Flashy Card

COMP 8045 – Major Project 1

Colin Bose – A00900656

5-1-2018

Contents

1 Introduction

1.1 Student Background

I entered the Computer Systems Technology (CST) program in 2014. In the second year I chose to take the Data Communications option. The experiences in Data Communications lead me to enroll in the Network Security Bachelor of Technology program upon completing the CST diploma.

1.1.1 Education

(BCIT) Bachelor of Technology in Computer Systems 2016 – 2018

- Specialized in Network Security

- Maintains honors status (80% +)

(BCIT) Computer Systems Technology (CST) Diploma 2014 – 2016

- Specialized in Data Communications

- Graduated with honors (80% +)

1.2 Project Description

This project is a half practicum and I worked alone.

The goal of this project is to create a language learning application that allows users to study a variety of languages. The application delivers study material through flash cards and allows the user to study alone or with a group.

1.2.1 Essential Problems

The problem this project tries to solve is a problem that many existing applications are already attempting to address – the best way to teach language through flash cards. The first way of addressing this problem is the delivery and evaluation methods used in the study process.

The second essential problem is the multiplayer aspect of the application. At the application level, the problems to be solved are how to best group users together so that everyone is studying similar cards, as well as how to provide a smooth study experience without excessive waiting on other users. At the technical level, the multiplayer study means providing a reliable and scalable server to handle all the users.

The third essential problem is how to test and evaluate the performance of the previously mentioned server. This means developing an application to simulate large amounts of user traffic to stress test the server, as well as implementing ways to measure performance on the server itself.

1.2.2 Goals and Objectives

The main goal of this project is to create a flash card based language learning application that offers superior features to those currently available. The client side of the application will serve as a front for the more relevant server and communications parts of the project while still providing a number of interesting and challenging problems itself.

The goal for the server part of the project is to develop a framework that could easily be adapted to handle any type of client/server application. An aspect of this is the ability to handle a large concurrent user base, as well as handle basic security concerns such as encryption and user data.

The final goal for the project is to develop a tool for testing the scalability of the server. This tool, along with measurements collected on the server during use, will give an idea of the performance capabilities of the server as well as providing information on how to better optimize the server.

2 Body

2.1 Background

The problem that FlashyCard hopes to solve is the task of learning a new language. For users who wish to learn new vocabulary, or practice grammar and sentence structure, it will provide a way to do so.

The concept of flash card based language learning applications is not a new one and there are a number of similar applications currently on the market. Each of these applications (Anki, Quizlet, Cram, Flashcard+, StudyBlue, etc.) provides a slightly different experience with differing strengths and weaknesses.

The basic idea behind the available flashcard applications is simple. A user has a deck of cards from which they study. These cards consistent of a “front” and “back”, which in language learning is generally a word or phase in the user’s native language as the front and the second language equivalence on the back.

The flashcard applications main purpose is to manage which cards are shown to the user each day and to provide a way for the user to practice or demonstrate their knowledge. Typically, the front side of the card is shown to the user and they are prompted to enter the answer from the back side into some input field. The method of input differs, with some applications taking text input, some providing a list of options to choose from, and some presenting words as part of a game.

The main feature that none of the current applications offer, and that will differentiate FlashyCard, is the idea of multiple user study sessions. The idea behind the multiplayer study groups is to allow users to review each card together, providing motivation through friendly competition.

2.2 Project Statement

This project will aim to answer the question of what makes a good language learning application. By attempting to improve and innovate on the current methods available in other flashcard applications, FlashyCard will hopefully provide a superior language learning format.

The second consideration is networked studying, both on how to implement it and what benefits it provides users.

2.3 Possible Alternate Solutions

There are a number of possible alternate solutions available in terms of language learning applications. The first solution is to move away from flashcards entirely and to provide structured lessons and examples, as might be found in a textbook.

The device on which the application is used is another major consideration, and provides different solutions. The most common device for current flashcard applications is the phone, as it provides a way to quickly pull up and review cards any time throughout the day.

Another alternate solution to the whole application is to drop the multiplayer aspect of the project. It's not something provided by any other application, and perhaps there is a reason why.

The final considerations for alternate solutions are technical – from which language to use to which database, etc.

2.4 Chosen Solution

After considering the alternative solutions, the final form of FlashyCard is set as follows:

- Flashcard based application
- Desktop application
- Networked study supported
- C++ development using the Qt library, SQLite database

The decision to go with a flashcard based applications is fairly simple. First, it provides a much more general solution than applications built specifically for one language. Second, it removes the burden of study material creation from the developer and puts it on the user.

The decision to go with a desktop application was the most difficult. The ability to pull out a phone and study anywhere is certainly a huge advantage, and is difficult to pass up. However, a desktop application does have its strengths. The first and most important is the keyboard. Many mobile flashcard applications offer limited input options for proving ones knowledge. An application like Anki only offers the user options on their feelings of the current card, such as 'Easy', 'Hard', etc. Other mobile applications use multiple choice buttons or matching games as input. Because FlashyCard uses text input to better evaluate the users level of knowledge of a given word or sentence, having a full keyboard is a big strength for a desktop application. This is especially relevant with a focus on studying sentences, rather than just vocabulary terms.

A second consideration for desktop over mobile is the idea of long term, large scale learning. The progression system of cards in FlashyCard is such that within a month or two, users will be looking at over 100 cards due daily. With text input, this becomes more of a sit down, long term task than something to do quickly during a break on a phone.

Incorporating a networked study component is what ties the whole project back into the knowledge that I've learned during COMP 7005 and COMP 8005. The general communications between client and server builds on basic network concepts learning in 7005, and the design of the server builds directly from the major scalable server project in 8005. The inclusion of a networked component and user communications and registration also brings up security issues that tie back into the security side of my learning.

Besides being required to make the project relevant to my field of study, the benefits of multiple user study seem to be worth its inclusion. Whether its extra motivation from competing to complete cards with others, or the opportunity to converse with others in the practiced language, there should be reasons for users to choose multi-user study over studying solo.

On the technical side, C++ is the language I am most comfortable with so it's what I selected. For the database side of the project, a database that requires no setup from the user was required. SQLite provides this, and so it was selected.

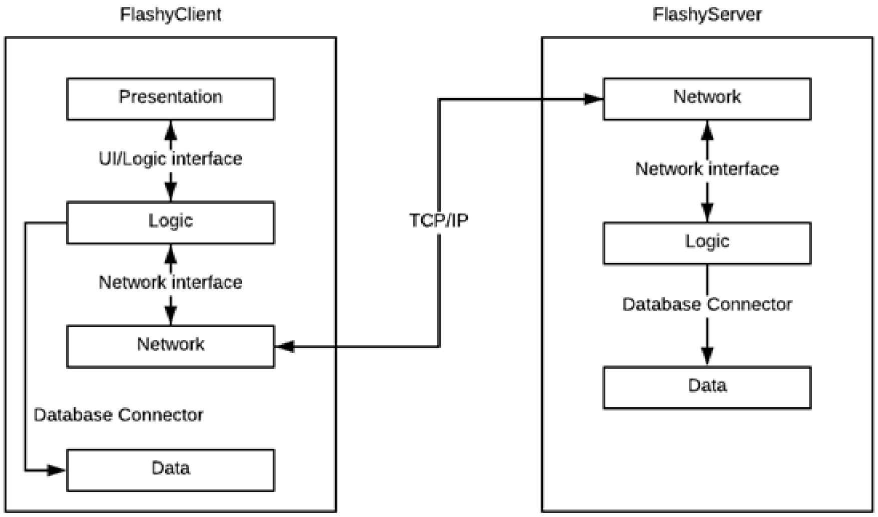
2.5 Details of Design and Development

2.5.1 Deliverables

- FlashyClient: The client side of the project – a desktop executable that a user would run.
- FlashyServer: The server side of the project – run on a server somewhere
- FlashyTester: The traffic simulator for load testing the server
- Final Report: Final report for COMP 8045

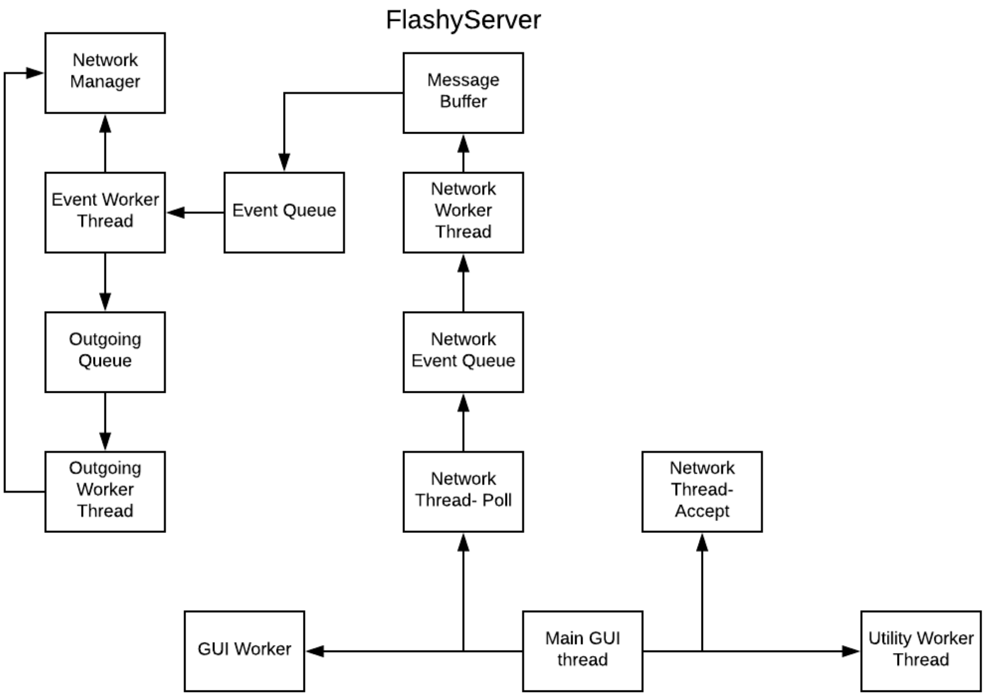
2.5.2 System Diagram

The following is the high level system diagram showing the client and server.



2.5.3 Server Architecture Diagram

The following describes the general architecture of the server.



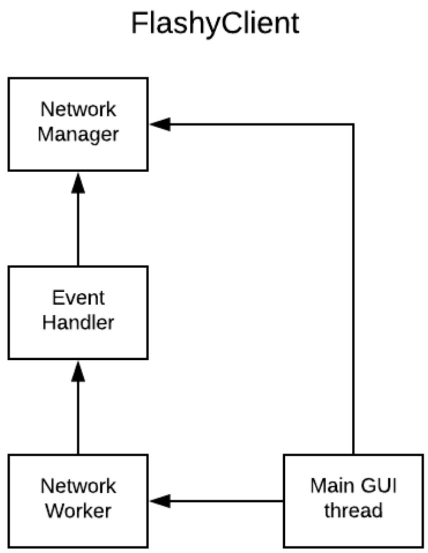
A description of the basic workings of the server and its components is as follows:

- Main GUI Thread: The main thread for the program. According to QT design rules, this thread must handle all GUI updates. Setting the server online starts a number of worker threads as described below.
- Network Thread – Accept: Handles all new incoming connections, establishing the connection and registering the new socket with the epoll structure.
- Network Thread – Poll: The main epoll loop which handles changes in socket descriptors. Generally this is an alert that data is ready to be read on some socket. Creates new network events and pushes them onto the Network Event Queue.
- Network Worker Thread: Pulls Network Events from the front of the Network Event Queue and deals with them. This means reading any available data and pushing it into the Message Buffer.
- Event Worker Thread: Pulls from the Event Queue and handles the event. These events are updates to the overall system such as a user joining a study room, answering a card, or making a guess in the multiplayer review game. Depending on the event a response may be sent directly through the Network Manager, or set as a delayed response and pushed onto the Outgoing Queue.

- Outgoing Worker Thread: Polls the Outgoing Queue – these are delayed events such as a round timeout for the current study round.
- Utility Worker Thread: Several Utility Worker Threads are created on server startup, but remain idle until needed. As server load increases and the current worker threads are unable to keep up, utility workers are woken up and set as either Event or Network workers. These threads also react to changing loads on the system. If the network queue begins to fall behind, a utility thread may switch from handling the event queue to start handling network events.
- GUI Worker: Works in 1 second ticks to collect server data from the past second including latency in network and event processing, packets/s, current users, etc. Pushes these updates to the main GUI thread.
- Message Buffer: Due to the way epoll and non-blocking sockets work, reading a socket reads until there is no data left. This does not guarantee that an entire message was read, and so all incoming data is pushed onto the message buffer at the corresponding index for the socket it was read from. The message buffer deals with splitting up this data into individual messages and pushing those messages onto the Event Queue.
- Queues: Network, Event and Outgoing – each holds some required data and serves to pass concerns from one part of the program to another - the network thread does not have to deal with event logic, for example. Measuring delay between pushing an event onto the queue and event completion gives a good measure of current resource requirements and allows utility worker threads to swap to whichever part of the system is struggling.

2.5.4 Client Architecture Diagram

The following describes the general client architecture

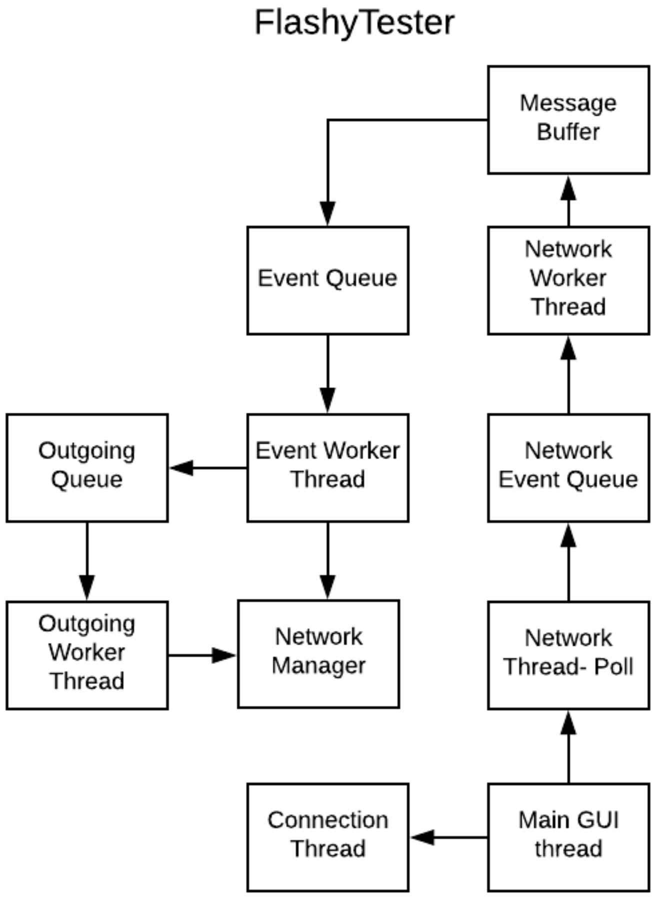


A description of the basic workings of the client and its components is as follows:

- Main GUI Thread: Main thread of the program that handles the various user inputs such as switching menus, inputting study answers, etc.
- Network Worker: Reads for data from the server and passes it to the event handler.
- Event Handler: Handles the logic of the networked study system, parsing messages such as which card to display next, new users joined, private messages, etc. If a response is required it is sent through to the Network Manager.
- Network Manager: Handles data communications between the client and server.

2.5.5 Tester Architecture Diagram

The following describes the general test program architecture



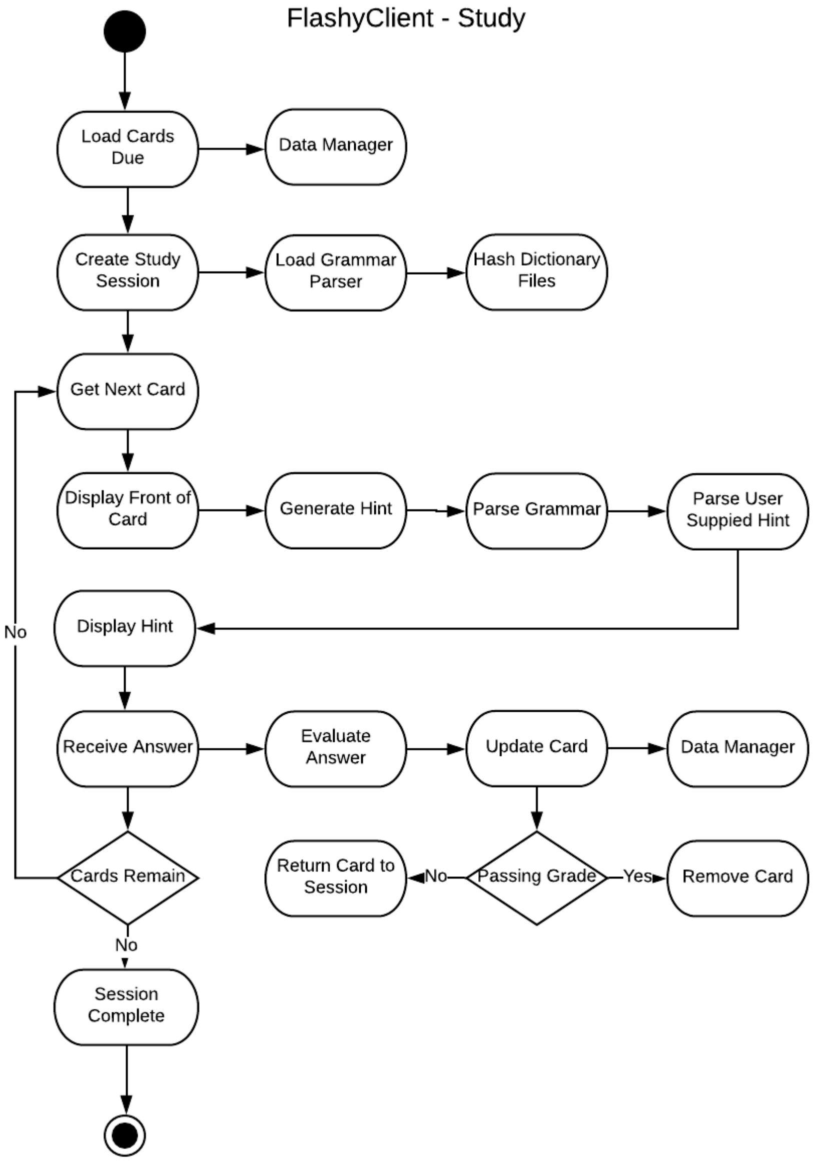
A description of the basic workings of the test program and its components is as follows:

- Main GUI Thread: The main thread for the program. Starting the testing process spawns the Network Thread – Poll and Connection Threads.
- Connection Thread: Handles the creation and connection of the specified number of clients. Each connected client is first added to the epoll structure, and then sends off an initial request for available study rooms.
- Network Thread – Poll: The main epoll loop which handles changes in socket descriptors. Generally this is an alert that data is ready to be read on some socket. Creates new network events and pushes them onto the Network Event Queue.

- Network Worker Thread: Pulls Network Events from the front of the Network Event Queue and deals with them – reads any available data. Data read is pushed into the Message Buffer.
- Event Worker Thread: A stripped down version of the event handler from the FlashyClient. This thread handles incoming events with limited functionality. If immediate response is required it responds through the Network Manager. The more common situation is to push an event onto the Outgoing Queue.
- Outgoing Worker Thread: Polls the Outgoing Queue – these events are all responses to the current study round. To more accurately mimic real user study patterns, a delay is randomized between receiving the current study card and sending an answer. This avoids each room running through 1000's of questions per second.
- Message Buffer: As with the server, the message buffer is used to manage fully complete messages. Once a full message is read, it is pushed onto the Event Queue.
- Queues: Network, Event and Outgoing – each holds some required data and serves to pass concerns from one part of the program to another.

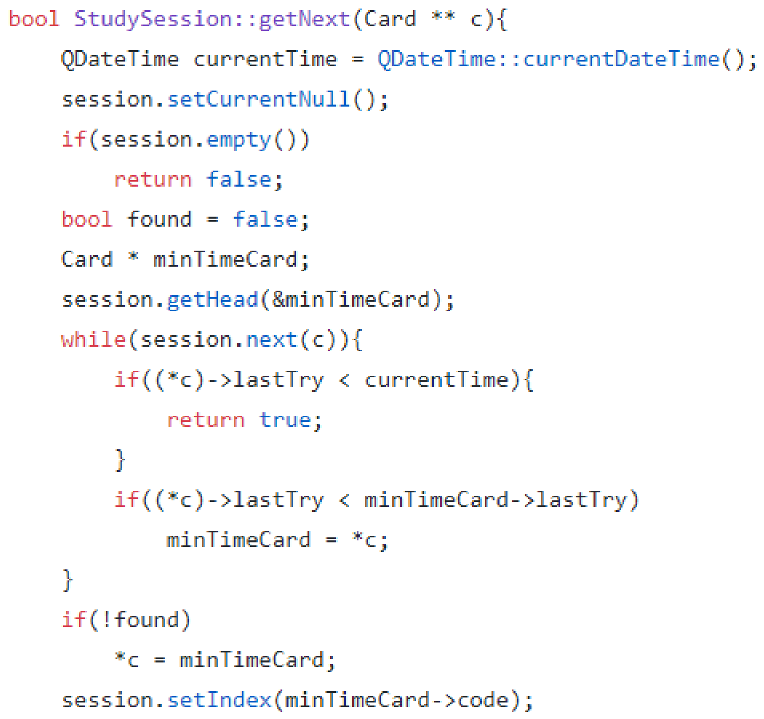
2.5.6 FlashyClient Study Implementation

The following shows the design for the main client side use case – studying



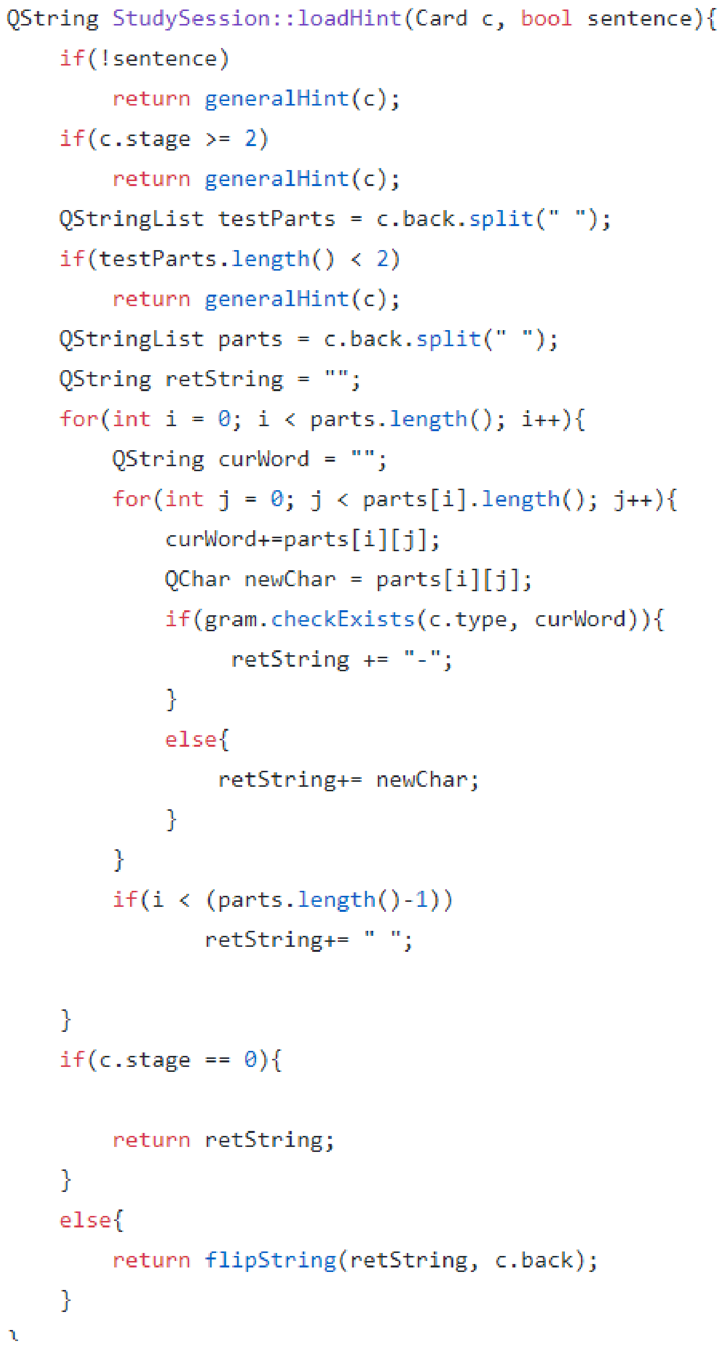
The overall flow for the Study use case is fairly straight forward. In the simplest terms, a user is shown a card and inputs an answer. If the answer meets a level of correctness, the card is removed and not shown again for some number of days. If not, it is shown again until they achieve the required correctness.

The following snippet shows the ‘Get Next Card’ function of the study use case.



```
bool StudySession::getNext(Card ** c){
    QDateTime currentTime = QDateTime::currentDateTime();
    session.setCurrentNull();
    if(session.empty())
        return false;
    bool found = false;
    Card * minTimeCard;
    session.getHead(&minTimeCard);
    while(session.next(c)){
        if((*c)->lastTry < currentTime){
            return true;
        }
        if((*c)->lastTry < minTimeCard->lastTry)
            minTimeCard = *c;
    }
    if(!found)
        *c = minTimeCard;
    session.setIndex(minTimeCard->code);
```

Following setting the current card, the hint is generated through the following code.

