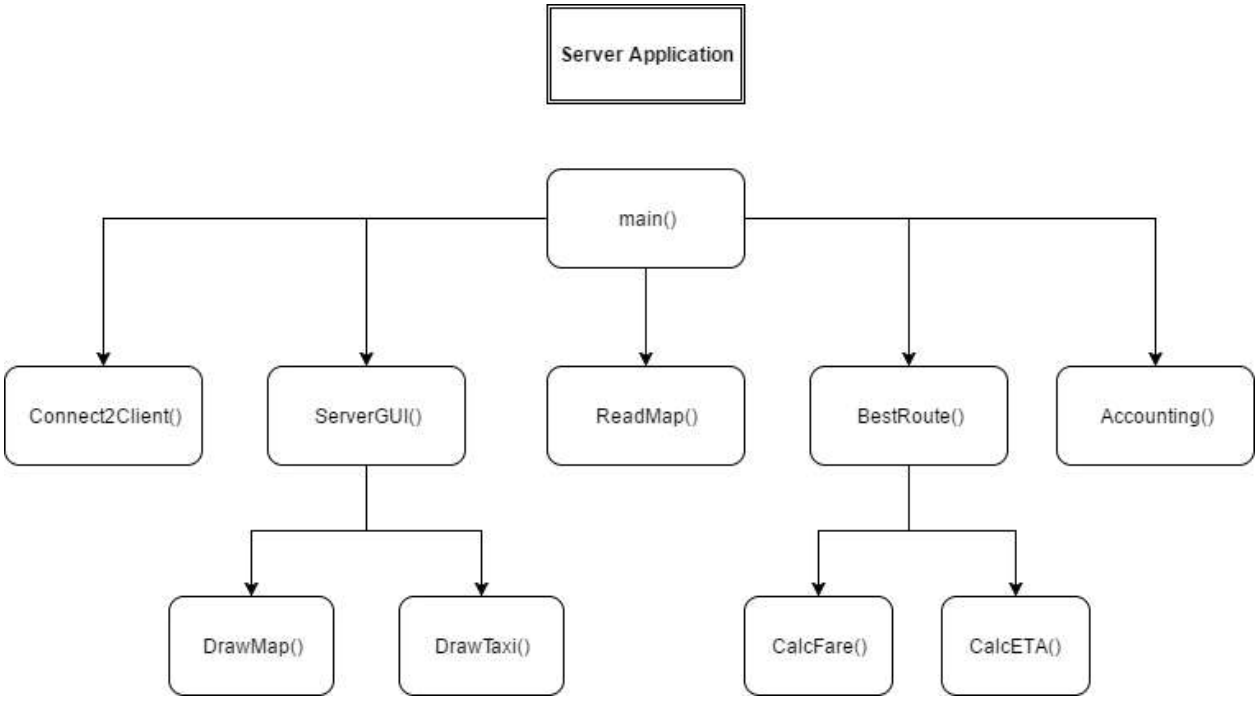
Reader - This module is responsible of reading the textual data of the map of a city and convert the data into coordinates used to make a map that is shown in the client window.

Taxi - The taxi module holds function definitions that create memory space for all basic blocks of the application: taxi, taxi fleet and taxi stand. It creates a double linked list that regulates the number of taxis occupied and available at various taxi stands. It also contains functions that free up the memory space allocated to taxi, taxi fleet and taxi stand.

DrawMap - This module creates the map on the Interface with landmarks and taxis.It primarily read through the an array that contains the street names and print them on the screen.

DrawTaxi- This module draw taxis on the map according to their locations and corresponding ID. The function definitions in the module uses cairo and gtk libraries to draw different graphics on the user and host screen.

Makefile - Collection of all commands for compiling the listed above modules and link all of the dependencies and targets to make a single executable file for the user.