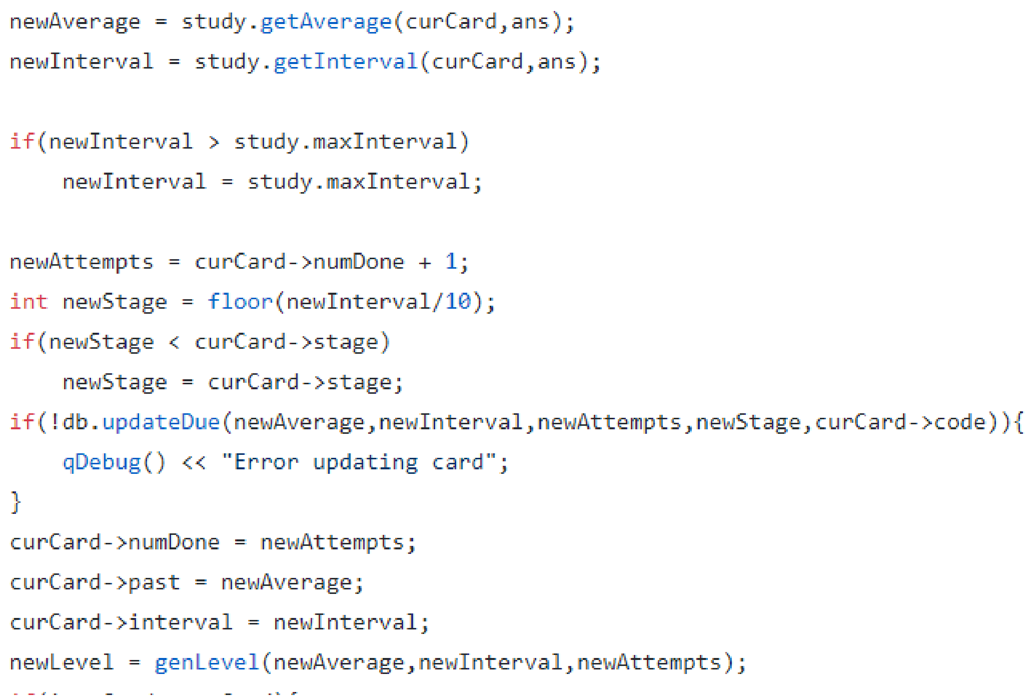


```
QString StudySession::loadHint(Card c, bool sentence){
    if(!sentence)
        return generalHint(c);
    if(c.stage >= 2)
        return generalHint(c);
    QStringList testParts = c.back.split(" ");
    if(testParts.length() < 2)
        return generalHint(c);
    QStringList parts = c.back.split(" ");
    QString retString = "";
    for(int i = 0; i < parts.length(); i++){
        QString curWord = "";
        for(int j = 0; j < parts[i].length(); j++){
            curWord+=parts[i][j];
            QChar newChar = parts[i][j];
            if(gram.checkExists(c.type, curWord)){
                 retString += "-";
            }
            else{
                retString+= newChar;
            }
        }
        if(i < (parts.length()-1))
               retString+= " ";

    }
    if(c.stage == 0){

        return retString;
    }
    else{
        return flipString(retString, c.back);
    }
}
```

Following user input, the answer is evaluated and the card is updated. The following is part of the process for update a card after it is evaluated.



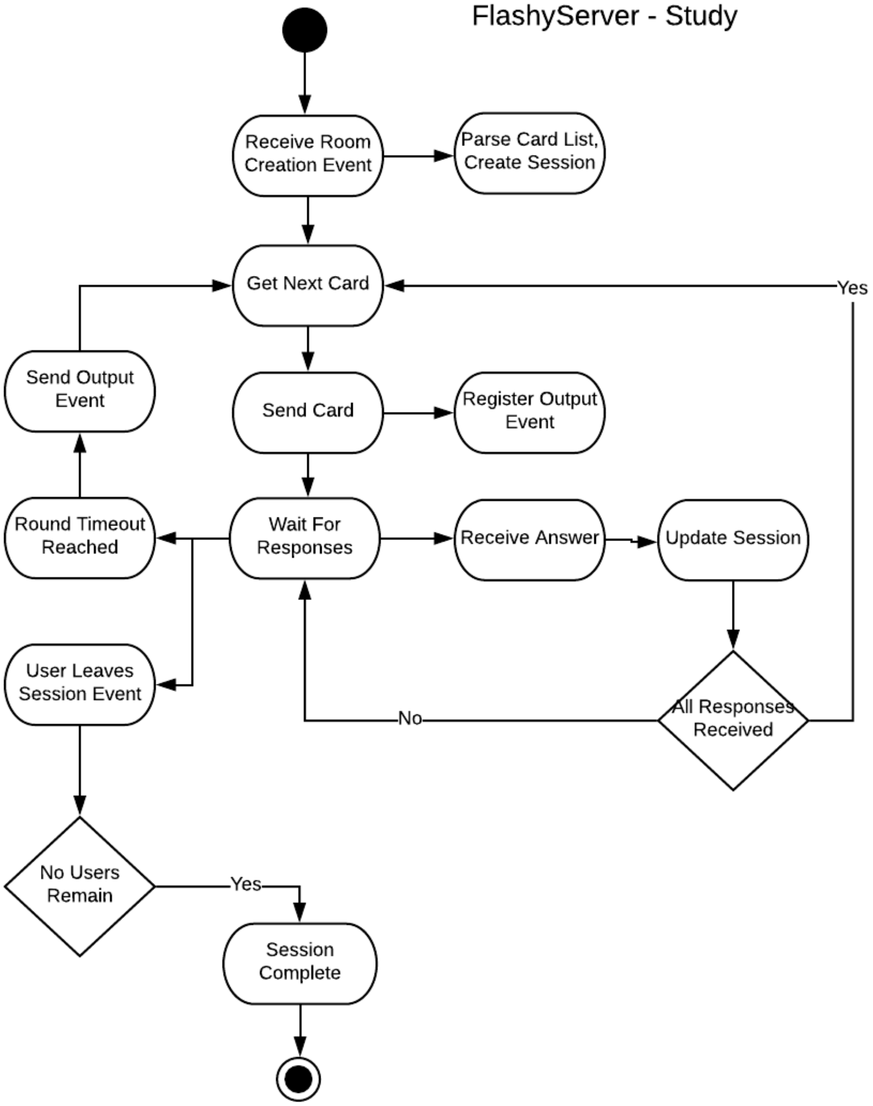
```
newAverage = study.getAverage(curCard,ans);
newInterval = study.getInterval(curCard,ans);

if(newInterval > study.maxInterval)
    newInterval = study.maxInterval;

newAttempts = curCard->numDone + 1;
int newStage = floor(newInterval/10);
if(newStage < curCard->stage)
    newStage = curCard->stage;
if(!db.updateDue(newAverage,newInterval,newAttempts,newStage,curCard->code)){
    qDebug() << "Error updating card";
}
curCard->numDone = newAttempts;
curCard->past = newAverage;
curCard->interval = newInterval;
newLevel = genLevel(newAverage,newInterval,newAttempts);
```

2.5.7 FlashyServer Study Implementation

The following shows the design of the main server side use case – multiplayer study



The design for study session handling on the server side is a fair bit more complex than on the client side. The basic ideas of the system are the same as the non-networked study – a list of cards is set for the session and the users work their way through that list until it is complete. A key difference between the two systems is the inclusion of a round timer, which pushes the next card to all users should the time run out. This was necessary in the case that a group is studying and one person is far too slow, or simply walks away from their computer.

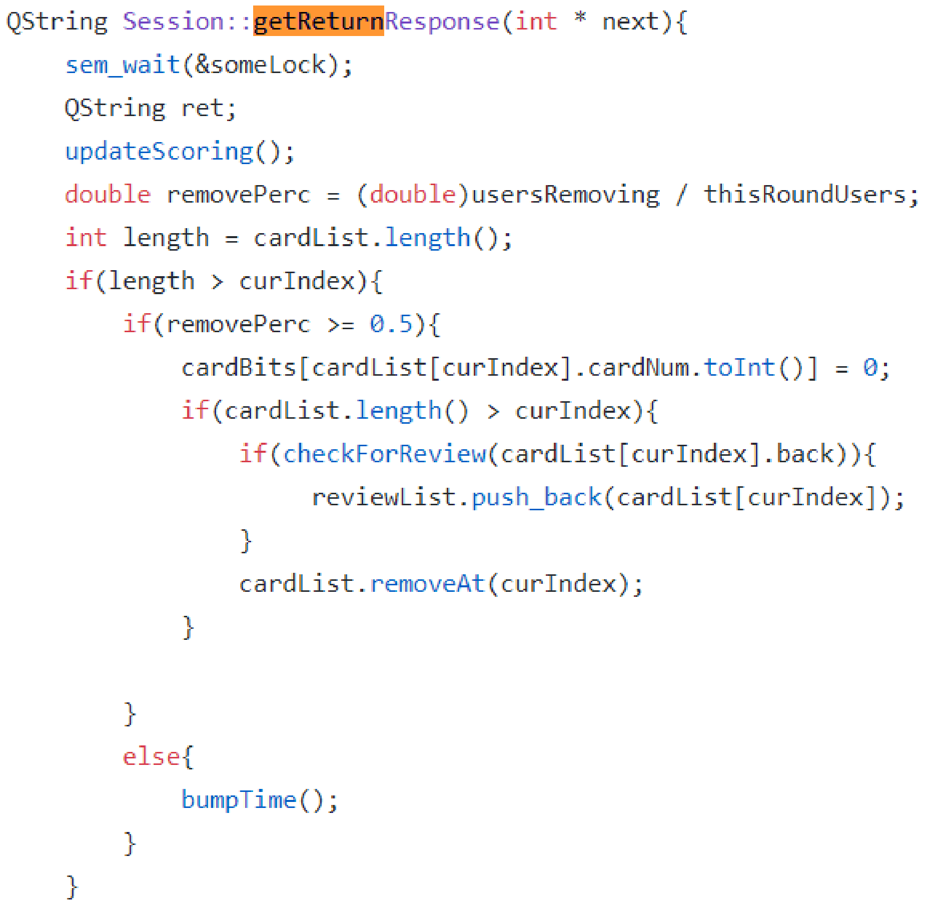
The handling of users and sessions is fairly straightforward. Each user has some small amount of information stored that can be indexed with their socket descriptor. This information includes things like their name, login status and a pointer to their current study session. This simple link makes managing all the users within the system easy. The complexity comes in how many possible scenarios and combinations of events can happen and must be handled during the overall study process, as well as ensuring that everything remains thread safe with all the different handlers.

The following code snippet is part of processing a ‘Room Creation Event’.

```
QList<fbCard> curCards = data.getAllCards(deckID);
for(int i = 0; i < curCards.length(); i++){
    int cardNum = curCards[i].cardNum.toInt();
    if(cardNum >= BITSETSIZE)
        continue;
    if(curReq.test(cardNum)){
        curCards[i].lastTry = QDateTime::currentDateTime();
        newSession.cardList.push_back(curCards[i]);

    }
}
user u;
//get name from global list here
u.score = 0;
u.sock = sock;
QString newName = userDat[sock].name;
if(newName == "")
    newName = getNewName();
u.name = newName;
userDat[sock].name = newName;
newSession.userList.push_back(u);
newSession.games = games;
newSession.interval = interval;
newSession.independant = independant;
newSession.setMaxCard();
newSession.uniqueId = totalSessions++;
```

The following shows part of the process for handling user responses to the current study round.



```
QString Session::getReturnResponse(int * next){
    sem_wait(&someLock);
    QString ret;
    updateScoring();
    double removePerc = (double)usersRemoving / thisRoundUsers;
    int length = cardList.length();
    if(length > curIndex){
        if(removePerc >= 0.5){
            cardBits[cardList[curIndex].cardNum.toInt()] = 0;
            if(cardList.length() > curIndex){
                if(checkForReview(cardList[curIndex].back)){
                     reviewList.push_back(cardList[curIndex]);
                }
                cardList.removeAt(curIndex);
            }

        }
        else{
            bumpTime();
        }
    }
```

The following shows the output queue loop.

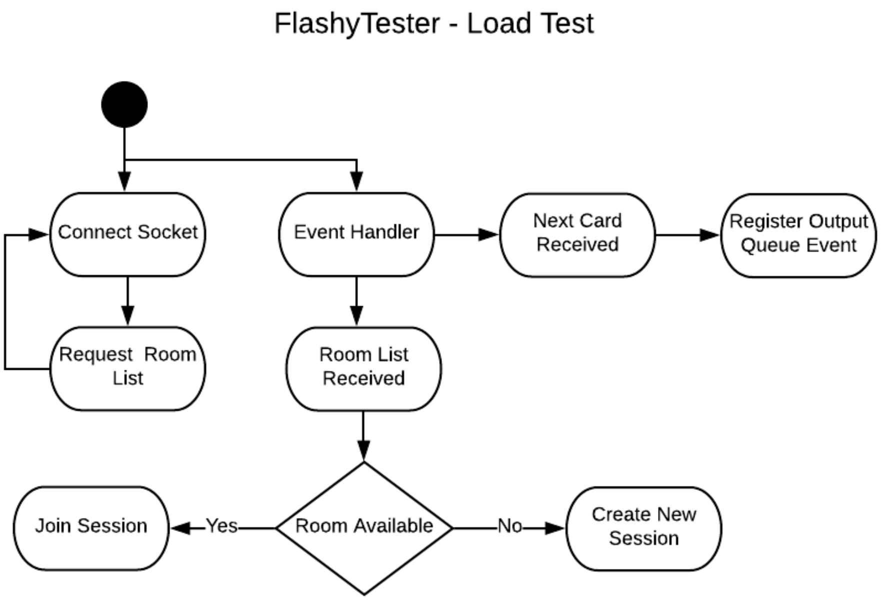
```
while(1){
sem_wait(&queueLock);
    rem = 0;
    cur = QTime::currentTime();
    for(int i =0; i < outQueue.length(); i++){
        if(outQueue[i].sendTime >= cur)
            break;

        rem++;

    }
    QList<outEvent> send;
    for(int i =0; i < rem; i++){
        if(outQueue[0].round == studySessions[outQueue[0].index].getRound()){
            send.push_back(outQueue[0]);
        }
        outQueue.removeAt(0);
    }
sem_post(&queueLock);
    for(int i = 0; i < send.length();i ++){
        sendNext(send[i].index);
    }
```

2.5.8 Tester Implementation

The following shows the design of the main testing program use case – load testing



The implementation of the load testing program is a combination of the client and server architecture, as mentioned above. The basics of the program are that some thousands of sockets are created and connected to the server. Each of these sockets sends a request for the current study session list. From this point any responses are handled by the event handler. Receiving a room list request causes the tester to either join an available room, or create a new one should none exist. The other event of note is the Next Card Received event. At this point the tester randomizes some delay and registers a send answer event with the output queue.

The idea behind the program is to test how well the server manages under significant user load. Under normal use, the majority of traffic will be generated during the setup and joining of rooms, and the following studying session itself.

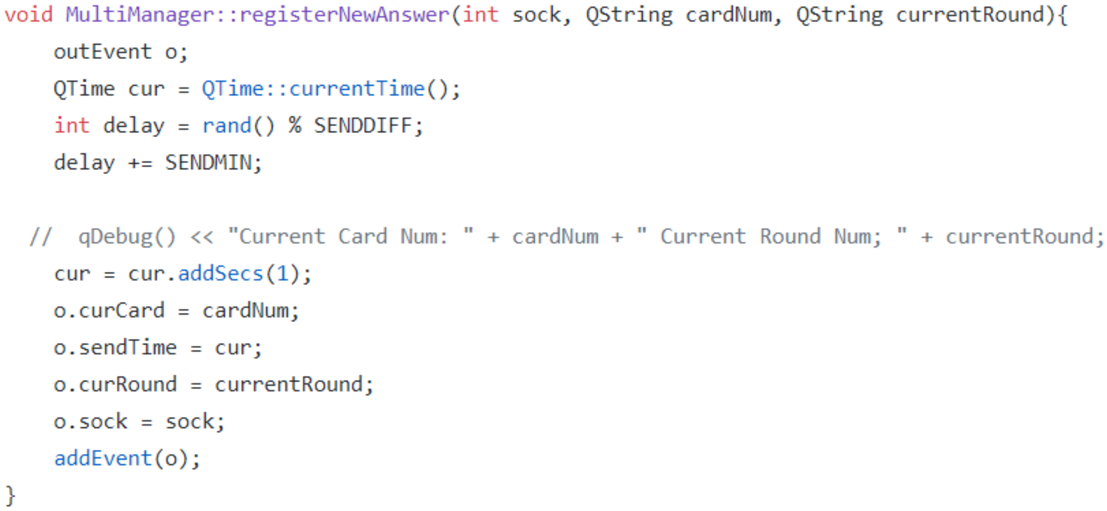
Although having a concurrent user base of several hundred thousand users is not a likely scenario for the overall application, a focus of this project was to ensure good server design which means being able to handle significant user load.

The following shows the beginning of the process for joining a study room. A fake list of cards is created, with each bit of the string representing one card.

```
int place, bit;
int cards = 0;
while(cards < 100){
    int cardNum = cards;
    place = cardNum / 7;
    bit = cardNum % 7;
    buff[place] += pow(2,bit);
    cards++;
}
for(int i = 0; i <= place; i++){
    buff[i] += 128;
}
QString bits = QString::fromLatin1(buff);

send += bits;
send += '\0';
sendData(sock, send);
```

After receiving the ‘Next Card’ event, a delayed response is added to the output queue with the following code.



```
void MultiManager::registerNewAnswer(int sock, QString cardNum, QString currentRound){
    outEvent o;
    QTime cur = QTime::currentTime();
    int delay = rand() % SENDDIFF;
    delay += SENDMIN;

  //  qDebug() << "Current Card Num: " + cardNum + " Current Round Num; " + currentRound;
    cur = cur.addSecs(1);
    o.curCard = cardNum;
    o.sendTime = cur;
    o.curRound = currentRound;
    o.sock = sock;
    addEvent(o);
}
```

2.6 Testing Details and Results

The following are some of the more relevant tests for both the client and server.

Test ID	Name/Description	Steps	Result
2.6.1.1	Create Deck creates a new deck	1. Select ‘Create Deck’ 2. Enter name ‘Test’	PASSED
2.6.1.2	Import Cards loads sample line	1. Select ‘Import Cards’ 2. Select test input file 3. Observe sample line	PASSED
2.6.1.3	Import Cards section selectors work	1. Select ‘Import Cards’ 2. Select sector values 3. Import Cards and observe through Deck Manager	PASSED
2.6.1.4	Deck Manager displays all cards	1. Select ‘Manage Cards’ 2. Observe each unit	PASSED
2.6.1.5	Deck Manager disables selected unit	1. Select ‘Manage Cards’ 2. Disable Unit 0 3. Return to Main Page, observe due cards	PASSED
2.6.1.6	Deck Manager re-enables selected unit	1. Select ‘Manage Cards’ 2. Activate Unit 0 3. Return to Main Paid, observe due cards	PASSED
2.6.1.7	Import Deck from server works	1. Connect to server 2. Select import deck 3. Import test deck, observe under Deck Manager	PASSED
2.6.1.8	Export Deck to server works	1. Connect to server 2. Select test deck 3. Export deck, observe deck display	PASSED
2.8.1.9	Study – Grammar hint displayed	1. Select test deck, ‘Study’ 2. Observe hint, grammar only shown	PASSED

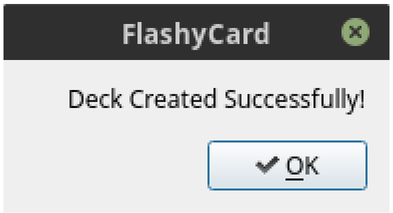
2.8.1.10	Study – Vocab hint displayed	1. Modify card level in database 2. Select ‘Study’ 3. Observe hint, vocab only shown	PASSED
2.8.1.11	Study – Neither hint displayed	1. Modify card level in database 2. Select ‘Study’ 3. Observe hint, only word length shown	PASSED
2.8.1.12	Study – Custom hint displayed	1. Select ‘Study’ 2. Add hint to first card 3. Observe overall hint	PASSED
2.8.1.13	Study – Incorrect card remains in deck, not shown for some time	1. Select ‘Study’ 2. Answer first card incorrectly 3. Card counter not reduced	PASSED
2.8.1.14	Study – Correct card removed from session	1. Select ‘Study’ 2. Answer first card correctly 3. Card counter reduced	PASSED
2.8.1.15	Study – Card interval updated positively on correct answer	1. Select ‘Study’ 2. Answer card correctly, observe new interval	PASSED
2.8.1.16	Study – Card interval updated negatively on incorrect answer	1. Select ‘Study’ 2. Answer card incorrectly, observe new interval	PASSED
2.8.1.17	Connect to server	1. Select ‘Connect’ 2. Observe server message	PASSED
2.8.1.18	Server handles disconnected client	1. Exit client program 2. Observe server message	PASSED
2.8.1.19	Server provides multi study room list	1. Select ‘Multiplayer’ 2. Observe room list	PASSED
2.8.1.20	Create study rooms works	1. Select ‘Multiplayer’ 2. Create study room	PASSED

2.8.1.21	Join study room works	1. Select ‘Multiplayer’ 2. Select room 3. Join room	PASSED
2.8.1.22	Similar card displayed to both users	1. Launch 2 clients 2. Create room, join room 3. Observe card displayed	PASSED
2.8.1.23	Study session handles user leaving – new card	1. Launch 2 clients 2. Create room, join room 3. Answer question on one client, leave on other 4. Update sent from server for new card	PASSED
2.8.1.24	Study session handles user leaving – user list updated	1. Create multi study room 2. Leave room, user removed from scores list	PASSED
2.8.1.25	Study session chat works	1. Create multi study room 2. Send chat from one client to other 3. Chat displayed	PASSED
2.8.1.26	Study session combines different daily cards	1. Create multi study room 2. Join room with separate client, different cards 3. Observe due cards	PASSED
2.8.1.27	Study session displays ‘No current card’ message	1. Create multi study room 2. Join room with 2 additional clients 3. Answer questions right with 1 client, wrong with other 2 4. Observe message once first card displayed again	PASSED
2.8.1.28	Study session scores update	1. Create multi study room 2. Join room with 2 additional clients 3. Score 100, 75 and 50	PASSED

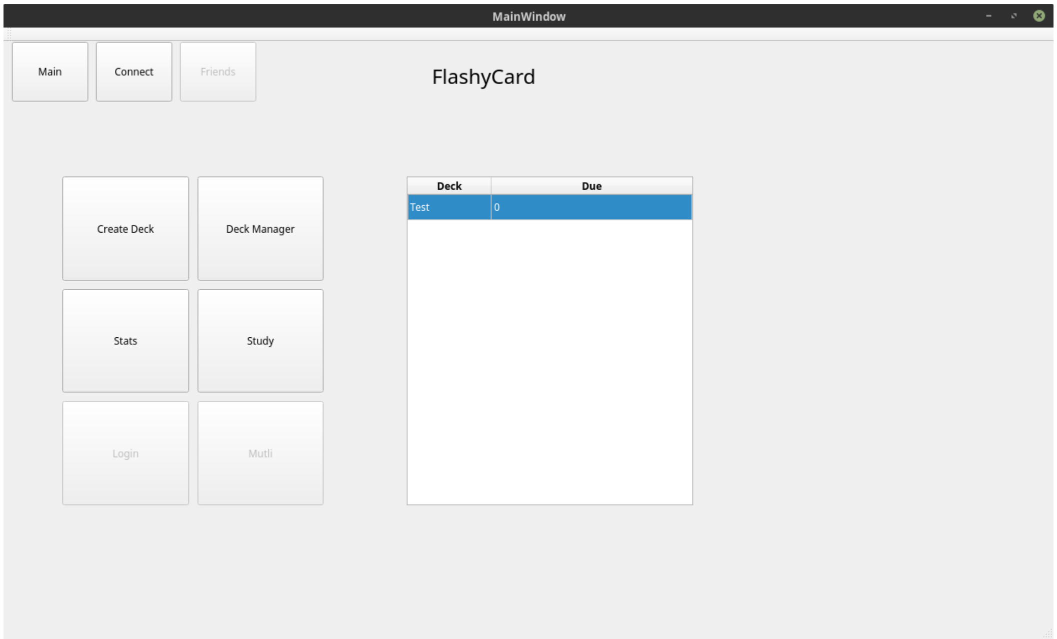
		4. Observe scores for each person	
2.8.1.29	Study session does not boot clients on study complete	1. Create multi study room 2. Complete daily cards	PASSED
2.8.1.30	Study session independent study works	1. Create multi study room 2. Join with second client, different cards 3. Different cards shown to each user	PASSED
2.8.1.31	Study multiplayer game launches	1. Create multi study room 2. Answer questions, wait for game to launch	PASSED
2.8.1.32	Study multiplayer game updates with others moves	1. Create multi study room 2. Answer questions, wait for game to launch 3. Answer question, observe on other client	PASSED
2.8.1.33	Study multiplayer game ends once all moves complete	1. Create multi study room 2. Answer questions, wait for game to launch 3. Complete game	PASSED
2.8.1.34	Client registration works, hashed password	1. Select 'Login' 2. Select 'Register' 3. Enter 'test', 'testpass' 4. Observe user database	PASSED
2.8.1.35	Client login works	1. Select 'Login' 2. Enter 'test', 'testpass' 3. Login message displayed	PASSED
2.8.1.36	Client add friend works	1. Register second user, 'test2' 2. Add user 'test2' 3. Select 'Friends'	PASSED
2.8.1.37	Client online friends show	1. Select 'Friends' 2. Login on second client 3. Observe status change	PASSED

2.8.1.38	Client – Private message works	1. Launch 2 clients 2. Login test, test2 3. Message one another	PASSED
2.8.1.39	Server – Room and user count updates	1. Create multi room 2. Observe room and user count	PASSED
2.8.1.40	Server – Handles large scale room creation and join	1. Run tester 2. Observe room and user count	PASSED
2.8.1.41	Server – Network and Cpu latency graphs update	1. Run tester 2. Observe graphs	PASSED
2.8.1.42	Server – Utility thread activates when event queue slow	1. Run tester 2. Observe utility thread message	PASSED
2.8.1.43	Server – Utility thread activates when network queue slow	1. Run tester 2. Observe utility thread message	PASSED
2.8.1.44	Server – Utility threads transfer from network to event when needed	1. Run tester 2. Observe utility thread message	PASSED
2.8.1.45	Database setup on first time run	1. Run client for first time 2. Observe database in file explorer	PASSED
2.8.1.46	Security – All user communications encrypted	1. Connect to Server 2. Request current session list 3. Observe traffic in Wireshark	PASSED
2.8.1.47	Statistics – Cards done per day tracked	1. Select ‘Stats’ 2. Observe Cards Done	PASSED
2.8.1.48	Statistics – Card level tracked	1. Select ‘Stats’ 2. Observe Card Levels	PASSED

2.6.1.1 Create Deck creates a new deck

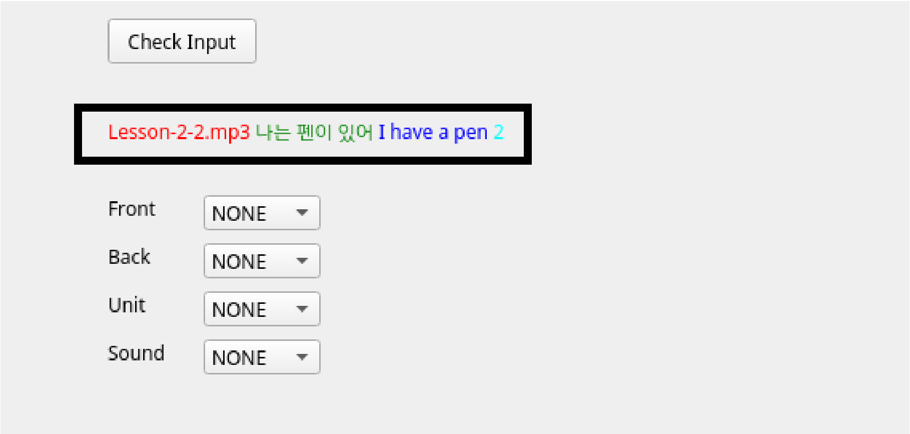


Deck Created Successfully!



Deck	Due
Test	0

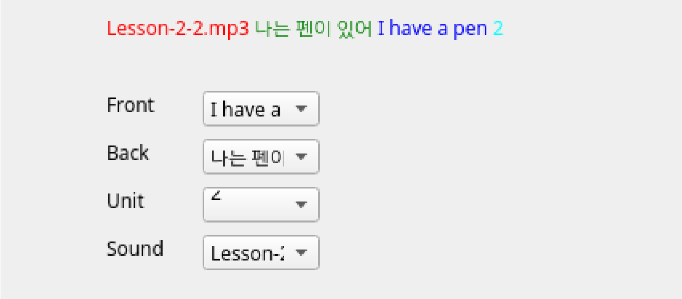
2.6.1.2 Import Cards loads sample line



Lesson-2-2.mp3 나는 펜이 있어 I have a pen 2

Front	NONE
Back	NONE
Unit	NONE
Sound	NONE

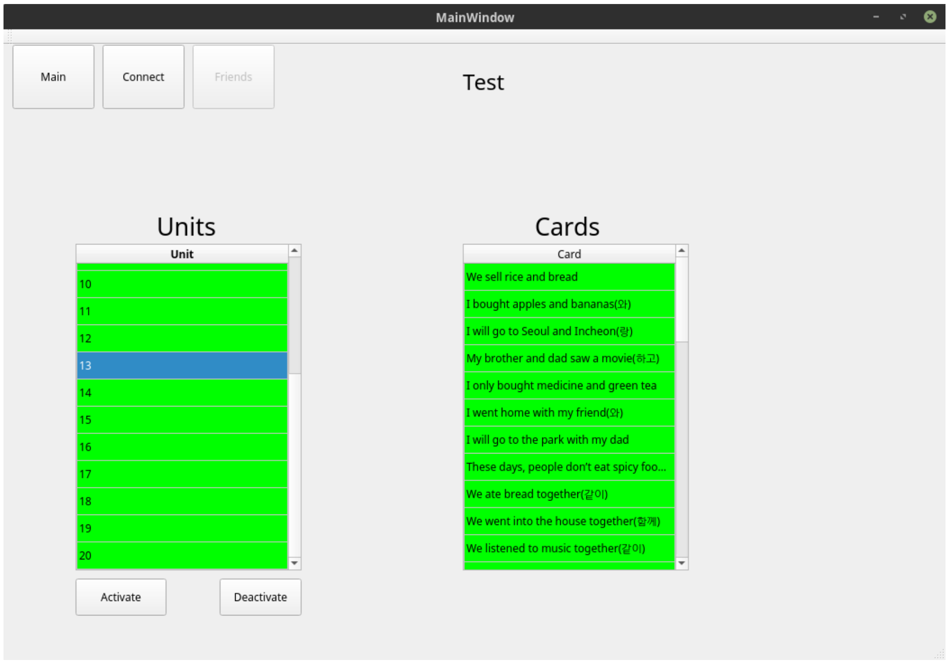
2.6.1.3 Import Cards section selectors work



Lesson-2-2.mp3 나는 펜이 있어 I have a pen 2

Front	I have a
Back	나는 펜이
Unit	2
Sound	Lesson-

2.6.1.4 Deck Manager displays all cards



Test

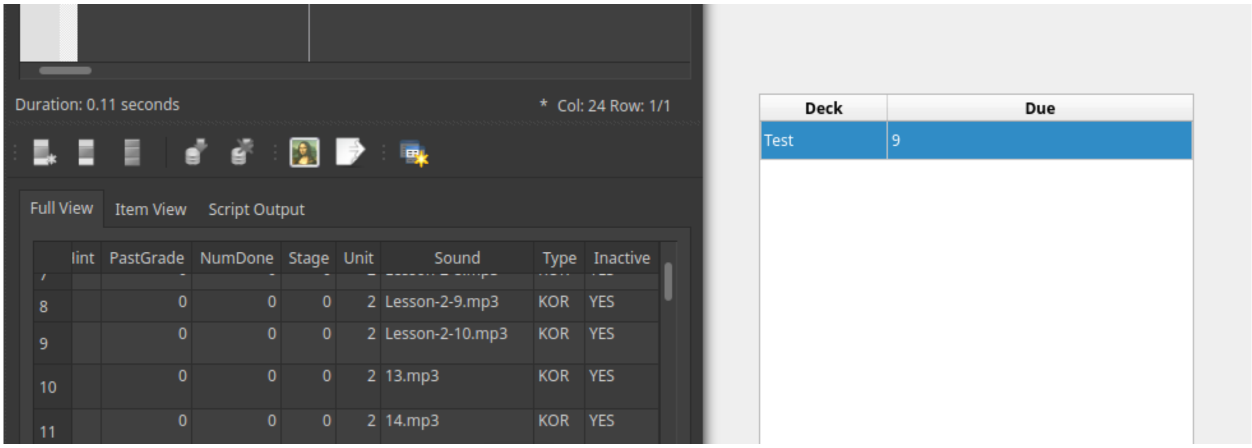
Units

Unit
10
11
12
13
14
15
16
17
18
19
20

Cards

Card
We sell rice and bread
I bought apples and bananas(와)
I will go to Seoul and Incheon(랑)
My brother and dad saw a movie(하고)
I only bought medicine and green tea
I went home with my friend(와)
I will go to the park with my dad
These days, people don't eat spicy foo...
We ate bread together(같이)
We went into the house together(함께)
We listened to music together(같이)

2.6.1.5 Deck Manager disables selected unit



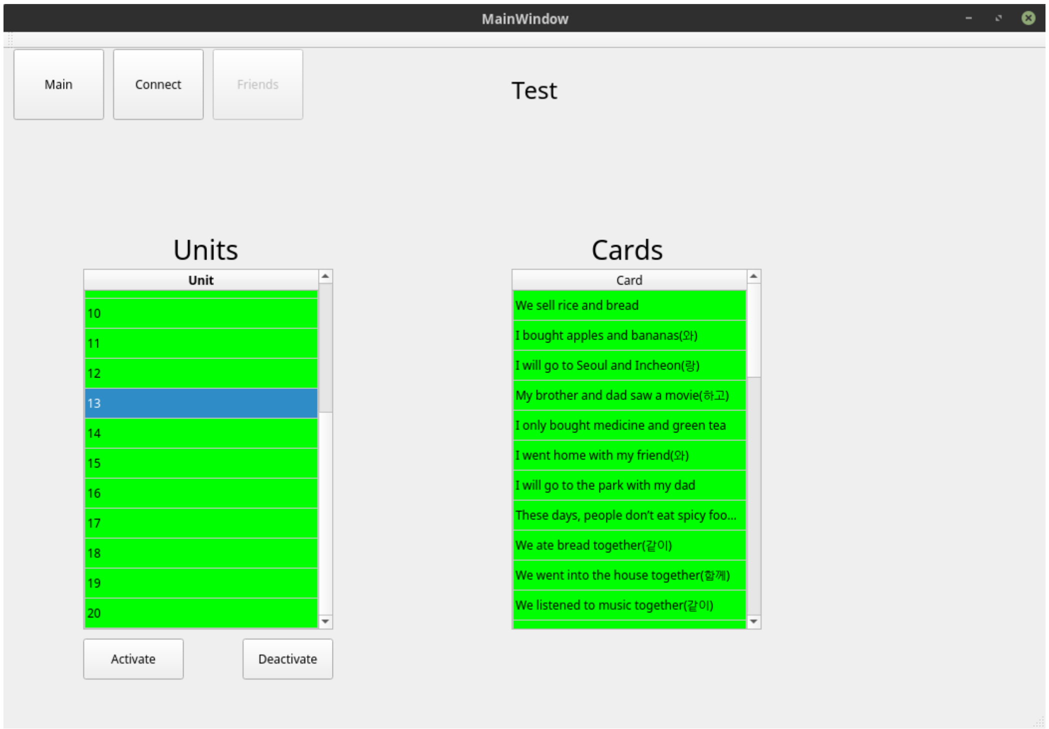
Duration: 0.11 seconds

* Col: 24 Row: 1/1

	lint	PastGrade	NumDone	Stage	Unit	Sound	Type	Inactive
8		0	0	0	2	Lesson-2-9.mp3	KOR	YES
9		0	0	0	2	Lesson-2-10.mp3	KOR	YES
10		0	0	0	2	13.mp3	KOR	YES
11		0	0	0	2	14.mp3	KOR	YES

Deck	Due
Test	9

2.6.1.6 Deck Manager re-enables selected unit



Test

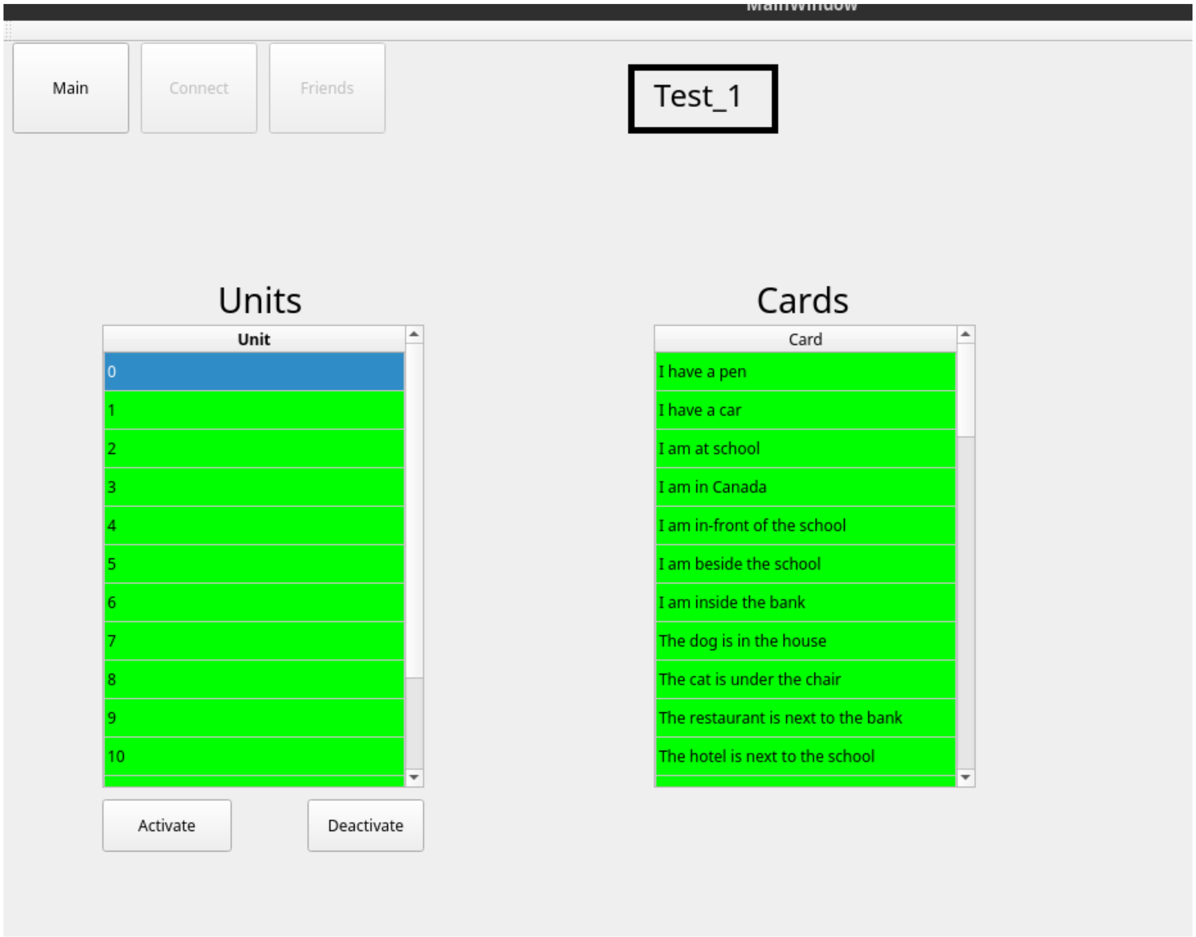
Units

Unit
10
11
12
13
14
15
16
17
18
19
20

Cards

Card
We sell rice and bread
I bought apples and bananas(와)
I will go to Seoul and Incheon(랑)
My brother and dad saw a movie(하고)
I only bought medicine and green tea
I went home with my friend(와)
I will go to the park with my dad
These days, people don't eat spicy foo...
We ate bread together(같이)
We went into the house together(함께)
We listened to music together(같이)

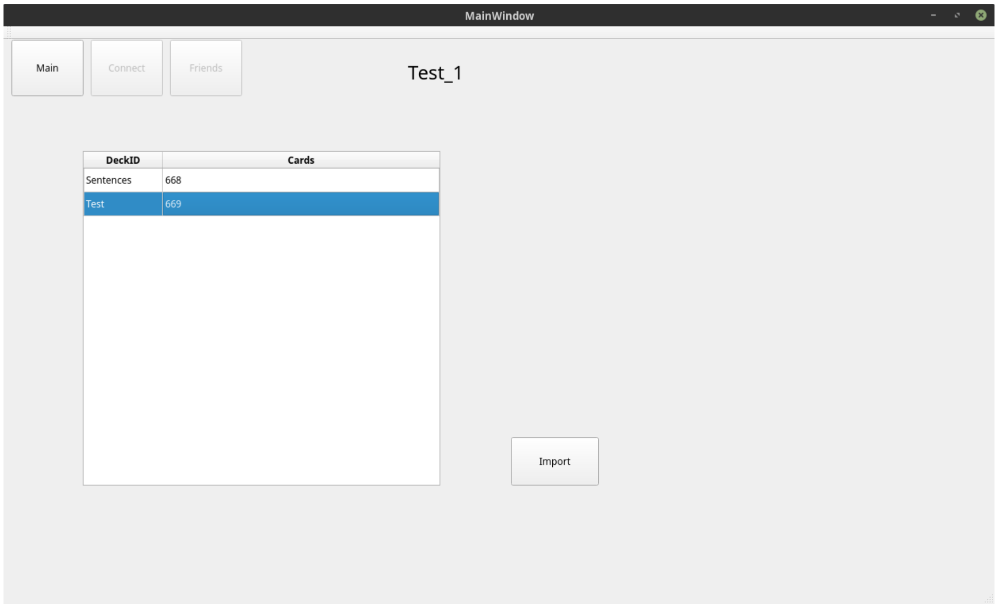
2.6.1.7 Import Deck from server works



Unit
0
1
2
3
4
5
6
7
8
9
10

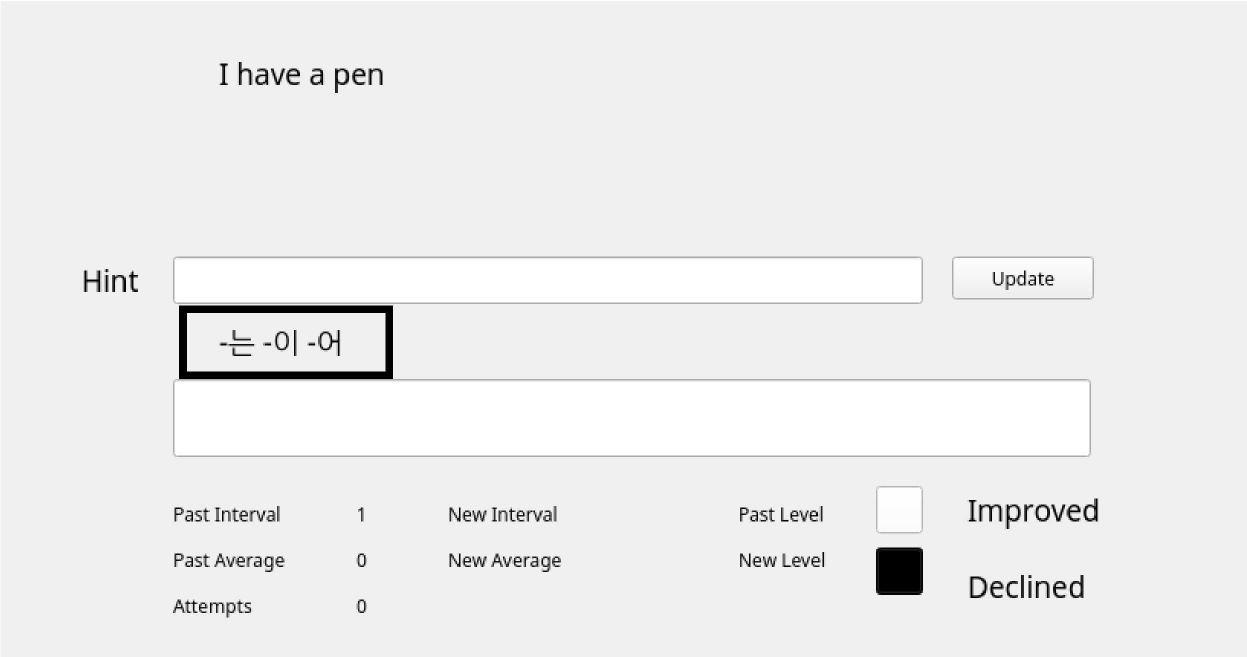
Card
I have a pen
I have a car
I am at school
I am in Canada
I am in-front of the school
I am beside the school
I am inside the bank
The dog is in the house
The cat is under the chair
The restaurant is next to the bank
The hotel is next to the school

2.6.1.8 Export Deck to server works



DeckID	Cards
Sentences	668
Test	669

2.8.1.9 Study – Grammar hint displayed



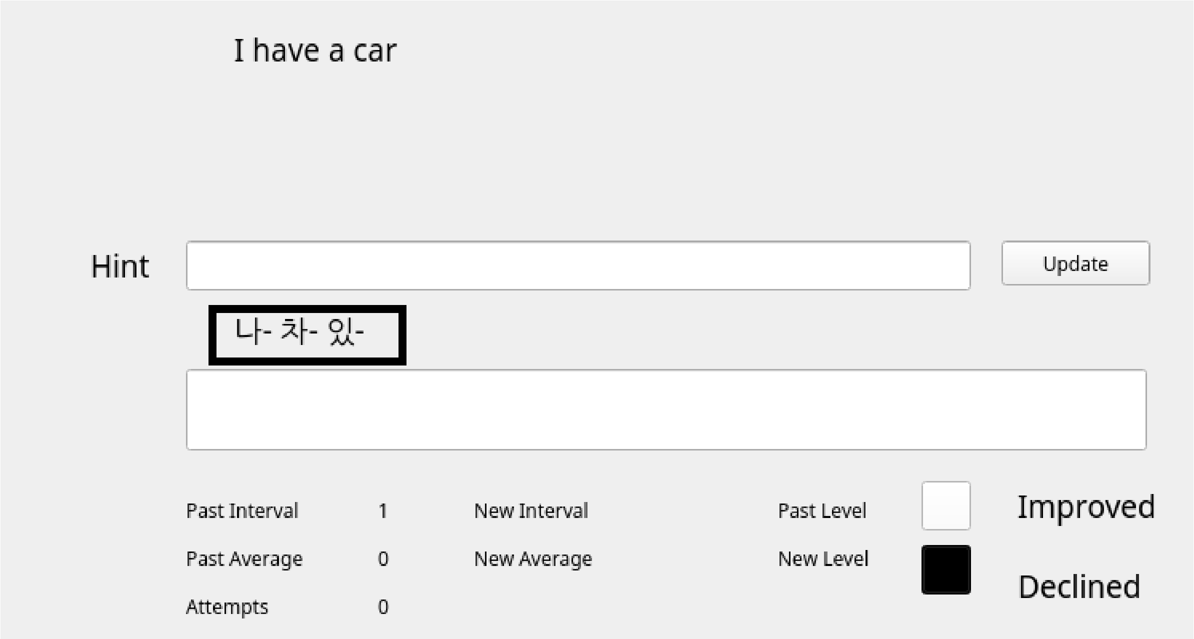
I have a pen

Hint

-는 -이 -어

Past Interval 1
Past Average 0
Attempts 0
New Interval
New Average
Past Level □ Improved
New Level ■ Declined

2.8.1.10 Study – Vocab hint displayed



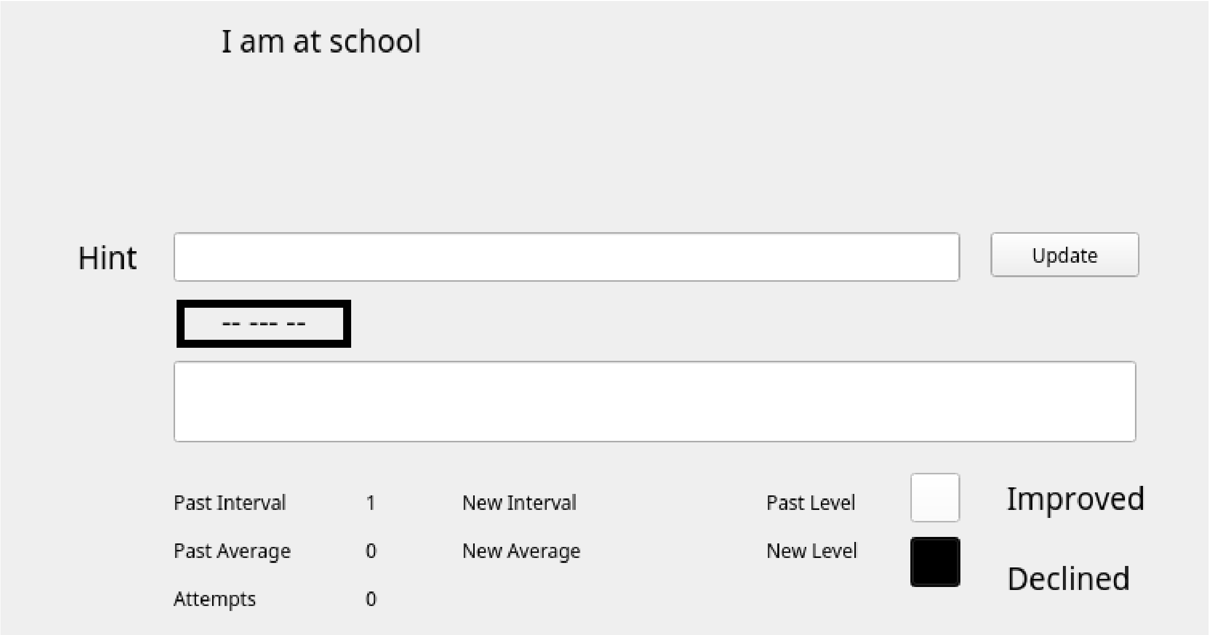
I have a car

Hint

나- 차- 있-

Past Interval 1
Past Average 0
Attempts 0
New Interval
New Average
Past Level □ Improved
New Level ■ Declined

2.8.1.11 Study – Neither hint displayed

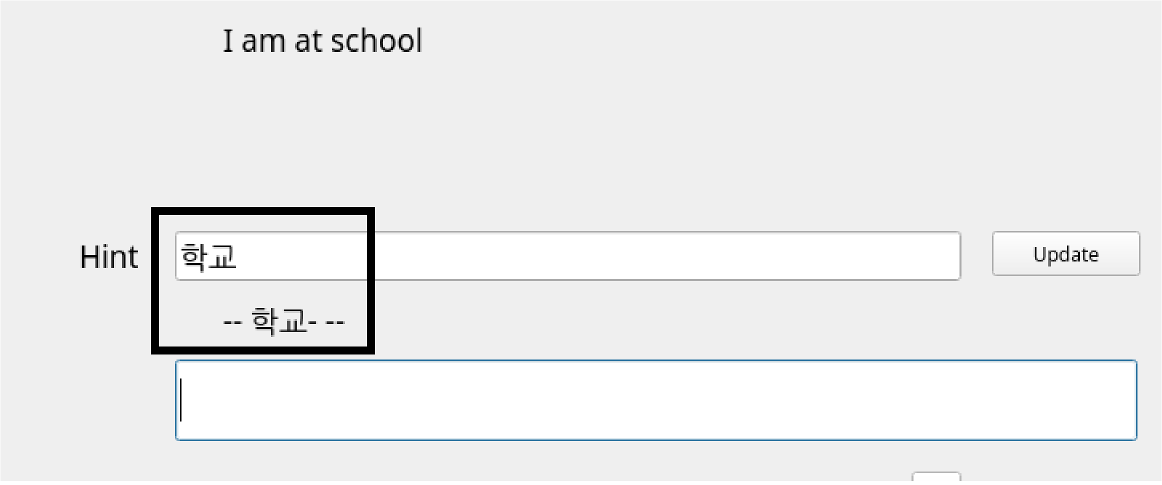


I am at school

-- --- --

Past Interval	1	New Interval		Past Level
Past Average	0	New Average		New Level
Attempts	0			