Module Interfaces
API of Major Module Functions

Modules declared in header files:

Reader.h	**Taxi.h**	**Guber.h**	**GuberClient.h**	**Belle.h**
ReadMap()	DeleteTaxiStand()	main()	enter_callback()	searchword()
DrawMap()	MakeTaxiStandList()	Connect2Client()	FatalError()	Belle()
DrawTaxis()	AddTaxiStandEntry()	ServerGui()	Talk2Server()	
DrawObjects()	MakeTaxiFleet()		GetTimeFromServer()	
	FreeTaxiFleet		ShutdownServer()	
	UpdateTaxi()		getServerClock()	
	SetLoction()			
	SetPickup()			
	SetDestination()			

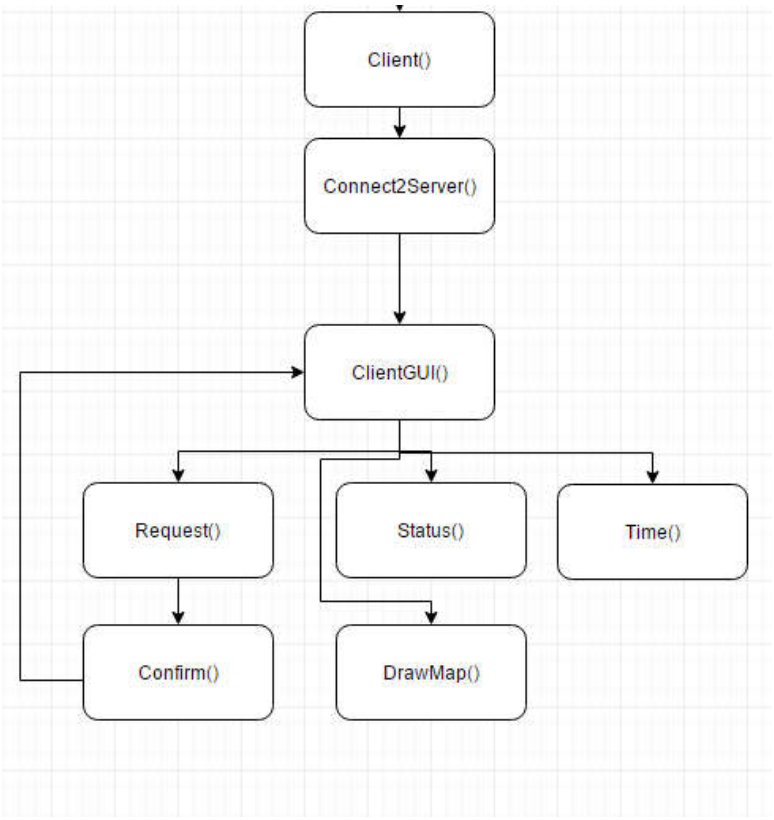
Header Files included in C files:

Reader.c	Taxi.c	GuberServer.c	GuberClient.c
Reader.h	Taxi.h	Taxi.h	GuberClient.h
	Guber.h	Calculate.h	Calculate.h
	Reader.h	Reader.h	Reader.h
		GuberClient.h	GuberClient.h
		Guber.h	Taxi.h

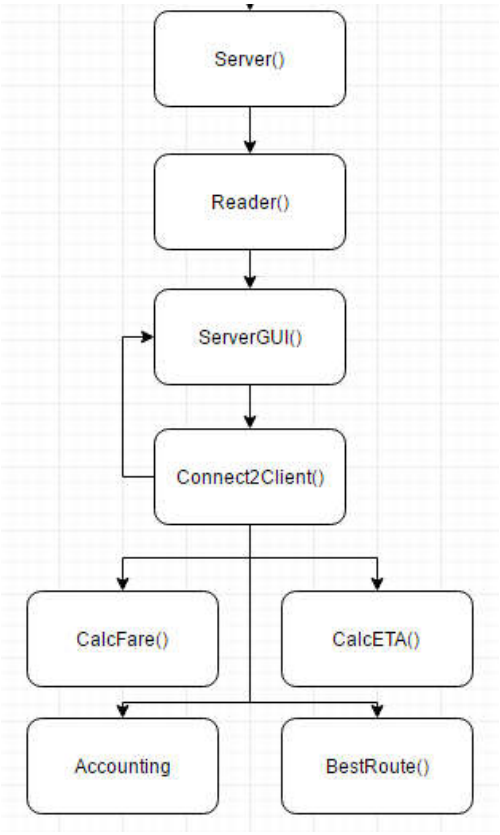
		Belle.h	Belle.h

Overall Program Control Flow

Client Side



Server Side


Server()
Reader()
ServerGUI()
Connect2Client()
CalcFare()
CalcETA()
Accounting
BestRoute()