2.8.1.12 Study – Custom hint displayed

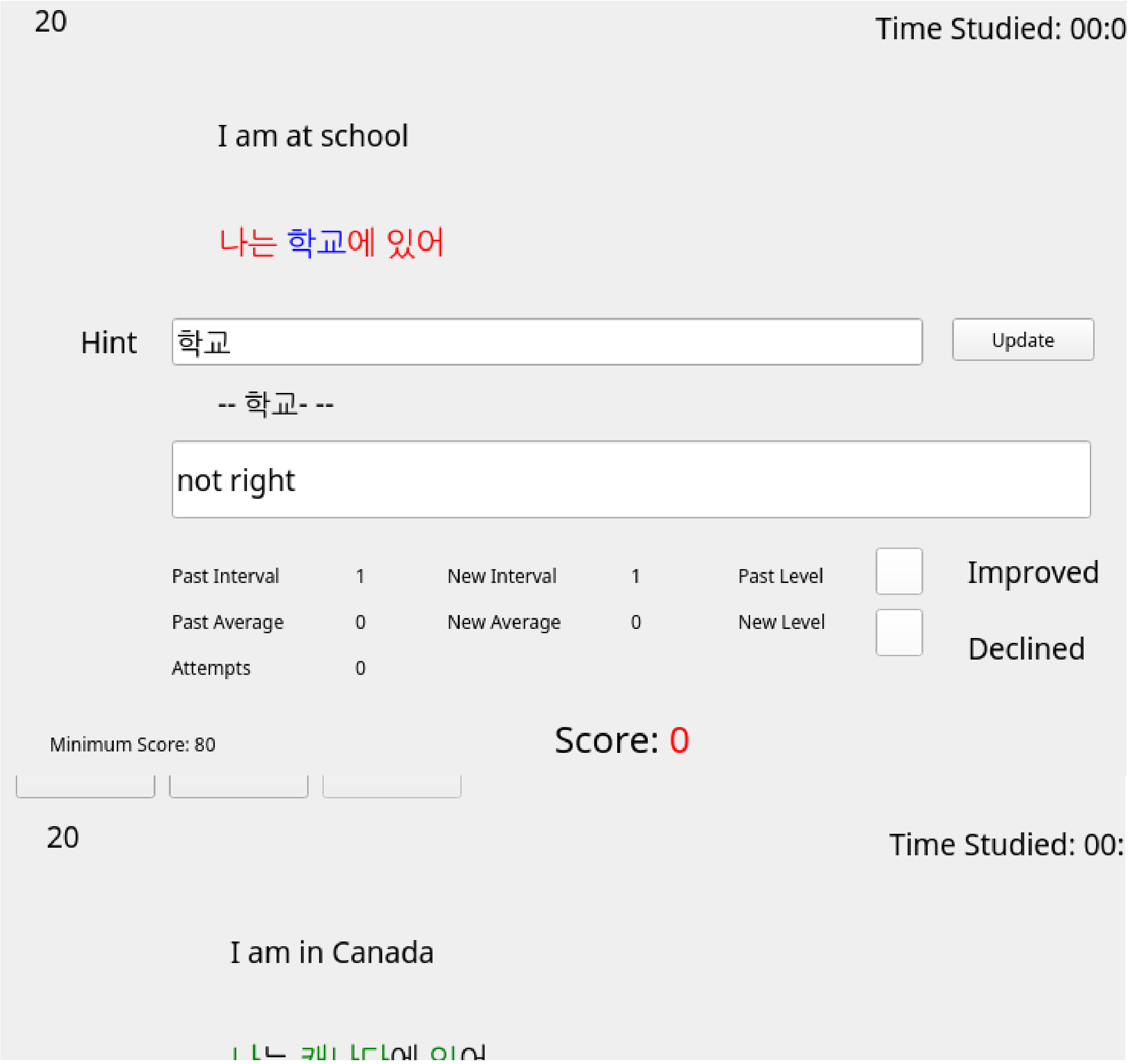


I am at school

학교

-- 학교- --

2.8.1.13 Study – Incorrect card remains in deck, not shown for some time



20

Time Studied: 00:0

I am at school

나는 학교에 있어

Hint: 학교

-- 학교- --

not right

Past Interval 1

New Interval 1

Past Level □ Improved

Past Average 0

New Average 0

New Level □ Declined

Attempts 0

Minimum Score: 80

Score: 0

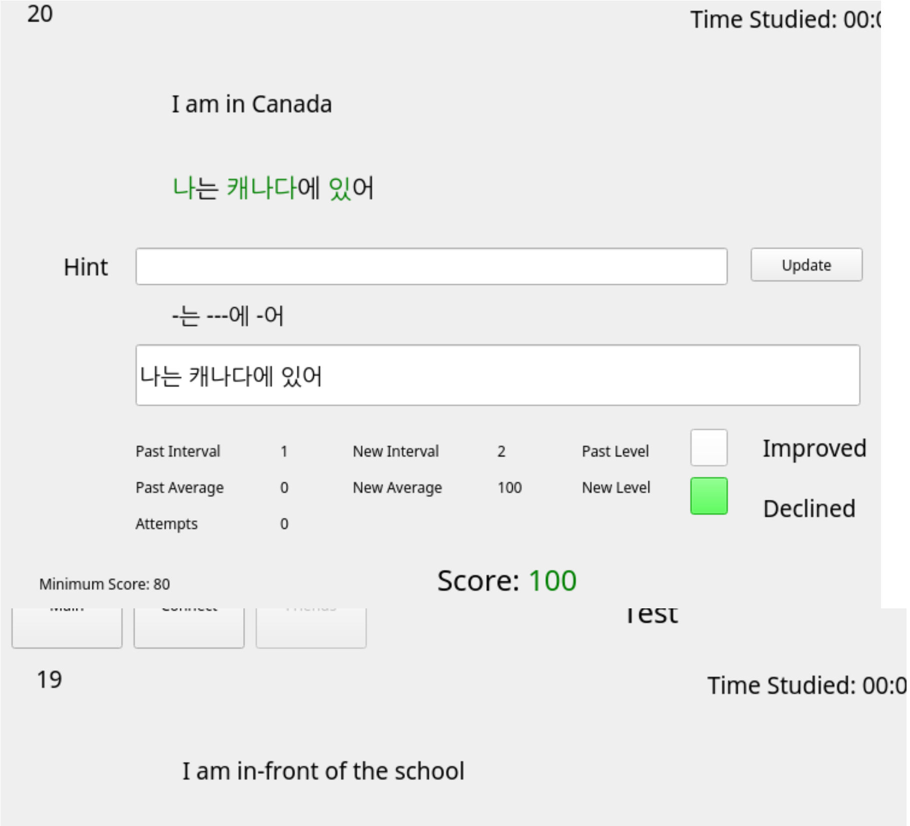
20

Time Studied: 00:

I am in Canada

나는 캐나다에 있어

2.8.1.14 Study – Correct card removed from session



20

Time Studied: 00:(

I am in Canada

나는 캐나다에 있어

Hint

-는 ---에 -어

나는 캐나다에 있어

Past Interval	1	New Interval	2	Past Level	□
Past Average	0	New Average	100	New Level	🟩
Attempts	0				

Improved

Declined

Minimum Score: 80

Score: 100

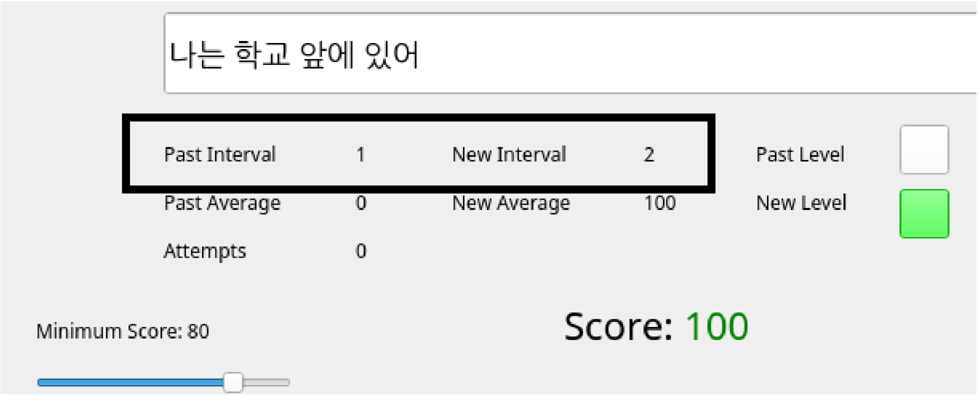
Test

19

Time Studied: 00:0

I am in-front of the school

2.8.1.15 Study – Card interval updated positively on correct answer



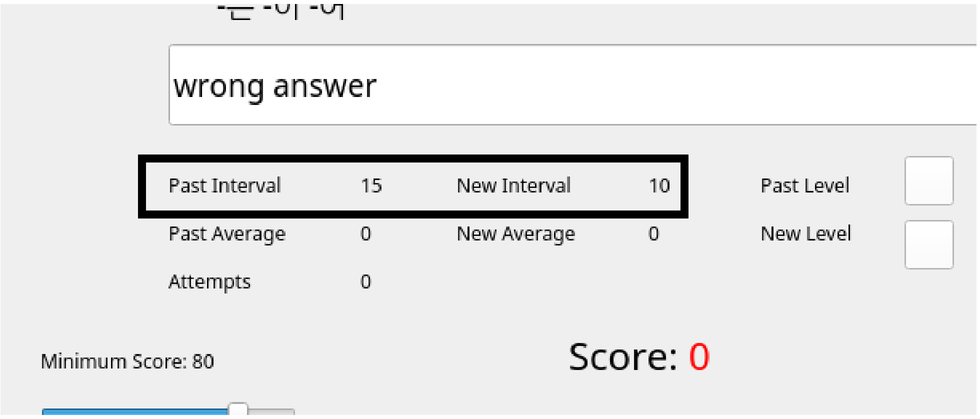
나는 학교 앞에 있어

Past Interval	1	New Interval	2	Past Level	□
Past Average	0	New Average	100	New Level	🟩
Attempts	0				

Minimum Score: 80

Score: 100

2.8.1.16 Study – Card interval updated negatively on incorrect answer



-는 -이 -어

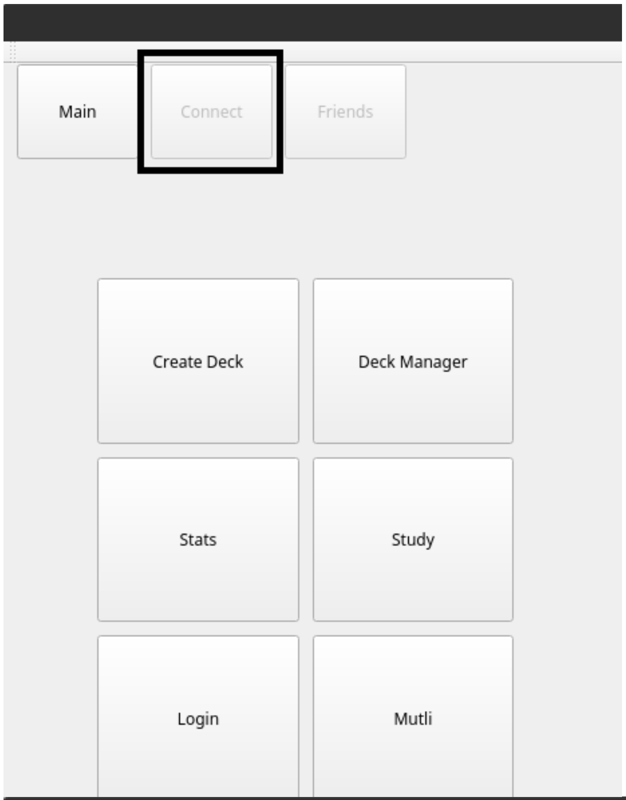
wrong answer

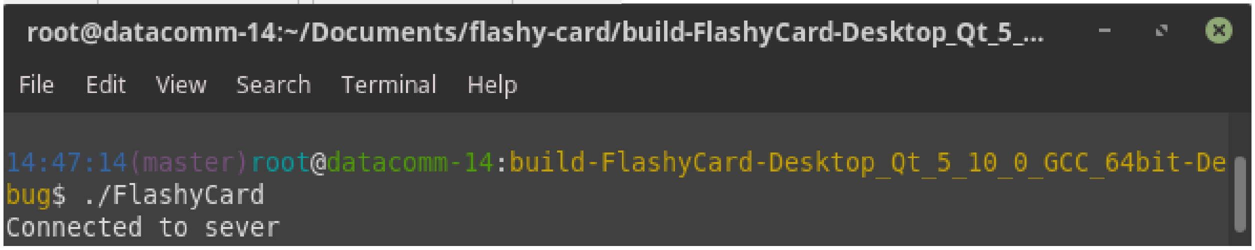
Past Interval	15	New Interval	10	Past Level	□
Past Average	0	New Average	0	New Level	□
Attempts	0				

Minimum Score: 80

Score: 0

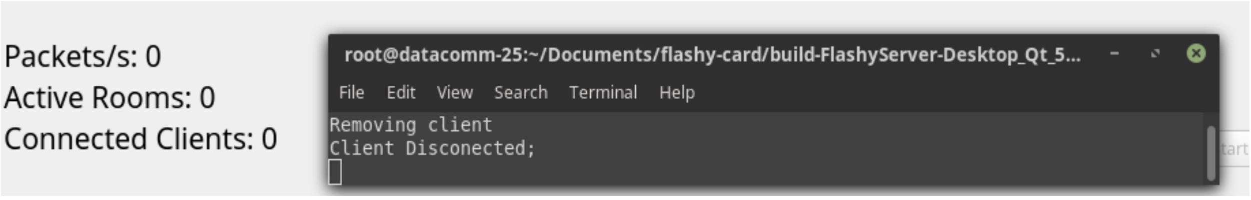
2.8.1.17 Connect to server




root@datacomm-14:~/Documents/flashy-card/build-FlashyCard-Desktop_Qt_5_...
File Edit View Search Terminal Help

```
14:47:14(master)root@datacomm-14:build-FlashyCard-Desktop_Qt_5_10_0_GCC_64bit-Debug$ ./FlashyCard
Connected to sever
```

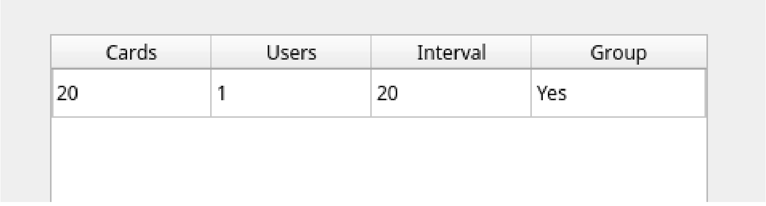
2.8.1.18 Server handles disconnected client


Packets/s: 0
Active Rooms: 0
Connected Clients: 0

root@datacomm-25:~/Documents/flashy-card/build-FlashyServer-Desktop_Qt_5...
File Edit View Search Terminal Help

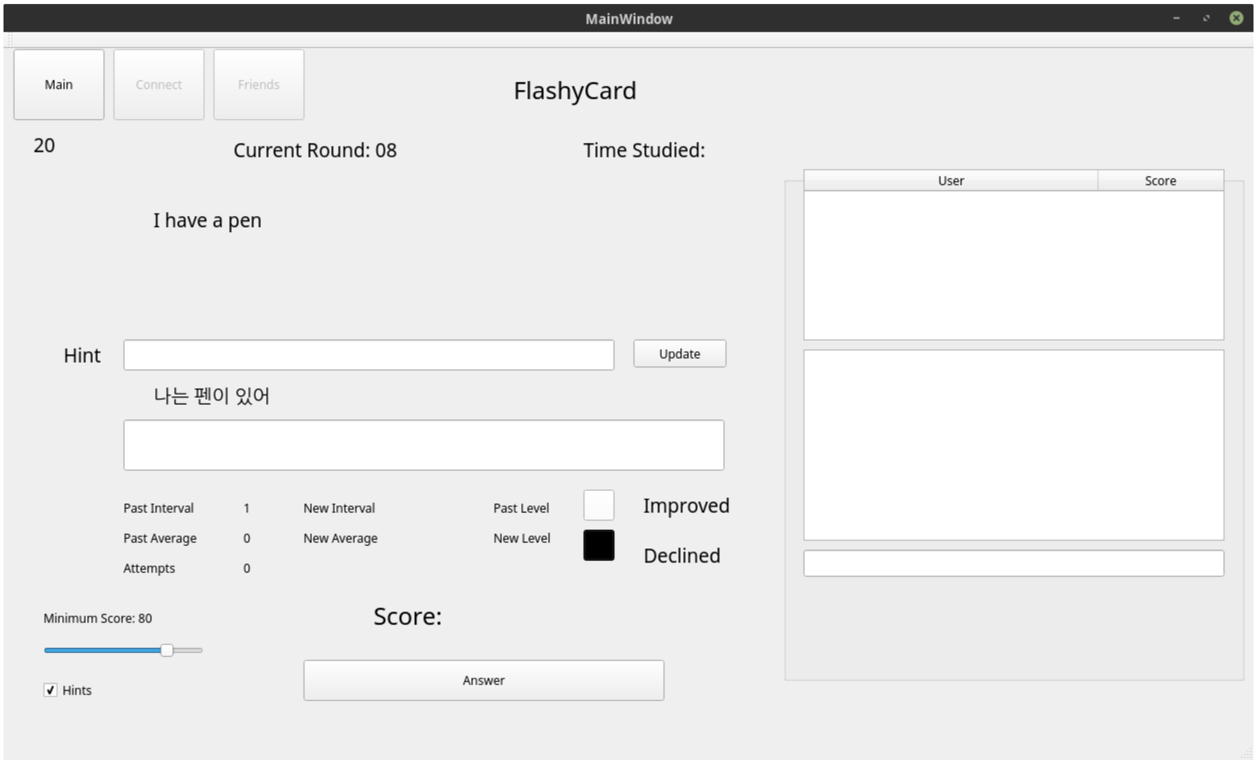
```
Removing client
Client Disconected;
```

2.8.1.19 Server provides multi study room list



Cards	Users	Interval	Group
20	1	20	Yes

2.8.1.20 Create study rooms works



20

Current Round: 08

I have a pen

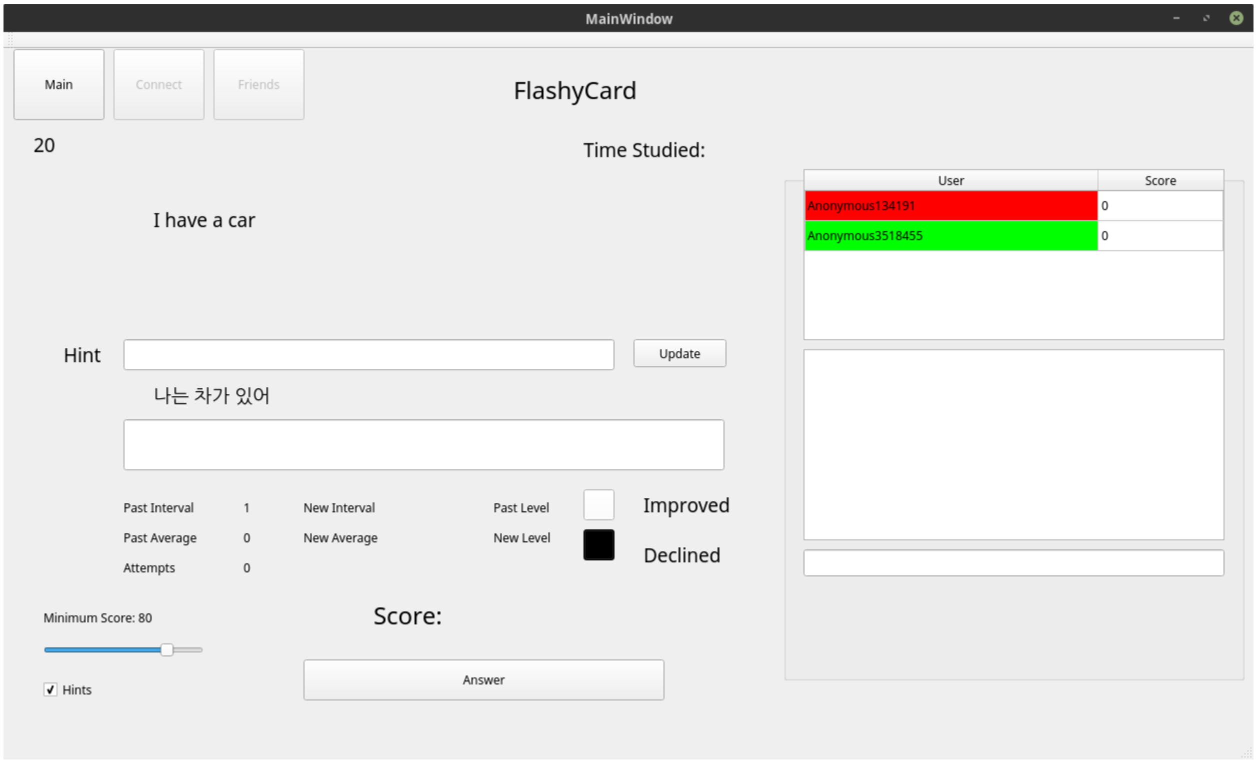
나는 펜이 있어

Past Interval	1	New Interval		Past Level
Past Average	0	New Average		New Level
Attempts	0			

Minimum Score: 80

☑ Hints

2.8.1.21 Join study room works



20

I have a car

나는 차가 있어

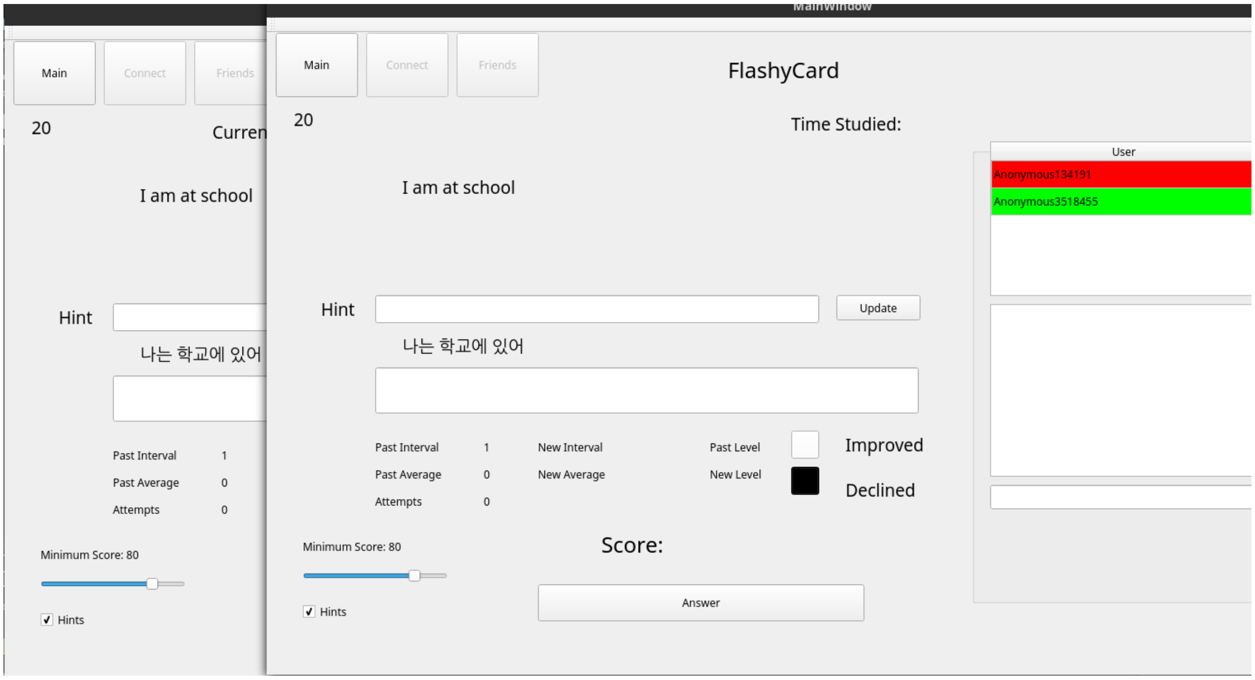
User	Score
🟥 Anonymous134191	0
🟩 Anonymous3518455	0

Past Interval	1	New Interval		Past Level
Past Average	0	New Average		New Level
Attempts	0			

Minimum Score: 80

☑ Hints

2.8.1.22 Similar card displayed to both users



20

I am at school

나는 학교에 있어

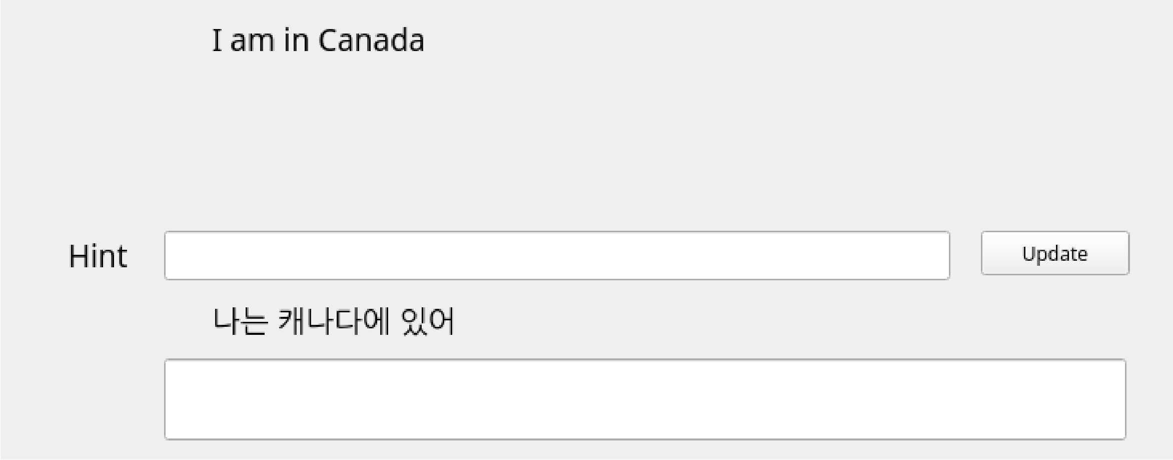
User
Anonymous134191
Anonymous3518455

Past Interval 1
Past Average 0
Attempts 0
New Interval
New Average
Past Level
New Level

Minimum Score: 80

☑ Hints

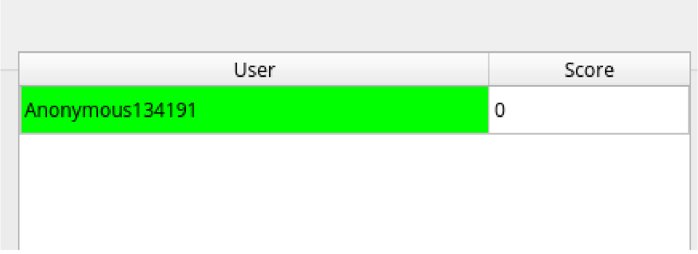
2.8.1.23 Study session handles user leaving – new card



I am in Canada

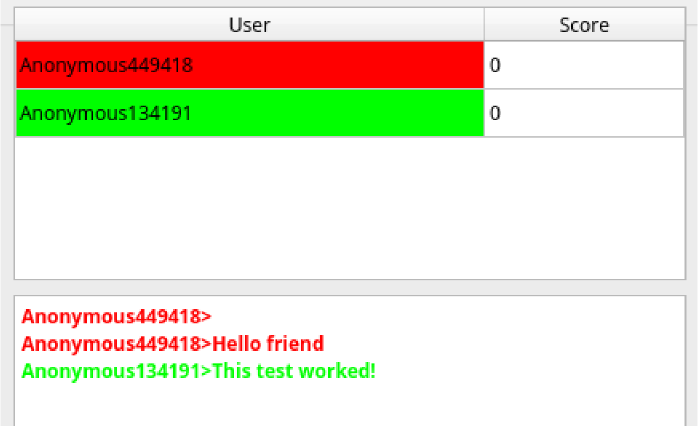
나는 캐나다에 있어

2.8.1.24 Study session handles user leaving – user list updated



User	Score
Anonymous134191	0

2.8.1.25 Study session chat works

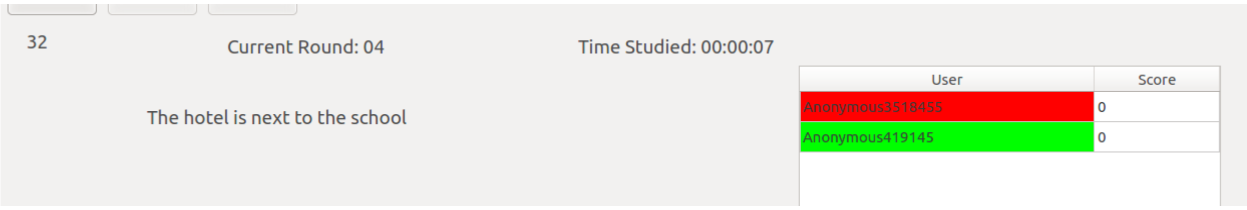


User	Score
Anonymous449418	0
Anonymous134191	0

Anonymous449418>
Anonymous449418>Hello friend
Anonymous134191>This test worked!

2.8.1.26 Study session combines different daily cards

Study Session Creator



32

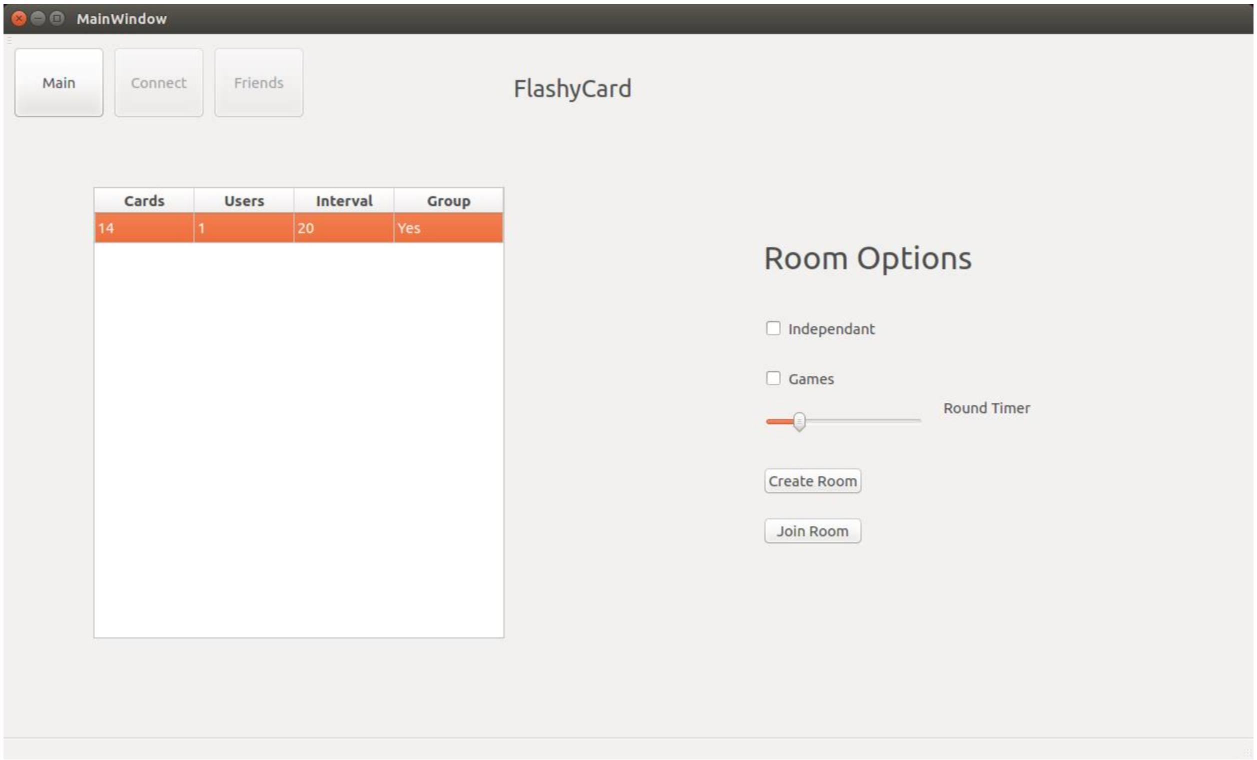
Current Round: 04

Time Studied: 00:00:07

The hotel is next to the school

User	Score
Anonymous3518455	0
Anonymous419145	0

User with partial matches

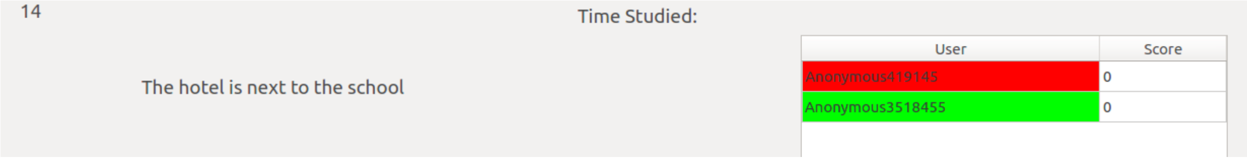


Cards	Users	Interval	Group
14	1	20	Yes

☐ Independant

☐ Games

Session adjusted to the 14 matched cards



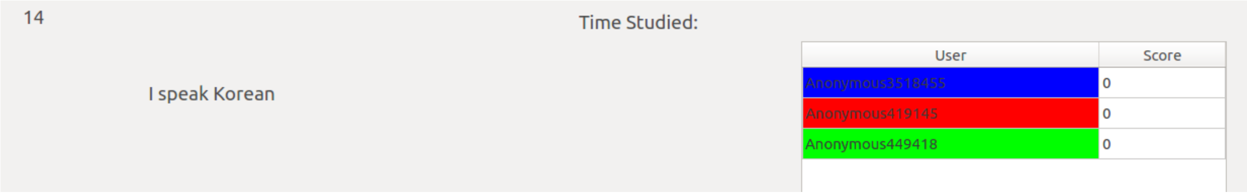
14

Time Studied:

The hotel is next to the school

User	Score
Anonymous419145	0
Anonymous3518455	0

New user also limited to shared cards



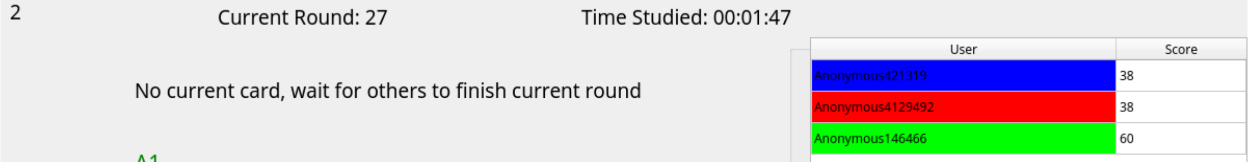
14

Time Studied:

I speak Korean

User	Score
Anonymous3518455	0
Anonymous419145	0
Anonymous449418	0

2.8.1.27 Study session displays ‘No current card’ message



2

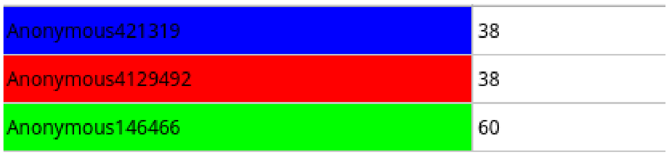
Current Round: 27

Time Studied: 00:01:47

No current card, wait for others to finish current round

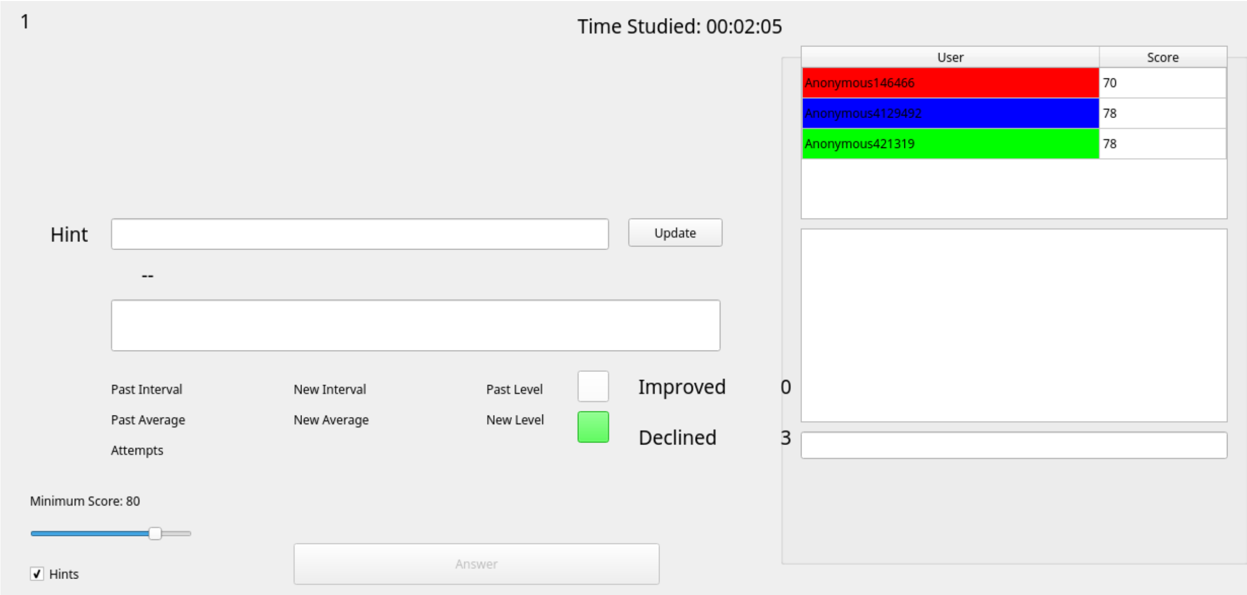
User	Score
Anonymous421319	38
Anonymous4129492	38
Anonymous146466	60

2.8.1.28Study session scores update



Anonymous421319	38
Anonymous4129492	38
Anonymous146466	60

2.8.1.29 Study session does not boot clients on study complete

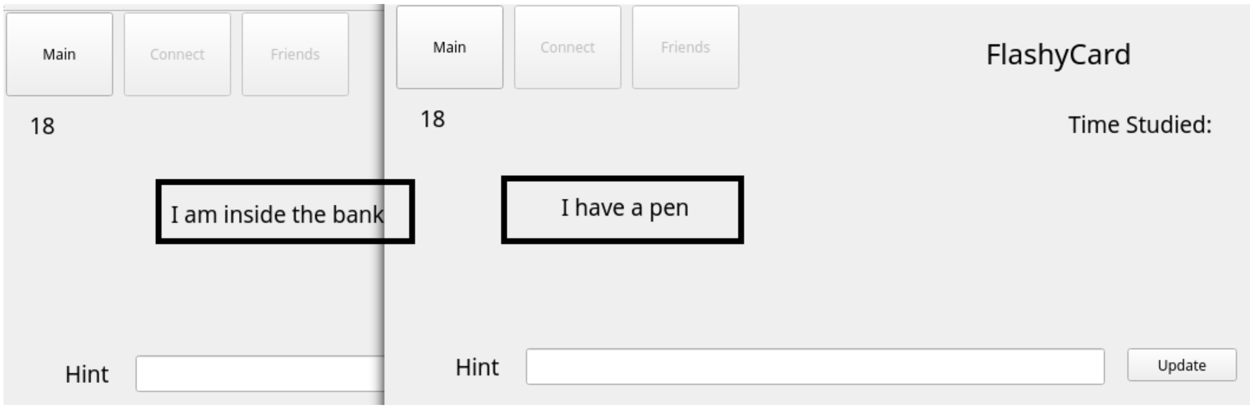


1

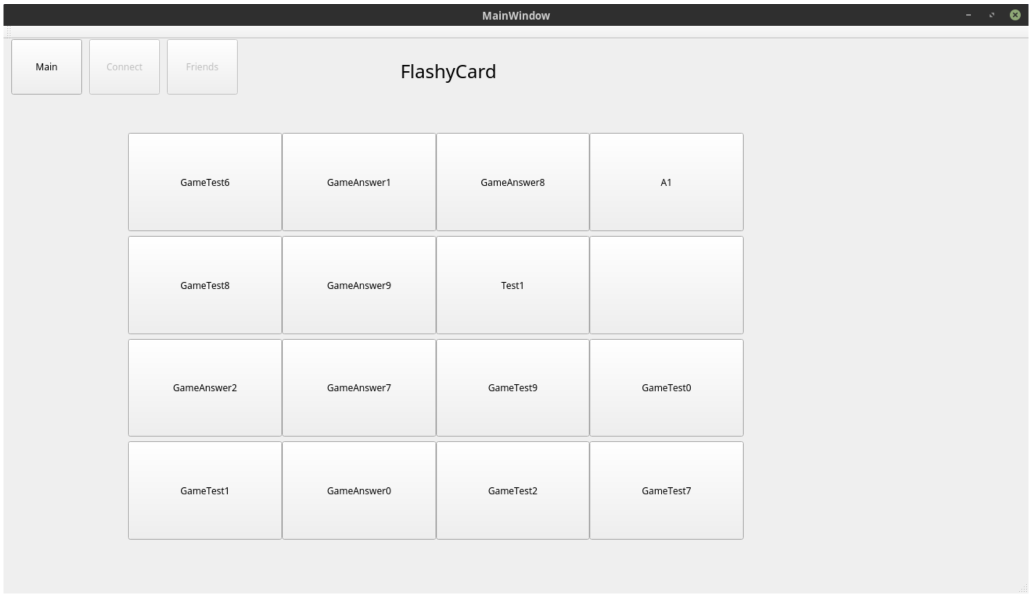
Time Studied: 00:02:05

User	Score
Anonymous146466	70
Anonymous4129492	78
Anonymous421319	78

2.8.1.30 Study session independent study works

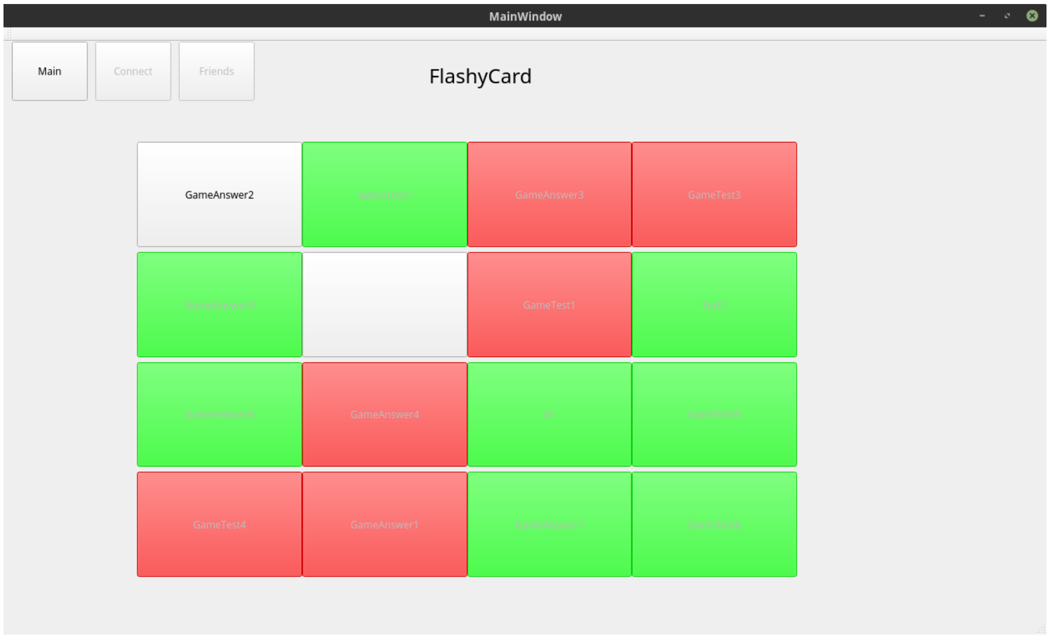


2.8.1.31 Study multiplayer game launches



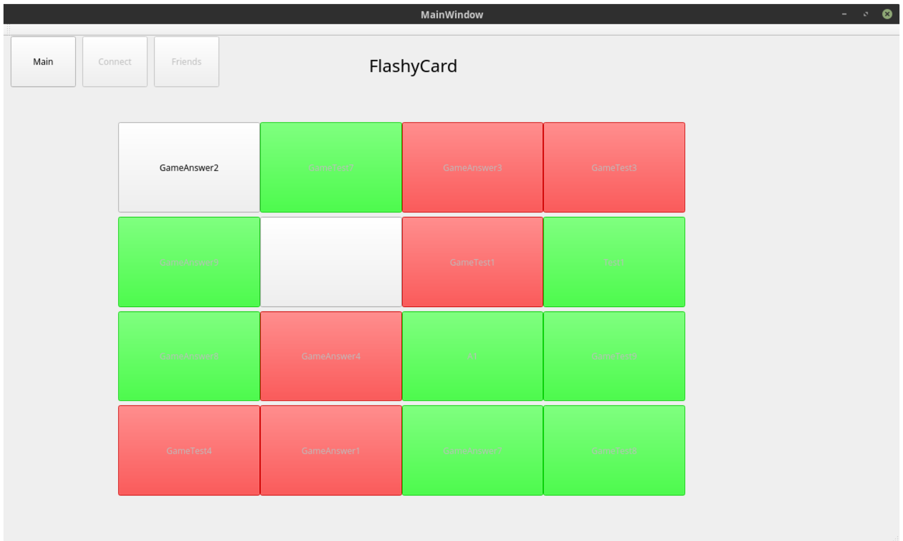
GameTest6	GameAnswer1	GameAnswer8	A1
GameTest8	GameAnswer9	Test1	
GameAnswer2	GameAnswer7	GameTest9	GameTest0
GameTest1	GameAnswer0	GameTest2	GameTest7

2.8.1.32 Study multiplayer game updates with others moves



GameAnswer2	GameTest7	GameAnswer3	GameTest3
GameAnswer9		GameTest1	Test1
GameAnswer8	GameAnswer4	A1	GameTest9
GameTest4	GameAnswer1	GameAnswer7	GameTest8

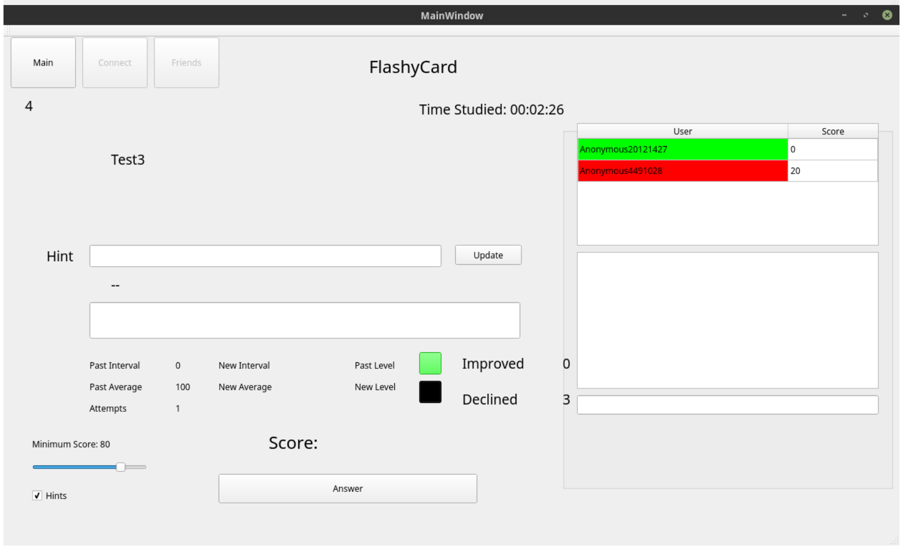
2.8.1.33 Study multiplayer game ends once all moves complete



MainWindow

FlashyCard

GameAnswer2	GameTest7	GameAnswer3	GameTest3
GameAnswer9		GameTest1	Test1
GameAnswer8	GameAnswer4	A1	GameTest9
GameTest4	GameAnswer1	GameAnswer7	GameTest8



MainWindow

FlashyCard

4

Time Studied: 00:02:26

User	Score
Anonymous20121427	0
Anonymous4491028	20

Test3

Hint

--

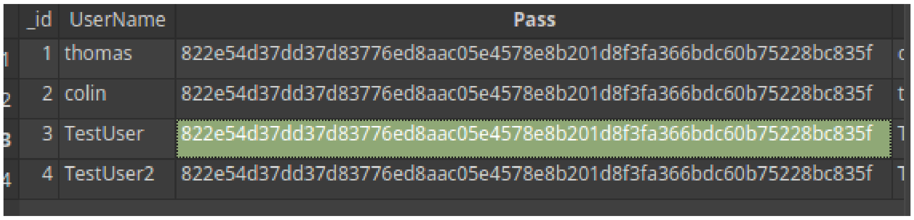
Past Interval 0
Past Average 100
Attempts 1
New Interval
New Average
Past Level 🟩 Improved 0
New Level ■ Declined 3

Minimum Score: 80

☑ Hints

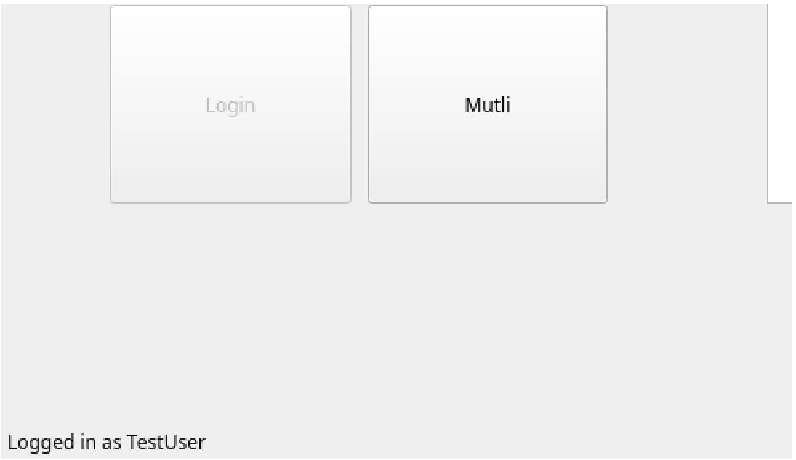
Score:

2.8.1.34 Client registration works, hashed password



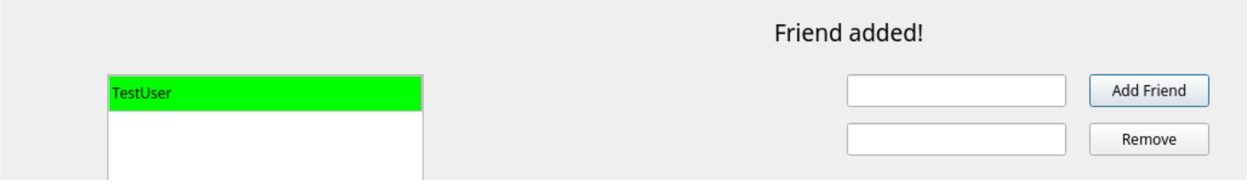
_id	UserName	Pass
1	thomas	822e54d37dd37d83776ed8aac05e4578e8b201d8f3fa366bdc60b75228bc835f
2	colin	822e54d37dd37d83776ed8aac05e4578e8b201d8f3fa366bdc60b75228bc835f
3	TestUser	822e54d37dd37d83776ed8aac05e4578e8b201d8f3fa366bdc60b75228bc835f
4	TestUser2	822e54d37dd37d83776ed8aac05e4578e8b201d8f3fa366bdc60b75228bc835f

2.8.1.35 Client login works



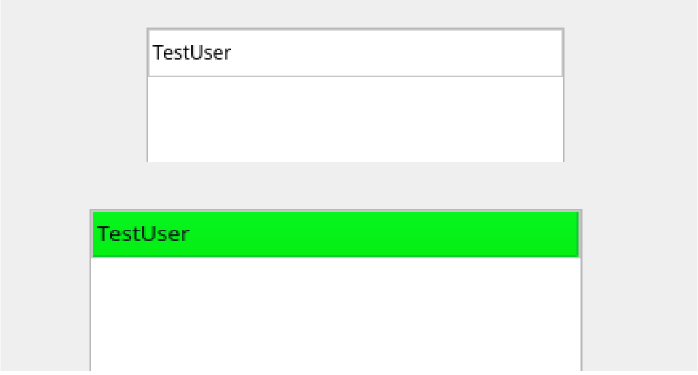
Logged in as TestUser

2.8.1.36 Client add friend works

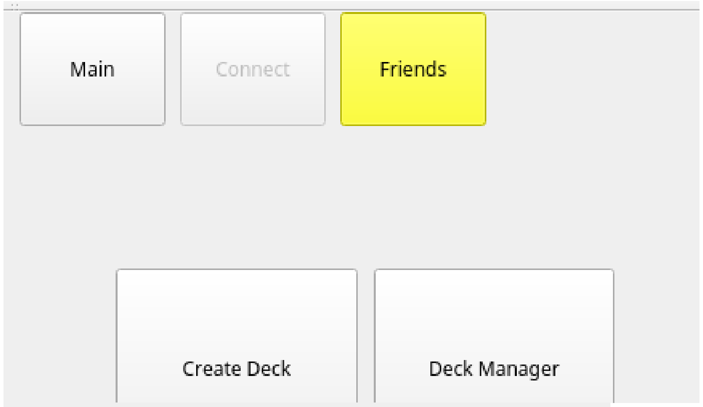


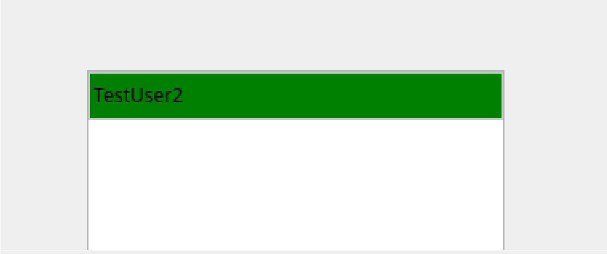
Friend added!