Installation

System Requirements and Compatibility

Minimum Requirements:

Operating System	Linux
CPU	Intel Pentium or better
GPU	Integrated Graphics Card or better
RAM	1 GB
HDD	1 GB
Software:	SSH client with X Forwarding

Setup and Configuration

1. You must first connect to the team repository located on the UCI EECS server. After logging into the personal EECS account, execute the following in the terminal:
    a. `setenv CVSROOT :ext:team5@zuma.eecs.uci.edu:/users/ugrad2/2017/winter/team5/cvsroot`
    b. `setenv CVS_RSH ssh`
    c. `setenv CVSUMASK 007`
2.
    a. First time folder setup:
        i. `mkdir project`
        ii. `mkdir project/taxi`
        iii. `cd project/taxi`
    b. Once the local folder has been created, obtain a copy of the files:
        i. `cvs checkout -d chkout project/taxi`
    c. The developer's local folder *taxi* should now have a copy of the source files.

Building, Compilation, and Installation

1. Extract the source file:
    a. `gtar xvzf Taxi_Beta_src.tar.gz`
2. The software documentation can be read by executing:
    a. `evince taxi/doc/Taxi_SoftwareSpec.pdf`
3. Go to the taxi directory and build the executable:
    a. `cd taxi`
    b. `make`
4. Run the programs
    a. `./GuberClient <host name> <port number>`
    b. `./GuberServer <port number>`
5. To delete any non-source files:
    a. `make clean`

Documentation of Packages, Modules, Interfaces

Detailed description of data structures

The main data structure for this program is the taxi struct. It contains a variety of data required by the server to effectively direct taxis around a city. First a taxi knows how many passengers it has. It also should be able to calculate how long a ride will take and how much it will cost. Taking this into consideration, the taxi will be able to provide a price to the user. There will be two types of a taxis, a personal taxi and a shuttle type. This affects the routes and rates. Each taxi will also know it's current location as well as its pickup and drop off location.

A linked list of Taxi Stands is generated to properly create the taxi fleet. There is a TaxiStandsList which contains a pointer to a series of TaxiStandEntrys. The TaxiStandList also contains how many taxis are within the entire list and how many stands are in total. Each TaxiStandEntry contains a pointer to its parent list, the next TaxiStandEntry and its own TaxiStand struct. Each TaxiStand knows the location of that specific struct and its max number of taxis.

Detailed description of functions and parameters

Functions for Client Application

*int DrawMap(int rowNum, int colNum, char *rowNames[], char *colNames[ ], Objects *objects)*

This function draws the city map using GTK and Cairo. The streets will be represented by vertical and horizontal lines, which will be determined by rowNum and colNum. The parameter rowNum is the number of streets to draw in the horizontal direction. Likewise, colNum is the number of vertical lines to draw. For example, if rowNum = 4 and colNum = 7, then there will be 4 horizontal lines and 7 vertical lines intersecting to form the streets. The names of the streets are passed through char *rowNames[] and char *colNames. Each street name will be drawn next to their respective street lines. Object *objects is a parameter referring to the Object struct that contains the landmarks, taxi stations, etc to

draw. The struct will be referenced to draw the rectangular objects on the map by accessing their widths, height, and coordinates on the map.

*int ClientGui(int argc, char *argv[ ])*

This function creates the graphical user interface for the client application. The city map and the taxi that the user has requested will be drawn. The window will also have a field box to enter the pickup location, destination, service type, etc. Other useful information such as the confirmation status, estimated far, and arrival time will be shown in a user friendly manner. The two parameters are for user text input. Int argc and char *argv[ ] will be used to scan user input and send the addresses to the server.

void Time()

This displays the time obtained from the server to the client window.

*void DrawTaxi(Taxi *taxi)*

DrawTaxi() draws the user's requested taxi on the map. It shows the real time location of the taxi. The struct taxi will be taken as an argument to access its location. An image of a simple taxi will be drawn onto the map.

*void Status(Taxi *taxi)*

Status() displays the ride information to the user. The estimated arrival time (ETA), fare, and confirmation of the request will be shown on the window. The information needed for these are all contained within the taxi struct.