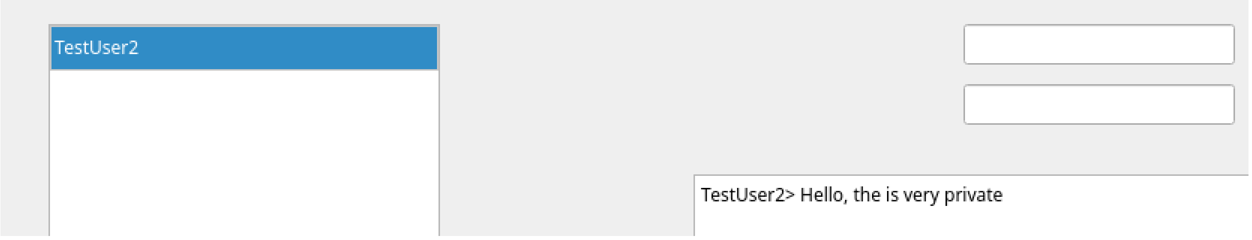
2.8.1.37 Client online friends show



2.8.1.38 Client – private message works

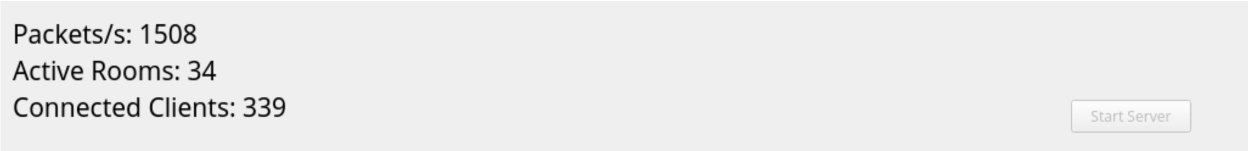






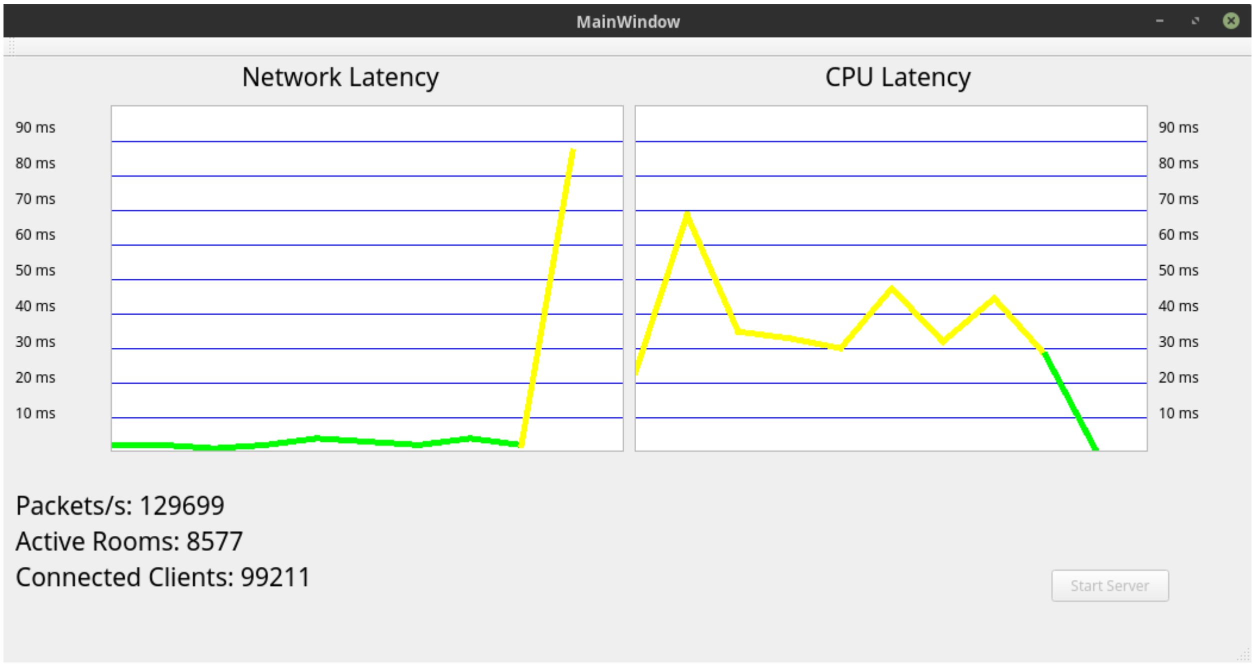
TestUser2> Hello, the is very private

2.8.1.39 Server – Room and user count updates



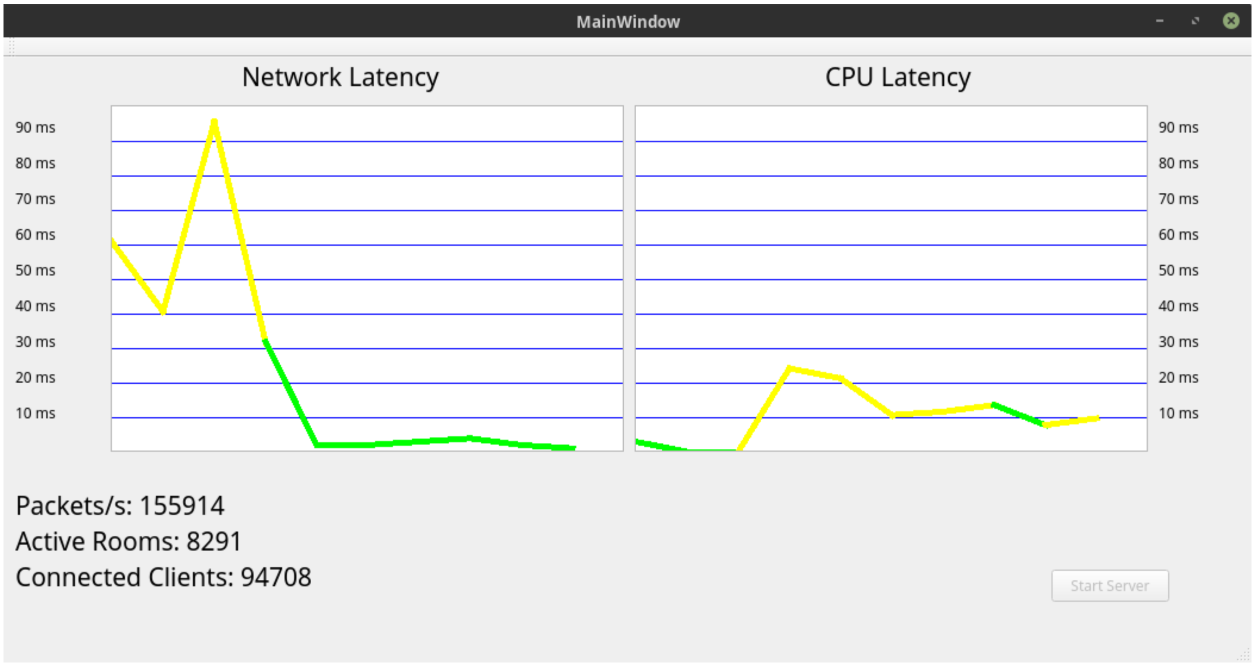
Packets/s: 1508
Active Rooms: 34
Connected Clients: 339

2.8.1.40 Server – handles large scale room creation and join



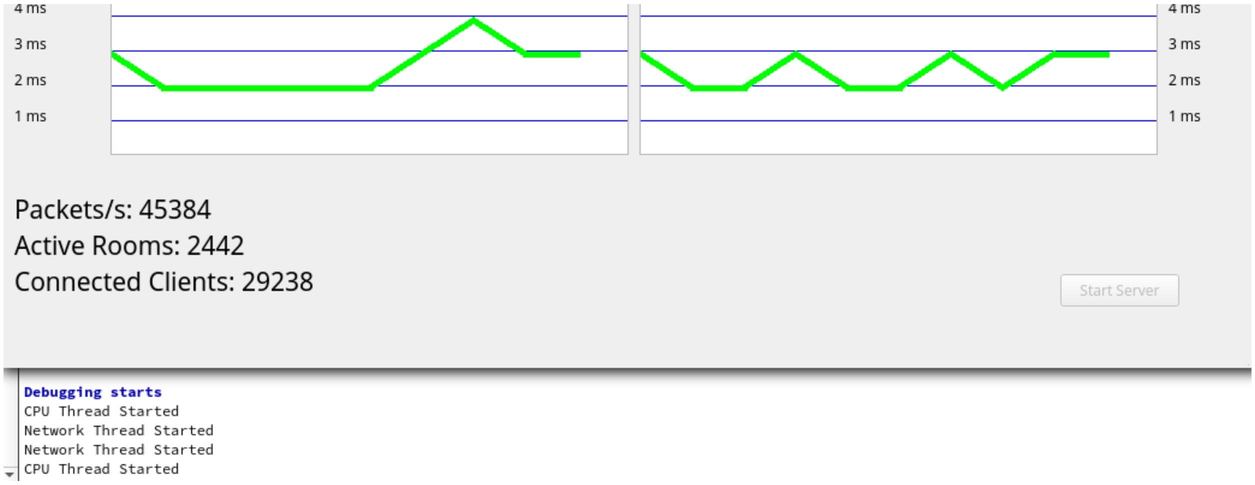
Packets/s: 129699
Active Rooms: 8577
Connected Clients: 99211

2.8.1.41 Server – Network and Cpu latency graphs update



Packets/s: 155914
Active Rooms: 8291
Connected Clients: 94708

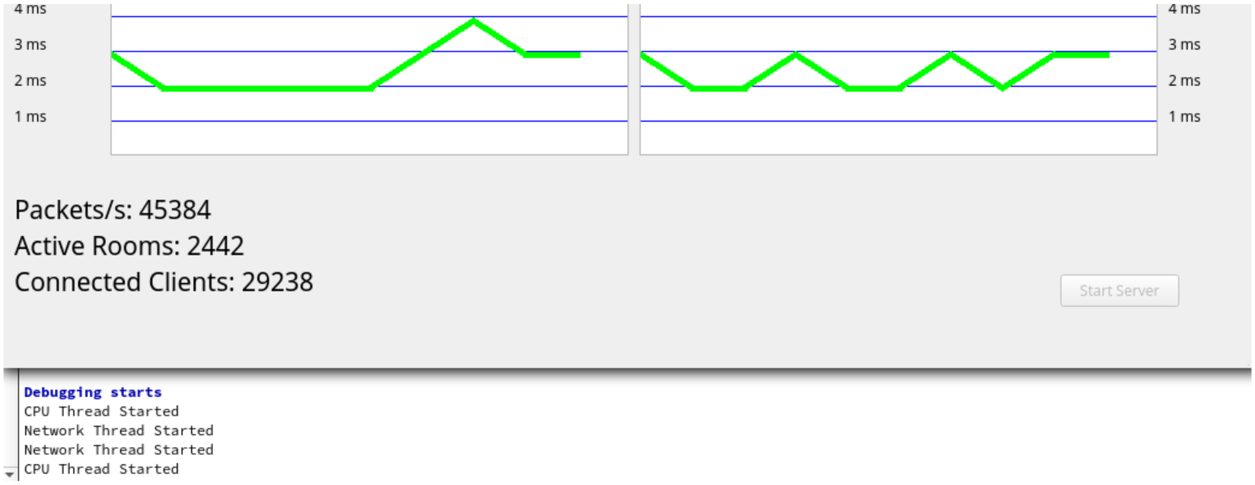
2.8.1.42 Server – Utility thread activates when event queue slow



Packets/s: 45384
Active Rooms: 2442
Connected Clients: 29238

```
Debugging starts
CPU Thread Started
Network Thread Started
Network Thread Started
CPU Thread Started
```

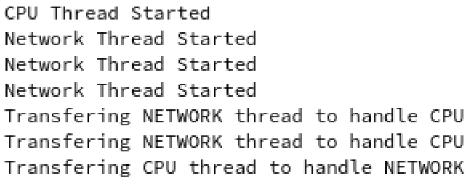
2.8.1.43 Server – Utility thread activates when network queue slow



Packets/s: 45384
Active Rooms: 2442
Connected Clients: 29238

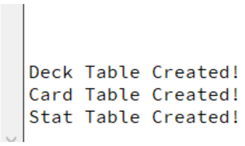
```
Debugging starts
CPU Thread Started
Network Thread Started
Network Thread Started
CPU Thread Started
```

2.8.1.44 Server – Utility threads transfer from network to event when needed



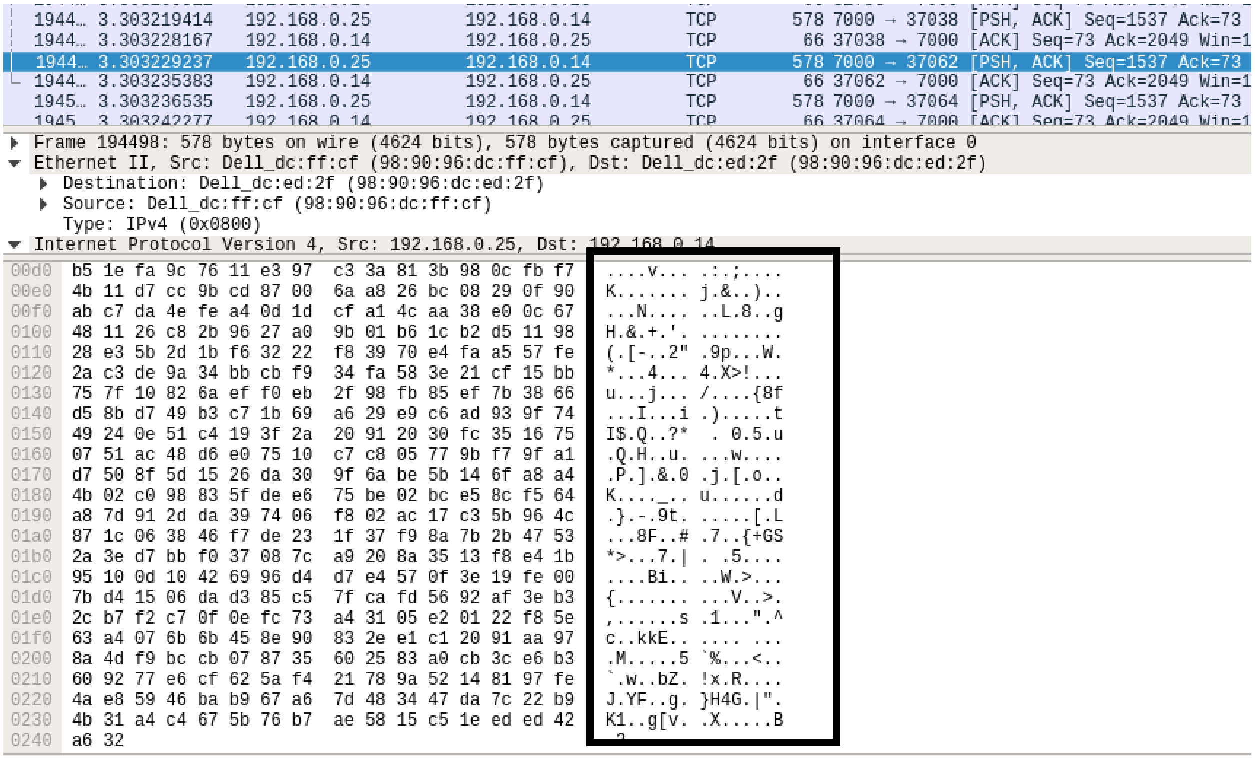
```
CPU Thread Started
Network Thread Started
Network Thread Started
Network Thread Started
Transfering NETWORK thread to handle CPU
Transfering NETWORK thread to handle CPU
Transfering CPU thread to handle NETWORK
```

2.8.1.45 Database setup on first time run



```
Deck Table Created!
Card Table Created!
Stat Table Created!
```

2.8.1.46 Security – All user communications encrypted

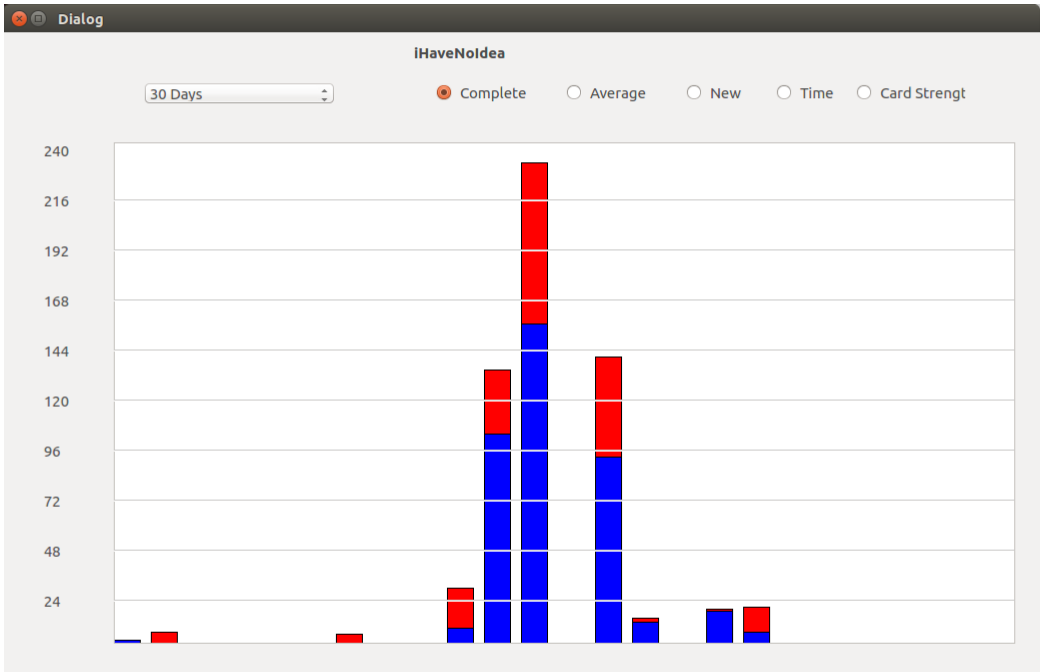


```
1944… 3.303219414   192.168.0.25   192.168.0.14   TCP   578 7000 → 37038 [PSH, ACK] Seq=1537 Ack=73
1944… 3.303228167   192.168.0.14   192.168.0.25   TCP    66 37038 → 7000 [ACK] Seq=73 Ack=2049 Win=1
1944… 3.303229237   192.168.0.25   192.168.0.14   TCP   578 7000 → 37062 [PSH, ACK] Seq=1537 Ack=73
1944… 3.303235383   192.168.0.14   192.168.0.25   TCP    66 37062 → 7000 [ACK] Seq=73 Ack=2049 Win=1
1945… 3.303236535   192.168.0.25   192.168.0.14   TCP   578 7000 → 37064 [PSH, ACK] Seq=1537 Ack=73
1945… 3.303242277   192.168.0.14   192.168.0.25   TCP    66 37064 → 7000 [ACK] Seq=73 Ack=2049 Win=1
```

```
▸ Frame 194498: 578 bytes on wire (4624 bits), 578 bytes captured (4624 bits) on interface 0
▾ Ethernet II, Src: Dell_dc:ff:cf (98:90:96:dc:ff:cf), Dst: Dell_dc:ed:2f (98:90:96:dc:ed:2f)
   ▸ Destination: Dell_dc:ed:2f (98:90:96:dc:ed:2f)
   ▸ Source: Dell_dc:ff:cf (98:90:96:dc:ff:cf)
     Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 192.168.0.25, Dst: 192.168.0.14
```

```
00d0  b5 1e fa 9c 76 11 e3 97  c3 3a 81 3b 98 0c fb f7   ....v... .:.;....
00e0  4b 11 d7 cc 9b cd 87 00  6a a8 26 bc 08 29 0f 90   K....... j.&..)..
00f0  ab c7 da 4e fe a4 0d 1d  cf a1 4c aa 38 e0 0c 67   ...N.... ..L.8..g
0100  48 11 26 c8 2b 96 27 a0  9b 01 b6 1c b2 d5 11 98   H.&.+.'. ........
0110  28 e3 5b 2d 1b f6 32 22  f8 39 70 e4 fa a5 57 fe   (.[-..2" .9p...W.
0120  2a c3 de 9a 34 bb cb f9  34 fa 58 3e 21 cf 15 bb   *...4... 4.X>!...
0130  75 7f 10 82 6a ef f0 eb  2f 98 fb 85 ef 7b 38 66   u...j... /....{8f
0140  d5 8b d7 49 b3 c7 1b 69  a6 29 e9 c6 ad 93 9f 74   ...I...i .).....t
0150  49 24 0e 51 c4 19 3f 2a  20 91 20 30 fc 35 16 75   I$.Q..?*  . 0.5.u
0160  07 51 ac 48 d6 e0 75 10  c7 c8 05 77 9b f7 9f a1   .Q.H..u. ...w....
0170  d7 50 8f 5d 15 26 da 30  9f 6a be 5b 14 6f a8 a4   .P.].&.0 .j.[.o..
0180  4b 02 c0 98 83 5f de e6  75 be 02 bc e5 8c f5 64   K...._.. u......d
0190  a8 7d 91 2d da 39 74 06  f8 02 ac 17 c3 5b 96 4c   .}.-.9t. .....[.L
01a0  87 1c 06 38 46 f7 de 23  1f 37 f9 8a 7b 2b 47 53   ...8F..# .7..{+GS
01b0  2a 3e d7 bb f0 37 08 7c  a9 20 8a 35 13 f8 e4 1b   *>...7.| . .5....
01c0  95 10 0d 10 42 69 96 d4  d7 e4 57 0f 3e 19 fe 00   ....Bi.. ..W.>...
01d0  7b d4 15 06 da d3 85 c5  7f ca fd 56 92 af 3e b3   {....... ...V..>.
01e0  2c b7 f2 c7 0f 0e fc 73  a4 31 05 e2 01 22 f8 5e   ,......s .1...".^
01f0  63 a4 07 6b 6b 45 8e 90  83 2e e1 c1 20 91 aa 97   c..kkE.. .... ...
0200  8a 4d f9 bc cb 07 87 35  60 25 83 a0 cb 3c e6 b3   .M.....5 `%...<..
0210  60 92 77 e6 cf 62 5a f4  21 78 9a 52 14 81 97 fe   `.w..bZ. !x.R....
0220  4a e8 59 46 ba b9 67 a6  7d 48 34 47 da 7c 22 b9   J.YF..g. }H4G.|".
0230  4b 31 a4 c4 67 5b 76 b7  ae 58 15 c5 1e ed ed 42   K1..g[v. .X.....B
0240  a6 32
```

2.8.1.47 Statistics – Cards done per day tracked

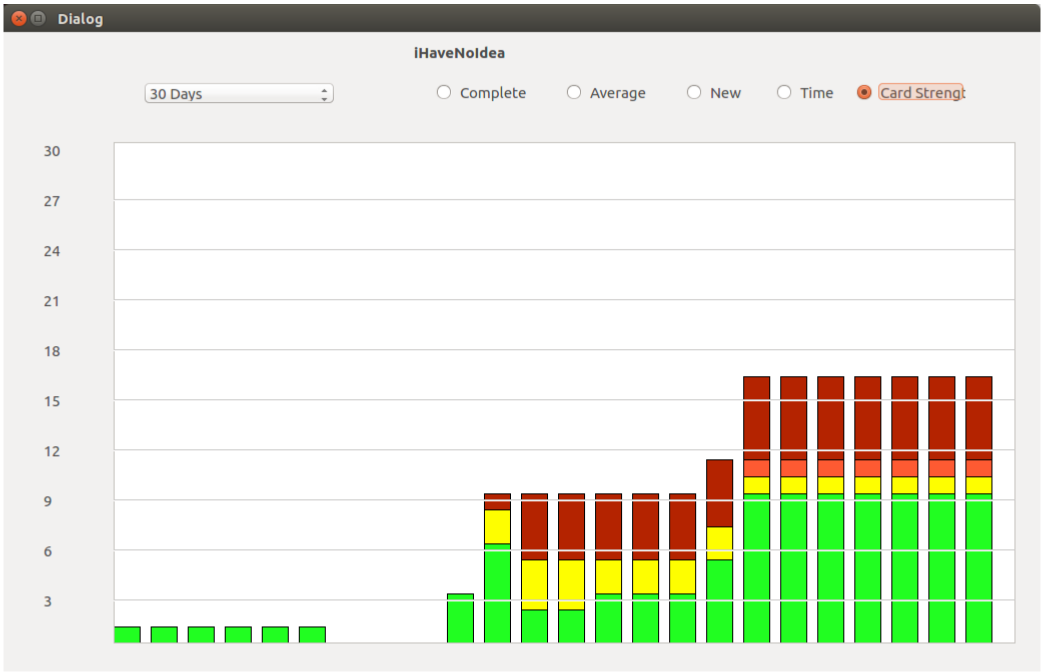


iHaveNoIdea

30 Days

● Complete ○ Average ○ New ○ Time ○ Card Strengt

2.8.1.48 Statistics – Card level tracked



iHaveNoIdea

30 Days

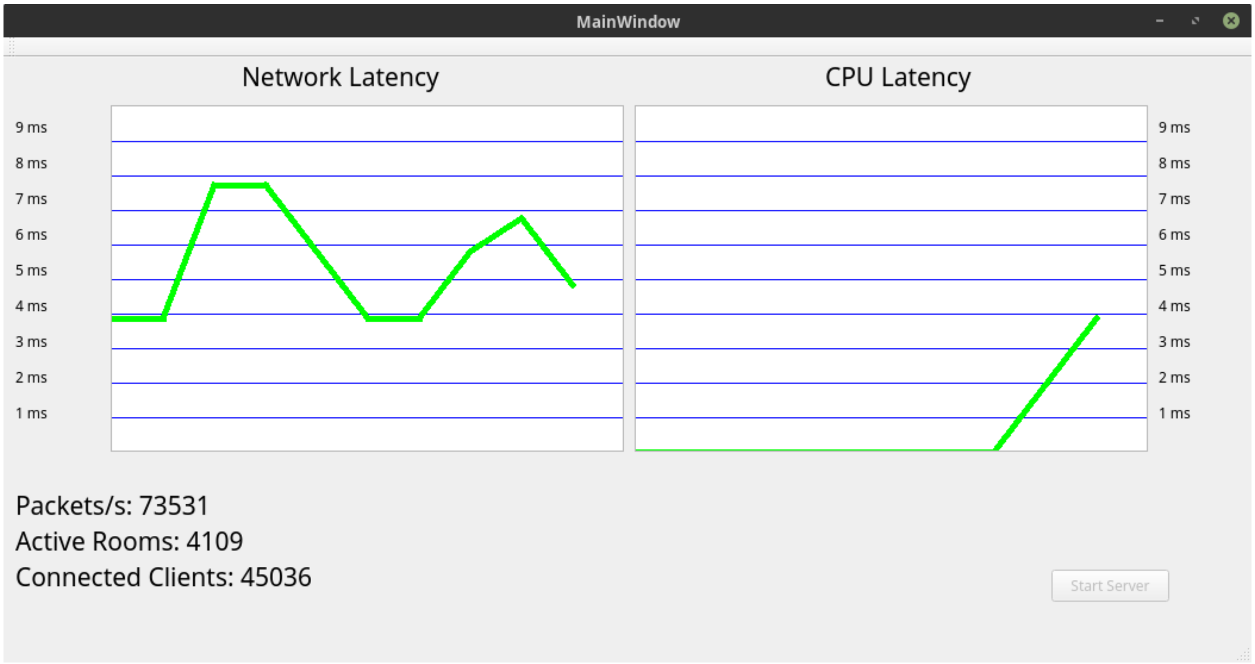
○ Complete ○ Average ○ New ○ Time ● Card Strengt

2.7 Load Testing

As mentioned previously, one of the concerns for this project was to implement the server in such a way that it could handle a significant user base. In order to test this, the FlashyTester program was developed as a means of simulating user traffic. By running multiple instances of the testing program on different computers, the server could be sufficiently stress tested.

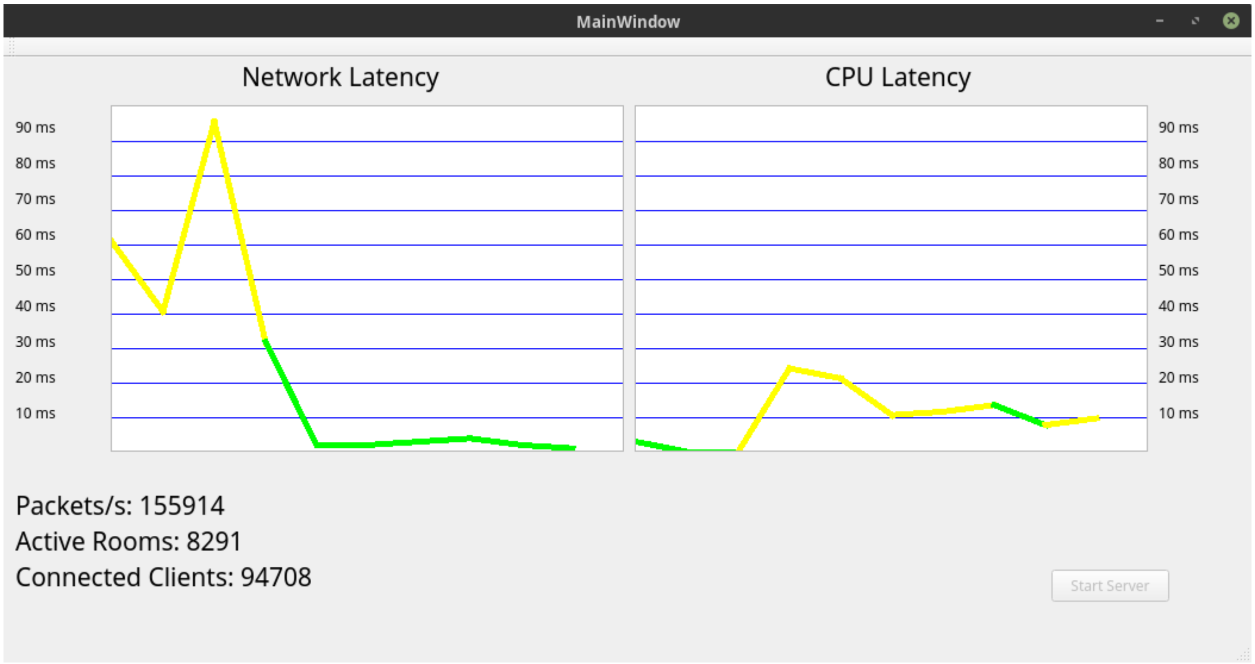
One issue with the testing program that I was not able to solve was the ability to join rooms quickly. It is still able to join hundreds of users per second, but this is a slowed down rate. This is not an issue with the test itself, as 100000 users connecting and joining study rooms in a minute's time is not a realistic real life use case anyways. However, by slowing the process down it became less feasible to connect 300000 or 400000 users due to the length of time it would take.

The solution for this was to simply triple, or more, the amount of traffic each user generates. This was done by setting the response delay for each card to 1 second. While this does mean the server has to handle fewer users, it is the traffic and event handling that consumes resources. Epoll performs socket monitoring in O(1), and large scale session management on the server such as finding the corresponding session for each user request is also O(1). This means that adding users does not add significant stress to the server, but adding traffic does. The 100000 users of the test, performing study operations at 3 times a maximum human rate, is essentially able to stress the server in the same was as 300000 users studying at a more normal rate.



Packets/s: 73531
Active Rooms: 4109
Connected Clients: 45036

The above image shows the middle of the stress test, as 45000 clients were studying. This number, again, is simulated users working at triple the speed, or more, of the normal human use case, and so could be thought of as closer to 150000 normal users. The term 'Network Latency' in the above graph refers to the amount of time between a read event first being posted and the completion of the read and decrypting of the corresponding data. The term 'CPU Latency' measures the time from the network handler registering an event and the event handler completing the handling of said event. These two numbers combined represent the total delay from a packet arriving at the server to the response leaving the server. In the above picture, ~10ms is an acceptable level for an application such as FlashyCard.



Packets/s: 155914
Active Rooms: 8291
Connected Clients: 94708

The above picture shows traffic near the end of the load test. At this point in the test peaks of 100ms such as the one pictured were not uncommon, and the total processing time was becoming significant – ranging between 50 and 150ms. Although FlashyCard is not interactive in the same way as a networked video game, for example, the above begins to represent the limits for how many users the system can handle without noticeable performance drops. As mentioned above, the 155000 packets handled per second would more reasonably be generated by a user base of 250-400 thousand.

In conclusion, although the load testing program did not work exactly how I wished, I was still able to successfully stress test the server and reach a point of significant performance delay. The amount of traffic handled by the server suggests that the original goal of creating a server able to handle a large user base was accomplished.

2.8 Implications of Implementation

The implementation of FlashyCard answered one of the original questions – whether a networked flashcard study application could would. At a minimum, the implementation allows for users with similar cards to study together in a way that is not significantly clunky or slow when compared to the normal single user study system.

The second question of what makes a good language learning application, and whether the innovations in the study system of FlashyCard have the desired effect required a usability study to attempt to answer.

2.8.1 Usability Study

2.8.1.1 Study Goal

The goal of this study is to observe if the innovative study systems implemented in FlashyCard have the desired effect of increasing memory retention in the learning of a new language.

2.8.1.2 Study Design

A key in designing the study was to keep in mind the reality of what was possible to actually carry out. Ideally, learning a combination of vocabulary and sentences over a several week period would have provided better insight into the effectiveness of FlashyCard, especially given its focus on grammar learning. However, the reality of available participants and their available time to spend on the study limited the study design to the following:

FlashyCard was compared with two popular flashcard applications – Anki and Quizlet

Participants were divided into groups by desired language. All participants chose languages that they had no prior experience with. These groups were further divided by application, with each person using one of the three applications. The selection of which participant used which application was done at random, to prevent personal knowledge of the participants from influencing which application each received.

A custom vocabulary deck of 100 terms was created for each group and added to their application. Both lists were created from available vocabulary lists, with some modification to remove words containing non-English characters. This was done out of convenience for the participants, so they would not be required to download a new keyboard layout and learn the new characters. Each user studied daily for a period of 1 week.

Following the week of studying, an exam was completed by each participant. This exam was conducted using a custom build of FlashyCard. The custom build of FlashyCard simplified the study screen to only display the card prompt and an input line for the user's response. The simplicity of the exam should remove any advantage users of FlashyCard had gained through familiarity with the application.

The format for the exam was 50 vocabulary terms semi-randomly chosen from the original 100. Terms were chosen 10 at a time from sections of 20 words. That is, 10 of the first 20 words introduced were chosen, then 10 of the next 20 that had been introduced. This was done to give an even distribution of words over the whole set. Each group of participants received the same 50 words and there was no time limit put on the exam. The exam measured two statistics – overall accuracy and number of cards completed with 100% accuracy.

Participants completed the exam on their own time, and were encouraged not to cheat in any way. It is possible that this was ignored, but given their understanding of why they were conducting the test, the lack of reward for good results, and the absence of any nearly perfect scores, the following results are likely valid.

2.8.1.3 Study Results

Group 1 – Language: Spanish

Particpant	Application	Average Score	Correct Words
Participant 1	Anki	56	15
Participant 2	FlashyCard	78	22
Participant 3	Quizlet	74	23

Group 2 – Language: German

Particpant	Application	Average Score	Correct Words
Participant 4	Anki	61	16
Participant 5	FlashyCard	68	19
Participant 6	Quizlet	71	21

2.8.1.4 Study Concussions

In order to analyze the results of the study, an analysis of variance (ANOVA) model was used. The results of this are as follows:

Application	Anki	FlashyCard	Quizlet	Total
Samples	2	2	2	6
Mean	58.5	73	72.5	68
Sum	117	146	145	408
Sum of Squares	6857	10708	10517	28082
Variance	12.5	50	4.5	67.6
Standard Deviation	3.5355	7.0711	2.1213	8.2219
Standard Error	2.5	5	1.5	3.3566

The above values were used to fill in the standard ANOVA table.

	Degrees of Freedom	**Sums of Squares**	**Mean Squares**	**Variance Ratio**	**P Value**
Between Groups	2	271	135.5000	6.0672	**0.0883**
Within Groups	3	67	22.3333		
Total	5	338			

The key value of the above table is the P value, which is 0.0883. In order to conclude that there was a significant difference between the groups, this value would have to be less than 0.05. As it was not, the conclusion to be drawn is that there was no statistically significant difference between the three applications.

2.9 Innovation

FlashyCard features two main innovative components.

The first of these components is the study system, which feels lacking in many of the current applications. There are several areas of innovation within the study system, the first being the scoring system - or how users prove their knowledge. There are various designs for this in other applications such as text input, multiple choice inputs, or 'Select your level of knowledge' input. For FlashyCard, with its focus both on vocabulary and grammar, text input seemed the most effective. The improvement that FlashyCard made to the current applications that use text input is to more accurately evaluate the input, scoring on a 0-100 system rather than a 0 or 100 system. Using the more accurate measurement of the user's understanding of a given card allowed for a better scaling of how frequently cards were shown.

The second innovate component in the study system is to provide better assistance for studying sentences. Current applications generally do not offer hints, and if they do they must be user defined. While this system works, it is a time consuming exercise to manually add hints to 4000 cards. While this system works fine for vocabulary, it is limited when dealing with sentences, especially in applications that require text input. An application that requires perfect text input on a 10 word sentence, with no guidance on word length or arrangement, is a difficult task for someone new to the language. In addition, possible ambiguity both in word order and word choice further complicate the process and leave the user to struggle on what exactly the translation should be. The idea behind FlashyCard's sentence hint system is to parse the grammar out of sentences. The effectiveness of this system is somewhat language dependant, and languages that feature significant grammar characters benefit more. An example from Korean for which this system works well is '차**가** 있**었으면** 좋**겠어요**'. The red characters in the previous sentence represent grammar particles. By splitting the sentence into grammar and vocabulary parts, it is possible to introduce hints in a staged delivery. In the first stage, the grammar portion of the sentence is provided as a hint for the user. This allows them to focus on the vocabulary of the sentence, which they are likely still learning, while still familiarizing themselves with the grammar concepts. The second stage reverses the hints, letting them focus on learning the grammar concepts while still providing a review for vocabulary. The final stage removes both hints and leaves them with only word length as a hint, giving some slight guidance as to word order and choice.

The second major innovate component to FlashyCard is the networked study portion. This is a feature not offered by any of the major flashcard applications and it's something that could offer significant benefits. A study of group work in a classroom setting for second language learning concluded that "in addition to strong pedagogical arguments, there now exists a psycholinguistic rationale for group work in second language learning." (Long & Porter, 1985). While there is a difference between the classroom setting and group flashcard study, there is reason to believe that some of the benefits will still apply.

In FlashyCard there are two multiplayer study methods. The first is a synced study, where each user studies the same card in a timed round. Answers are collected and points rewarded for performance to provide motivation. The second method is an independent, but grouped, study. This allows users to study different cards at their own pace, with a count of the number of completed cards by each user as motivation. In both settings, the goal is to increase users motivation to study and keep them focussed on the application. Another benefit present in both study methods is the ability to chat with one another. For users with little exposure to the language they are learning it gives them a group of people to practice the language with. It also serves as a way to share study materials, or study tips and strategies.

2.10 Complexity

There are several elements of complexity within the three parts of the overall FlashyCard project. As an overall project, the areas that required the most effort were not exactly those expected when writing the proposal. This seems to stem from paying more attention to how different features could be implemented and less attention to what exactly had to be handled by each feature and all the possible use cases and scenarios.

The complexity and issues with each portion of the project are as follows.

2.10.1 Client

The complexity of the client was mainly just in problem solving. Issues such as how to efficiently parse grammar out of sentences, how to accurately evaluate user input or how to parse varied file input into cards all required some effort to solve. Even seemingly simple things like managing the intervals at which cards are shown occasionally ended up requiring more effort than expected.

The client overall doesn't feature any groundbreaking new technology or techniques from the degree program, it is simply an exercise in good software development with a number of interesting problems to solve. Comparing it to my previous attempt at a similar application from the summer after the diploma, there is a clear difference in my ability now and then, and parts of the application that I did not have working back then now function as intended.

2.10.2 Server

The server is the main point of learning for this project, and it's where lessons learned during the degree were implemented. The server is an improvement on the scalable server project from COMP 8005, which was likely the largest project in terms of development hours for me during the degree program. Comparing that assignment to the server developed for the practicum, the complexity increased significantly. The system architecture became more complex, featuring more threads, different queue systems, and variable task threads. The improved message buffer system and event handling system were two further improvements on the original project, which was essentially an echo server.

Comparing the server now to what was done during the diploma is an even bigger change. Diploma client/server projects were simple things like a chat program that supported a handful of users. Even the class wide networked game project had issues handling a simple game lobby and managing the 20 users involved.

There are other aspects of the project where my knowledge has slowly improved during the degree. Something like the encryption implemented in FlashyCard, which is a personal version that I have slowly improved throughout the degree, was improved again with influence from the most recent COMP 7481 course.

2.10.3 Tester

The tester, just like the server, is an improvement on the scalable server project which also featured a client for testing. In that case, it sent random strings of data and simply read and discarded any response. The tester for FlashyCard featured the improved server architecture from the scalable server, and pulled some of the event handling out of the client to create a test program that more accurately mimics real traffic, and can be modified to test different use cases for the server.

2.11 Future Enhancements

All the major features that were planned have been implemented into the current version, and so there is limited future enhancement. Things like cleaning up the user interface, adding card sync so users can study across different devices, and adding more specific language modules are some possible minor improvements.

In terms of possible system changes, there may be options in terms of making the network study system more user friendly. The current round system keeps everything moving at a decent pace, but there may be alternative options available.

2.12 Timeline and Milestones

As discussed in the proposal, development was split up into a number of sprints with specific milestones to be completed by the end. The actual timeline differed fairly significantly from the proposed, with the scope of some of the multiplayer features being underestimated. This was balanced out by some of the client functionality being easier than expected. The order in which tasks were completed also changed slightly, so the following table will be divided up into major categories rather than the originally proposed sprints.

Component	Feature	Completion	Hours
Project Documentation	Proposal	Feb 13 -Mar 4	10
	Report	Apr 30	30
	Usability Study – Implementation, monitoring	April 25	10
Subtotal			**50**
Client	Design – Database, single user features	Feb 15	15
	Database implementation	Feb 16	5
	Card import/export, deck creation, management	Feb 20	15
	Card management, study session basics	Feb 22	10
	Scoring and grammar parsing	March 8	25
	Statistics tracking	March 9	5
	Testing – all non-networked features	March 11	10
	MILESTONE – Non networked functionality complete		
	Design – networking, message format, events	March 17	5
	Implement network code, event handling	March 18	5
	Import/Export deck over network,	March 24	10
	Networked study lobby functionality	March 28	5
	Create/Join networked study	March 30	5
	Networked study implementation	April 10	15
	Networked game implementation	April 12	5
	Registration, login	April 13	3
	Encryption added to network code	April 14	2
	Testing – networked features	April 20	10
Subtotal			**150**

Server	Design – overall architecture, database	March 13	15
	Epoll implementation, message buffer, event queue	March 18	15
	Database – implement database, basic functions	March 20	5
	Deck import/export handling	March 24	5
	Study session creation, basic management	March 28	15
	Study session matching, response to available session request	April 2	15
	All networked study management	April 10	50
	Multiplayer review game	April 12	10
	User registration, friends system, private messaging		10
	Rewrite encryption, add to network code	April 14	5
	Testing – all major networked use cases	April 20	15
	MILESTONE – Major networked use cases complete		
	Timing tracking, thread safe concerns	April 21	10
	Testing – load testing, server optimization	April 24	10
Subtotal			**180**
Tester	Design – overall system	April 21	5
	Epoll implementation, event queue, message buffer	April 22	5
	Event handling, basic join and study functionality	April 24	15
	Testing	April 25	5
Subtotal			**30**
Grand Total			**410**

3 Conclusion

This practicum gave me great experience on a larger scale project than I have ever worked on before, and made clear the importance of good design and planning. The number of hours and abundance of challenging features implemented helped me improve as a software developer. Finally, the networked and server portion of the project furthered knowledge from previous COMP courses greatly.

3.1 Lessons Learned

Some of the main things learned during this project are:

- Improved epoll implementation and server architecture
- Better thread management
- More accurate server load tracking and performance monitoring

In addition to the major lessons learned above, there were several other takeaways from the project. The first of these was scheduling. During the start of the project I took tasks from the schedule in the proposal and began working on them. I quickly realized that despite the original schedule being fairly detailed, it was still not enough. This realization lead me to break down each task into more detailed lists of exactly what needed to be done to complete even simple sounding tasks. Doing this improved productivity and eliminated time spent wondering what to work on next.

The second thing learned is similar to the scheduling, and is to not overlook the details of implementation. I spent a lot of effort on figuring out the overall design of the client and server and not enough on the details of things like the multiplayer study system, which turned out to involve a ton of possible cases that needed to be handled.

3.2 Closing Remarks

This practicum has provided a great opportunity to improve my skills as a software developer, and has greatly increased my knowledge of server implementation and management. Overall, the practicum was an enjoyable experience.

4 References

Epoll basics - https://eklitzke.org/blocking-io-nonblocking-io-and-epoll

Long, M., & Porter, P. (1985). Group work, interlanguage talk, and second language acquisition. TESOL Quarterly, 19, 207–228.

5 Change Log

- March 26, 2018 – Initial draft

6 Appendix

6.1 Pseudo Code

6.1.1 Client

Below is the pseudo code for the main client use case – single player study

Load Cards Due

```
    Take selected deck as input
    Call to data manager to load cards due today
    GO TO Data Manager
    GO TO Create Study Session
```

Data Manager

```
    Query card table for any cards due on or before current date
    Ignore cards that have been set inactive in deck management
    RETURN Card List
```

Create Study Session

```
    Pass list of due cards to study session manager
    Reset basic study session variables
    GO TO Load Grammar Parser
    GO TO Get Next Card
```

Load Grammar Parser

```
    Scan current card list for card languages
    GO TO Hash Dictionary Files
```

Hash Dictionary Files

```
    Read files for each language studied in current session
    Hash each portion of the word - Test -> 'T', 'Te', 'Tes', 'Test'
```

Get Next Card

```
    Get current time
    Iterate through card list
    IF Card with next review time < current time
        Select card
    ELSE
        Select card with minimum time until next review
    GO TO Display Front of Card
    IF Hint checkbox is active
        GO TO Generate Hint
```

Display Front of Card

```
    Clear basic GUI elements
     Display front of card
```

Generate Hint

```
    Check card language
    Parse back side of card - answer side
    IF word count of answer > 1
        IF card level == 2
            Return basic word length hint - '-' character in place of each character in the
            answer
        ELSE
```

```
                GO TO Parse Grammar
    ELSE
        Replace each character with '-' character for simple word length hint
    GO TO Parse User Supplied Hint
    GO TO Display Hint
```

Parse Grammar

```
    Iterate through each part of each word - same as hashing grammar list
    Check each part of word against hash table
    IF word found
        Ignore character, add '-' to hint string
    ELSE
        Add current character to hint string

    IF card level == 1
        Reverse string -> Grammar hints replaced with '-', vocab characters replace current '-'
        characters
```

Parse User Supplied Hint

```
    Iterate each character in user supplied hint
    Iterate each character of back of card - answer
    IF characters match
        Add character to hint string
```

Display Hint

```
    Display hint for the user
```

Receive Answer

```
        Read in answer input

        GO TO Evaluate Answer
```

Evaluate Answer

```
  IF word count of answer > 1

        Iterate through each character of back of card

        Iterate through each character of supplied answer

        IF current index in hint string != '-'

                Continue

       IF character match

                Increment correct character counter

                Set found character in answer string to garbage character - no double matching to
                another part of string

        ELSE

                Increment incorrect character counter

        ELSE

        IF Korean card

                Evaluate with Korean module

        ELSE

                Match character and position

        Calculate score - right / right+wrong

        GO TO Update Card
```

Update Card

```
        Calculate new card average - average grade over every attempt
```

```
    Calculate new interval
    Check card level change
    Send new card information to Data Manager
    IF card score < minimum required
        GO TO Return Card to Session
    ELSE
        GO TO Remove Card
    IF card list is empty
        GO TO Session Complete
    ELSE
        GO TO Get Next Card
```

Data Manager

```
    Update CardTable with supplied card number, interval and average grade
```

Return Card to Session

```
    Increment next review time depending on level of error - worse score = review sooner
```

Remove Card

```
    Remove current card from session card list
```

Session Complete

```
    Cleanup GUI
    Cleanup study session manager
    Cleanup grammar tables
    Return to Main Menu
```

6.1.2 Server

Below is the pseudo code for the main server use case – multiplayer study

Receive Room Creation Event

```
    Event handler catches Room Creation Event
    Parse request – user, deck, games, independent study variables
    GO TO Parse Card List, Create Session
```

Parse Card List, Create Session

```
    Card list received in bit form – last 7 bits of each character
    Iterate through card list portion of packet
    Iterate through each bit of each character of list
    IF bit is set
        Set bit in current card list
    ELSE
        Continue
    Add session to current session list
    GO TO Get Next Card
```

Get Next Card

```
    IF card list is empty
        GO TO Session Complete
    Iterate through card list
    IF next review time < current time
        Select card
```

```
ELSE
    Select card with minimum next review time
Add next card to prepared packet
GO TO Send Card
GO TO Register Output Event
GO TO Wait for Responses
```

Send Card

```
Iterate through user list for current session
Send packet to each user
```

Register Output Event

```
Create output event
Set outgoing time, current study round
```

Wait for Responses

```
IF multiplayer answer event triggered
    GO TO Receive Answer
ELSE IF User Leave event triggered
    GO TO User Leaves Session Event
```

Receive Answer

```
Update current session round responses
Update current session users removing card
IF round responses == required responses for round
```

```
        GO TO Update Session
```

Update Session

```
    Iterate through scores received current round
    Assign points to each user
    IF more than half the users removing the card
        Remove card from session
    Increment current round
    GO TO Get Next Card
```

Round Timeout Reached

```
    Thread polling output events reaches previously registered end of round event
    IF study session current round == event round
        GO TO Update Session
        GO TO Send Output Event
```

Send Output Event

```
    Send next card packet to all users in current session
```

User Leaves Session Event

```
    Update session user list
    IF no users left in current session
        GO TO Session Complete
    Decrement required round responses
    IF current round response == required responses
```

```
        GO TO Update Session

    Send user leave message to each user in session



Session Complete

    Clean up session variables, lists
```

6.1.2 Tester

Below is the pseudo code for the main use case of the tester – load testing

```
Connect Socket

    Loop for desired number of connections

    Connect to server

    GO TO Request Room List


Request Room List

    Set room list request packet – current due cards, deck id, etc

    GO TO Connect Socket


Event Handler

    Handle incoming events

    IF event = Next Card

        GO TO Next Card Received

    IF event = Room List

        GO TO Room list received


Room List Received
```

```
    Parse room list
    IF at least one room exists
        GO TO Join Session
    ELSE
        GO TO Create New Session

Join Session
    Select first available room
    Craft join session request
    Send request

Create New Session
    Craft create room request
    Send request

Next Card Received
    Parse round number
    GO TO Register Output Queue Event

Register Output Queue Event
    Randomize send time
    Set round number
```

6.2 Interval Progression System

Below is a table showing the interval progression based on average result.