*char Belle(char *input)*

This function is our AI assistant which communicates with the client. It breaks down the sentence inputted by the user and get the necessary information from it like pick up, destination or confirm, then calls the functions which calculate price and ETA. The function will then return back a sentence answering what the user is asking for like a human interaction/ live customer service.

*Taxi *Confirm(char *ServResponse)*

This function will decompose the string sent by the server to the client by accessing *ServResponse. The formatted message will be broken down and data will be stored in the taxi struct. The taxi struct will then be returned.

Functions for Server Application

Connect2Client()

This function is used by the server to establish a connection with a potential client. It checks whether the clients are connected on the same ports used by the host.

Char *Reader()

This function scans a user provided .map text file for the relevant map information. It constructs and returns a 1 d array filled with various chars such as L and X to indicate traversable points and impassable landmarks.

Char *BestRoute(Taxi *taxi)

This function calculates the optimal route for the taxi to take. It will route the shortest and most efficient path. It stores the the optimum route as a series of directions in a string.

*Taxi CalcFare(Taxi *taxi)*

This function will calculate the estimated fare cost given the pickoff and destination locations. The car or service type will also be considered. The fare will include the base price and a fee per distance price. It will store the value in dollars to the taxi struct as Fare.

*Taxi CalcETA(Taxi *taxi)*

The estimated time arrival of the client to their destination will be calculated. The pickup location and destination will be accessed to calculate the time it will take for the taxi to reach the destination given a fixed speed limit. The ETA will be stored in the taxi struct as RideLength.

*double Accounting(Taxi *taxi)*

The company's revenue and expenses will be calculated in this function. The expenses to pay the taxi drivers will be subtracted from revenue made by all

fares paid by the client. The fare and expenses of each taxi will be accessed for all taxis and the total will be returned as a double in dollars.

DeleteTaxiStand()

This function frees the memory allocated to creating a taxistand struct.

*Static TaxiStand *DeleteTaxiStandEntry(TaxiStandEntry *entry);*

This function frees the memory allocated to creating a TaxiStandEntry struct and frees appropriate pointers.

*TaxiStandList *MakeTaxiStandList()*

This function allocates memory to creating a TaxiStandList and creates a single empty TaxiStandList.

*void AddTaxiStandEntry(TaxiStandList *list, int location, int num);*

This function allocates memory to creating a TaxiStandEntry. This creates an entry with the provided location and number of taxis and appends it to the list sent in.

*TaxiStand *MakeTaxiStand(int location, int num);*

This function allocates memory to creating a single TaxiStand struct. The struct has values equal to the given parameters.

*Taxi *MakeTaxiFleet(TaxiStandList *list);*

This function creates an array of taxi structs. It uses the information in the TaxiStandLIst to make the appropriate number of taxis with the correct locations.

*void FreeTaxiFleet(Taxi *taxiFleet);*

This function frees the memory allocated to the taxi fleet.

*void SetLocation(Taxi *fleet, int index, int location);*

This function sets the location member for the taxi at the given index in the taxi fleet.

*void SetPickup(Taxi *fleet, int index, int pickup);*

This function sets the location pickup for the taxi at the given index in the taxi fleet.

*void SetDestination(Taxi *fleet, int index, int destination);*

This function sets the destination member for the taxi at the given index in the taxi fleet.

void UpdateTaxi();

This function is periodically checked to update each taxi's location. If a set time has passed then the taxi's location is updated to the next step in it's route.

*int ServerGui(int argc, char *argv[ ])*

This function creates the graphical user interface for the server application. The city map and all taxis will be drawn. There will be a button to shutdown the server. The time will also be displayed.

*int DrawMap(int rowNum, int colNum, char *rowNames[ ], char *colNames[ ], Objects *objects)*

Same as the function used for the client application.

*void DrawTaxi(Taxi *taxi)*

This function will be called for all taxis for the server map. Thus, every taxi will be drawn for the server application map.

Int DrawObject(Object *object)

This function will be called by all objects for the server map. Every object will be drawn for the server application map.

Detailed Description of Routing and Scheduling

Description of Routing Algorithm

The routing will take move east or west along the city until it reaches a landmark or reaches the same column as the destination. If a landmark is reached before reaching the destination, the taxi will move north or south until a legal move east or west towards the destination is possible. After dropping of the passengers at the desired destination the taxi will head back towards a taxi stand or service another request.

Description of Scheduling and Communication

The server will create a list of all requests and add future request at the end of the list. It scans through the list and selects the first request that can be serviced reasonably. It then looks for the closest free taxi on the road or taxi at a taxi stand and sends it to the pickup.

- Client Request

```
<request>  ::= REQUEST_TAXI <location> TO <location> <special>*
             |   REQUEST_POSITION <taxi>
             |   CONFIRM <reservation>
             |   CANCEL <reservation>
<location>::= <pos>
             |   CORNER <street_name> AND <street_name>
             |   <landmark_name>
<special>  ::= [ASAP]
             |   [AT <time>]
             |   [FOR <number_persons>]
             |   [NON_STOP]
<time>     ::= <hours>:<minutes>
```

- Server Response

```
<response>::= OK <taxi> PICKUP <pos> [<time>] DROPOFF <pos> [<time>]
                    $<amount> CONFIRM <reservation>
             |   DECLINED <reason>
             |   OK <taxi> POSITION <position> [ETA <position> <time>]
             |   INVALID <reservation>
             |   ERROR <message>
<amount>   ::= <dollars>.<cents>
```

Testing Plan

Unit test of module Reader.c

1. ReadMap()
    a. .map file with sample streets and landmark
    b. The names will be ordered in the one dimensional array to use as a parameter
2. DrawMap()
    a. .map textfile
    b. Print out of the map in a text format.
3. DrawTaxi()
    a. Taxi struct
    b. Number of taxis
4. DrawObjects()
    a. Object struct
    b. Number of object

Unit test of module Calculate.c

1. CalcFare()
    a. Pickup and drop off location for the trip
    b. A double value of the price for the
2. CalcETA()
    a. Pickup and drop off location for the trip
    b. A double value for the expected time of the entire trip
3. Accounting()
    a. Taxi
    b. Updated total balance of the current running taxi service
4. BestRoute()
    a. Pickup and drop off location
    b. Creates a string containing directions to the drop off location.

Unit test of module Server.c

1. Connect2Client()
    a. none
    b. Print statement confirming connection
2. ServerGUI()
    a. Map array and taxi object
    b. Window displaying the guber client to the user with the city map and current taxi location

Unit test of module Client.c

1. Request()
    a. Pickup and Dropoff location
    b. Printout of Pickup and Dropoff location and string containing desired locations.
2. Confirm()
    a. Message from server indicating confirmation of taxi request
    b. Confirmation that requested taxi is available, not available, or error.
3. ClientGUI()
    a. Time()
        i. Time as a string input
        ii. Graphical display of the time
    b. DrawMap()
        i. Array of street names and objects (few to test with)
        ii. A fully drawn map with color

    c. DrawTaxi()
        i. A sample taxi struct. The location, carID and type will be initialized.
        ii. A single taxi will be drawn (an image) with its current location. The location will update to provide a smooth rendering of the taxi moving along the city.
4. Connect2Server()
    a. Port number
    b. Print statement confirming connection
5. Time()
    a. None
    b. Returns time since the server has started running

6. Status()
    a. Taxi object
    b. Provides current information of desired taxi

Development Plan and Timeline

Partitioning of Tasks

Hikaru - Submit weekly deliverables, GUI
William - Build Calculate.c
Brian - Work on taxi request assignment
Desmond - TCP/IO, GUI
Jay - TCP/IO, GUI

Team Member Responsibilities

Hikaru - Submit weekly deliverables, GUI
William - Build Calculate.c
Brian - Work on taxi request assignment
Desmond - TCP/IO, GUI
Jay - TCP/IO

Copyright

References

Beal, Vangie. "API - Application Program Interface." *What Is API - Application Program Interface? Webopedia*. QuinStreet Inc., n.d. Web. 22 Jan. 2017.

"How to Write a Copyright Notice." *PlagiarismToday*. Jonathan Bailey, 08 Nov. 2011. Web. 22 Jan. 2017.

Index