The far left column values, in dark red, are the average grade for the card throughout its lifetime – meaning a consistent score of that value.

Each blue column represents successive rounds. Each card starts with an interval of 1, then progresses to 3, 5, etc.

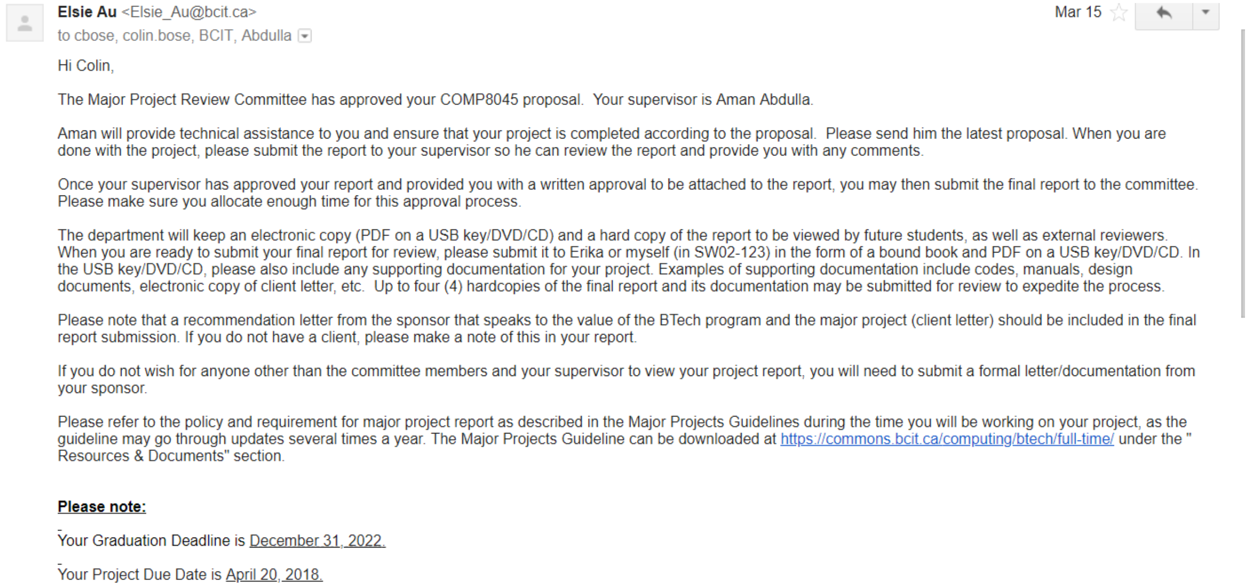
The default maximum interval for a card is 30 day. The point at which various scores hit this value is highlighted in red.

100	1	3	5	8	12	17	24	34								
95	1	3	5	8	12	17	23	31								
90	1	3	5	8	11	15	20	26	33							
85	1	3	5	7	10	13	16	20	24	29	35					
80	1	3	5	7	9	11	13	16	19	22	25	29	33			
75	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
70	1	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
65	1	2	3	4	5	6	7	8	9	9	9	9	9	9	9	9
60	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6
55	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
50	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

The interval formula is as follows:

> Past average score * 1.35 * + Log(100-past average)*0.3 – Log((100-currents score)^2)/5 * past interval

6.3 Proposal Approval



Elsie Au <Elsie_Au@bcit.ca>

to cbose, colin.bose, BCIT, Abdulla

Mar 15

Hi Colin,

The Major Project Review Committee has approved your COMP8045 proposal. Your supervisor is Aman Abdulla.

Aman will provide technical assistance to you and ensure that your project is completed according to the proposal. Please send him the latest proposal. When you are done with the project, please submit the report to your supervisor so he can review the report and provide you with any comments.

Once your supervisor has approved your report and provided you with a written approval to be attached to the report, you may then submit the final report to the committee. Please make sure you allocate enough time for this approval process.

The department will keep an electronic copy (PDF on a USB key/DVD/CD) and a hard copy of the report to be viewed by future students, as well as external reviewers. When you are ready to submit your final report for review, please submit it to Erika or myself (in SW02-123) in the form of a bound book and PDF on a USB key/DVD/CD. In the USB key/DVD/CD, please also include any supporting documentation for your project. Examples of supporting documentation include codes, manuals, design documents, electronic copy of client letter, etc. Up to four (4) hardcopies of the final report and its documentation may be submitted for review to expedite the process.

Please note that a recommendation letter from the sponsor that speaks to the value of the BTech program and the major project (client letter) should be included in the final report submission. If you do not have a client, please make a note of this in your report.

If you do not wish for anyone other than the committee members and your supervisor to view your project report, you will need to submit a formal letter/documentation from your sponsor.

Please refer to the policy and requirement for major project report as described in the Major Projects Guidelines during the time you will be working on your project, as the guideline may go through updates several times a year. The Major Projects Guideline can be downloaded at https://commons.bcit.ca/computing/btech/full-time/ under the " Resources & Documents" section.

Please note:

Your Graduation Deadline is December 31, 2022.

Your Project Due Date is April 20, 2018.

6.4 Supervisor Report Proposal

In Progress

6.5 Approved Proposal

Flashy Card

COMP 8045 – Major Project 1

Colin Bose – A00900656

5-1-2018

Table of Contents

1. Student Background

Colin Bose is a fourth year student at BCIT and is currently enrolled in the Bachelor of Technology program in the Network Security option. Prior to entering the degree program he completed the Computer Systems Technology diploma, having specialized in Data Communications.

Education

(BCIT) Bachelor of Technology in Computer Systems 2016 – 2018

- Specialized in Network Security

- Maintains honors status (80% +)

(BCIT) Computer Systems Technology (CST) Diploma 2014 – 2018

- Specialized in Data Communications

- Graduated with honors (80% +)

2. Project Description

This project is for the half practicum, for which I will be working alone.

The aim of the project is to build a language learning application. The application will focus on learning using flash cards, and require users to test their knowledge through varying inputs. The application will be networked, allowing users to study with one another.

The project will consist of two parts, a client and server. The purpose of each is as follows:

Client: A desktop based application, the client will allow the user to download or create sets of flash cards. Using these flash cards, the user can study either by themselves, or search for users with similar cards to study with.

Server: The server half of the project allows users to group up and study together. It also manages things like registration, upload/download of cards, syncing of devices, etc.

3. Problem Statement and Background

The problem that FlashyCard hopes to solve is the task of learning a new language. For users who wish to learn new vocabulary, or practice grammar and sentence structure, it will provide a way to do so.

The idea of using flash cards or similar systems for language learning is not a new one and a number of applications currently exist. These applications (Anki, Quizlet, Cram, Flashcard+, StudyBlue, etc.) each provide slightly differing services, but none of these feel fully complete.

The main goals that this application will focus on are:

- Support for large volume learning.
- Interactive input methods
- Support for sentence learning
- Networked studying

The first of the problems is large volume learning. Anki is a good example of this, with the user working from a set of several thousand cards, and learning 10 or 20 new words each day. As a user builds familiarity with each card, it is shown less frequently. However, the progression takes time and an average daily study session may be 100-200 cards.

The second problem, interactive input methods, has two concerns. First, the application should have a way to prove your knowledge. Quizlet, Cram and others provide a way to do this through text input or matching. Second, studying should have some fun in it. The only application of those listed earlier that addresses this is Quizlet, through a timed matching game it offers.

Support for sentence learning is another focus for FlashyCard, and is something that all the previous applications lack, for one reason or another. Some applications, like Anki, don't offer the user any way to test their knowledge and input an answer. For the applications that do offer text input, it is generally evaluated as all or nothing. If the user makes a single mistake, the answer is counted as wrong and no credit given. The last issue is a lack of direction. If you were prompted in another language to create a sentence in English with the effect of asking someone for a book, there are dozens of ways to phrase it. "Can you pass me that book", "Can I have that", "Please pass the book", etc. Because the answer is tied to the flashcard, there is only one answer that it is looking for, so some direction and flexibility is needed.

The final point that FlashyCard will address is networked studying. Again, this is not something offered by any of the current applications. The goal will be to match users up with others who are studying the same material as them, and have them study it together.

I have built separate applications for Android and Windows that have started to address some of these problems, but they are largely incomplete. FlashyCard will take some specific solutions from these, such as general card management, but it will mainly be built from scratch, using the examples of what didn't work in the previous applications as guidelines for what not to do.

4. Complexity

There are a number of technical challenges involved with this application. The most notable of these are:

- Scalability – The application will seek to provide service for a large number of concurrent users. Ensuring it responds well with 10000, or 50000, users will likely be challenging.
- Networked studying – Providing the user a fast and easy way to find other users studying the same cards as them and connecting. With thousands of users, there will be thousands of study sessions that must be managed.
- Usability of UI – The application will have a number of different functions, all of which should be obvious and easily usable.

The networked part of the project will likely prove to be the most difficult, especially with a focus on large scale performance. The managing of thousands of users into thousands of different study sessions will also likely be challenging. Each session must handle a number of different functions, and the inclusion of multiplayer game style activities adds another need for performance.

Some of the required skills needed are:

- Networked programming
- Database programming
- C++ programming
- UI Design
- Version control

5. Scope and Depth

The application will at a minimum include a client and server, and provide a method of networked group study. The client will manage and track a user's cards and provide them with material to study each day, which will be graded and effect future studies.

Specifically, the final project will provide the following:

- Allow the upload and download of cards/decks of cards
- Provide solo study
- Provide networked study
- Provide networked game
- Track and display statistics
- Allow users to chat with one another
- Language specific modules
- Registration, friends list
- Sync cards between devices

6. Test Plan

Testing for FlashyCard will be done in three parts. The first part is unit testing using MinUnit, a testing library for C/C++. Unit testing will be used for many of the card functions such as input acceptance, interval updates, etc.

The second part of testing will be stress testing. In order to determine how many users the server can handle, a client emulator will be written. This will simulate the actions of a real user – joining a room and participating in the study session. The goal will be to simulate a large number of users and observe things like latency to determine how well the server can handle a large user base.

The second part of the testing plan is acceptance testing, which will be completed manually. An example of some of these tests can be seen below.

Test #	**Description**
1	Deck created, all cards imported 1. Select input file 2. Run import
2	Deck created, all cards downloaded 1. Select deck to download 2. Run download
3	Card scored 100, removed from daily study 1. Enter correct input 2. Observe score
4	Card scored 50, not removed from daily study 1. Enter half correct input 2. Observe score
5	List of current study sessions displayed 1. Connect to networked study 2. Request room list
6	Join networked study session 1. Connect to networked study 2. Request room list 3. Select room, join
7	Networked game functions 1. Connect to networked study room 2. Participate in networked game
8	Auto setup runs, creates decks/imports cards for preloaded decks 1. Launch application 2. Run auto setup
9	Language specific scoring module works 1. Select Korean deck 2. Input correct answer 3. Observe score
10	Language specific grammar module works 1. Select Korean sentence deck 2. Observe grammar hints

7. Methodology

FlashyCard will be built using the Scrum methodology. The main appeal of this methodology is the flexibility of task that it provides. There will be a fairly high level of dependency between the client and server, and so some flexibility may be needed to ensure tasks can be moved up in priority should they become required to continue other work.

Scrum also offers the idea of sprints, which are something that should help keep the project on track. I am unfamiliar with long deadlines and large projects, so there is the worry that work will be continually put off, until there is too much to do at the end. By having a new plan and clear goals every sprint, it should keep the project moving and provide constant checkpoints.

Obviously one of the focuses of scrum, the standup meetings, will not be used for this project given that there I will be working alone. Still, the remaining parts of the methodology are enough that I believe it will provide the best chance of success.

Technologies Used

Design

- Lucid Chart: online diagraming tool used to create all design documentation for the project.

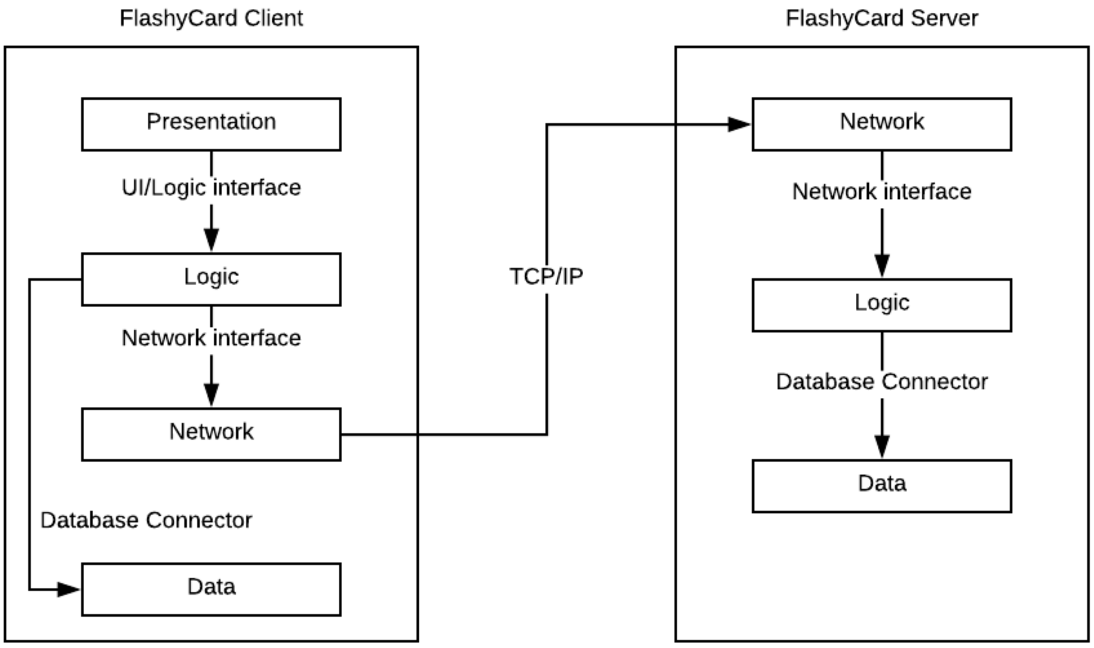
Development

- Qt Creator: main development tool for both client and server. Provides UI designed and a number of Qt libraries for C++.
- Git: used for version control

Management

- Trello: provides a scrum board like interface to track each task.

8. System/Software Architecture Diagram



Client

- Presentation: The presentation layer is in charge of displaying information generated by the logic layer. This includes displays during studying, networked game screens, etc.
- Logic: The logic behind the application. Manages networked and solo study sessions, as well as deck management, networked management, etc. The logic layer draws on data in the data layer, and sends data to the networked layer.
- Network: Connects the client to the server. Sends updates like scores and chat messages to the server, receives study session and game updates from server.
- Data: Local database that holds a user's cards, settings, friends, etc.

Server

- Network: Connects to the client. Receives updates from the client and sends out updates for study sessions, games, chat, etc.
- Logic: Manages all the study sessions currently taking place. Makes requests of the data layer and sends data to the client through the network layer.
- Data: Holds registered users, downloadable decks, etc.

9. Innovation

FlashyCard will feature two main innovative components.

The first of these components is the networked aspect of the application. This is not something offered by any of the current flashcard applications, and it's a big part of FlashyCard. The basic idea is that each user will have a number of cards due each day. The application will search all the current study rooms for those with significant overlap in due cards. The user can then join one of these rooms and study with anyone else connected. Each user will be shown the same card, as dictated by the server, and given a set period of time to answer. The timer, and a points system to reward the fastest correct answers, will provide motivation for those studying. The networked study option will also provide quick, multiplayer, review exercises for all participants to complete. In addition, the ability to chat with other users will provide the opportunity to practice communication in the studied language.

The benefits and importance of group work for language learning has been studied extensively in the classroom setting. The findings of these studies suggest that "in addition to strong pedagogical arguments, there now exists a psycholinguistic rationale for group work in second language learning." (Long & Porter, 1985). Obviously differences between classroom and self-directed flashcard learning exist, but there remain two arguments for a networked, group study: motivation and increased practice opportunities.

The first benefit of group work that FlashyCard hopes to capture is motivation. The repetitive nature of answering 150 or more flashcards in one sitting, one after the other, can make studying with flashcards a boring and demotivating experience. Through group study, FlashyCard will address the challenge of motivation with features such as a competitive points system, multiplayer review games, card timer to keep everyone on track, and by breaking the large daily set of cards down into several smaller group sessions. While some of these could be implemented in a solo study manner, the directness of the group method seems preferable. For example, a solo method could update the user on how they are progressing against all other users with the same deck of cards. However, comparing this to actively competing against others in your group as you study seems to be more direct and rewarding.

A second benefit of studying as a group is the networking between people it encourages. Just as in a classroom setting, the group study of FlashyCard will provide users with increased practice opportunities. These practice opportunities are especially important for someone using a flashcard application to study, as they may have little or no opportunity to practice communication with others in their daily life. The ability to talk with others in the group while studying is therefore an important feature of FlashyCard as a language learning tool. In addition to communication as a way to practice, it also provides users with a way to share general study tips and tricks, as well as more specific ways of how they thought about a particular card or grammar concept in order to remember it. This sharing also extends to material, with users able to suggest useful decks or resources to each other.

The second innovative aspect of FlashyCard is the scoring systems, in particular the support for sentences. As mentioned previously, the current applications are built more for single word, vocabulary learning, and require a completely perfect answer to be considered correct. This is not an ideal solution for sentences, and so FlashyCard will provide some generalized hint and scoring system for sentences, as well as the ability to add modules for specific languages.

The idea behind these modules is to separate grammar from vocabulary to allow for staged learning. The first stage is a review of vocabulary, with hints provided for any grammar components. The second stage focuses on grammar and provides all the basic vocabulary of the sentence for the user. The final stage requires the user to complete the entire sentence on their own. The concept is less significant in English, but as an example: ‘Colin**’s** proposal is faili**ng**’. The bold sections signify the potential grammar and conjugation parts of the sentence. The effect is more significant when seen in other languages: ‘차**가** 있**었으면** 좋**겠어요**’. Again, the bold section denotes the grammar and conjugation parts of the sentence, and in this extreme case makes up nearly the whole sentence.

The rest of the project will focus mostly on taking good aspect of different applications and combining them into one. Things like advanced deck management, significant statistic collection and more precise performance based card tracking are somewhat innovative within flashcard applications, but are not major parts of the project. One other significant aspect is that the application will be made for the desktop, unlike most of the current applications which are built for mobile. With a focus on sentences and primarily typed input, a proper keyboard is preferable. The networked study method also encourages a sit down, more lengthy study period that may favour a desktop application.

10. Technical Challenges

- Session Management: In order to be scalable and serve a significant number of users, the application must host and manage hundreds or thousands of different study sessions. Each of these sessions must manage the different connected users, the current session state, different event timeouts and game state and input. I have experience with a single lobby and subsequent game instance from Data communications, but FlashyCard will require a totally different approach to handle the required number of distinct sessions.

- Scalability: In order for the networked study idea of FlashyCard to be successful, it requires a significant number of users. The idea of the application is to match up users with similar cards due on a given day. The issue is that a user may be pulling a daily 100 cards from a pool of 4000. This creates vast differences between users despite having the same base deck of cards, and requires a large user base in order to find significant overlap. A second reason for scalability is general deployment. If the application were to be deployed, and were to become successful, it would be required to handle a large user base.

  A second technical challenge in terms of scalability will be developing a way to test the application and prove that it works as intended. This will require developing a way to simulate user traffic on a large scale, as well as capturing and analyzing traffic information. This testing will seek to stress test the application to determine how many concurrent users it can support while still remaining responsive, and will likely be quite challenging.

- Usability: The usability of FlashyCard will be another challenge. Being a language learning application, the core goal of the application is to provide a superior learning method. As mentioned previously, FlashyCard will implement several innovative language learning features, with the hope that they will help the user learn more effectively. Therefore, in order for the development of FlashyCard to be deemed successful, it must be compared to competing applications in a usability study. The study will focus on the users experience with the learning system of FlashyCard, and compare how effective it was against several similar applications. While I gained some experience in school with designing a similar study, doing it in the real world, with more realistic parameters, and carrying out the implementation and analysis phases will be a new experience and could prove challenging.

- Security: The application has two features which will require a level of security. First, the user information held on the server from registration, as well as their card information held for syncing between devices. The second critical component is communications between uses, both to their group and in private.

11. Development Schedule and Milestones

Task	Duration	Total Hours	Sprint Start	Sprint End
Proposal	10	10	1/24/2018	2/7/2018
Milestone - Sprint 1		10	2/7/2018	2/17/2018
Client Database design	2	12		
Design basic client functionality	2	14		
Client GUI mockup	2	16		
Client - Basic UI Complete	3	19		
Client - Basic database creation, connection	2	21		
Client - Card creation, importing, exporting	4	25		
Client - Deck management	6	31		
Client - Write all major card and deck queries	8	39		
Client - Basic VOCABULARY scoring algorithms, card intervals, etc	20	59		
Client - Study session basics - displaying cards, taking input, session intervals, etc	10	69		
Milestone - Basic Useable Client Complete		69		
TESTING - Single user study, card import	5	74		
Milestone - Sprint 2		74	2/17/2018	2/28/2018
Design server architecture, basic message structure	2	76		
Server Database design	1	77		
Server - Implement basic functionality	8	85		
Server - Message parsing, response formatting	4	89		
Server - Event loop, queue	6	95		
Server - Study session manager	15	110		
Server - Study session basics - send card, track responses, timeouts	15	125		
Milestone -Server basic functionality complete		125		
TESTING - Server testing	5	130		
Milestone - Sprint 3		130	2/28/2018	3/10/2018

Design - Registration, un-synced study, friends list	4	134		
Client - Message parsing, sending	6	140		
Client - Networked study basic functionality	12	152		
Client - Upload deck, download deck	4	156		
Server - Deck storage, listing	2	158		
Client - Send current session	2	160		
Server - Match current user session to existing rooms	4	164		
Server - User registration	4	168		
Server/Client - Friend list	6	174		
Client/Server - Networked, individual study (study with friend but with different cards)	8	182		
Milestone - Basic Networked Study Complete		182		
TESTING - Networked studying	8	190	3/10/2018	3/21/2018
Milestone - Sprint 4		190		
Design - Statistics tracking, hint system	2	192		
UI Mockup - statistics screens, add stats to study	1	193		
Client - Statistics Database, basic add/modify/get	3	196		
Client - Statistics tracking within study session	6	202		
Client - Create graphing functions, display	8	210		
Client - SENTENCE scoring algorithm	12	222		
Client - Generalized hint system	5	227		
Client - User defined hint system	5	232		
Client - Proof of concept language specific module	12	244		
Client - Preloaded decks, first time application setup to load these	8	252		
Milestone - Statistics, Language Modules Complete		252		
TESTING - Language modules, statistics	5	257	3/21/2018	3/31/2018
Milestone - Sprint 5		257		
Design - Networked game/exercise, settings	4	261		
UI mockup - game screens	1	262		
Client - Implement game UI, basic functionality	12	274		
Server - Implement game functionality	10	284		
Server - Study room settings	4	288		
Client - General settings	2	290		
Client - UI fixes, improvements	6	296		
Server - Add UI for displaying current statistics	4	300		
Milestone - Game Complete		300		
TESTING - Game	8	308		

Milestone - Sprint 6		308	3/31/2018	4/12/2018
Design - Client network traffic simulator	6	314		
Client - Implement traffic simulator for scalability testing	24	338		
Server - Optimization	15	353		
Client - tutorial	10	363		
Milestone - Development Complete		363		
TESTING - Final Testing	10	373		
Milestone - Sprint 7		373	4/12/2018	4/20/2018
User Guide	10	383		
Final Report	30	413		
Milestone - Practicum Complete		413		

The schedule is split into 7 sprints, with distinct goals for each sprint. Each sprint varies slightly in length, between 9 and 12 days. The flexibility in length is to allow for variance in feature development time. The goal at the end of each sprint is to have some fully functional and testable features, and so some sprints required a longer schedule.

12. Deliverables

The project will have four deliverables:

- Client
- Server
- User guide
- Final Report

13. Conclusion and Expertise Development

This project will focus mainly on the networking side of my education the last two years. This learning proved to be the most interesting to me, and most applicable to future work. Specifically, this project will require further learning about client/server architecture, server scalability, etc. The project will also touch on the security side of the course, in the need to both keep users data secure server side, as well as secure all communications between users. Finally, through this project I hope to practice more general software development skills such as design, documentation, database, and testing.

14. References

Long, M., & Porter, P. (1985). Group work, interlanguage talk, and second language acquisition. TESOL Quarterly, 19, 207–228.

15. Change Log

- Jan 26, 2018 – Initial draft
- Feb 7, 2018 – Final revisions
- Feb 25, 2018 – Address group study benefits(Page 7, Section 9, Paragraph 3-5)
- Feb 25, 2018 – Added security challenges(Page 9, Section10, Paragraph 4)
- Feb 25, 2018 – Address networking focus(Page 11, Section 13)
- March 2, 2018 – Updated Technical Challenges – Usability, Scalability (Page 9, Section 10, Paragraph 2-3)