Generating Customized, Intelligent, and Dynamic Narratives

By

Yu Feng

A thesis submitted in partial fulfillment of the requirement of the University Honors Program at Southern Polytechnic State University and the Bachelor of Science degree in Computer Science

Southern Polytechnic State University
November 26, 2012

Approved by:

_____	_____
Faculty: Dr. Jon Preston	Department Chair: Hassan Pournaghshband
_____	_____
Honors Com. Rep:	Honors Director: Dr. Nancy Reichert

Acknowledgements

It is my honor to extend my gratitude to the people who have guided me in the process of writing my thesis paper, especially thanks to my advisor Dr. Preston who provided me with this great project; to Dr. Nancy Reichert who gave me valuable advice and suggestions on my paper; to my girlfriend Yuan Yuan and my best friends Zoltan, Shawn Day, and Austin Ford who have spent their time helping me check the research paper, and thanks to my parents who have shown great concern and support to my study.

Notice to Borrowers

In presenting this thesis paper as a partial fulfillment of the requirement of the University Honors Program of Southern Polytechnic State University, I agree that the college library shall make it available for inspection and circulation in accordance with regulations governing materials of this type. It is understood that any copying from this document must be done in accordance with proper citations and that any potential use of this thesis paper for financial gain will not be allowed without written permission of its author.

The author of this thesis is:

Yu Feng
1100 South Marietta Parkway SE
Marietta, Georgia 30060

The director of this thesis paper is:

Jon Preston
Computer Science
Southern Polytechnic State University
Marietta, Georgia 30060

Borrowers of this thesis paper not regularly enrolled as students at Southern Polytechnic State University are required to indicate acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis paper for the use of their patrons are required to see that the borrower records here the information requested.

Name of Borrower Address Date

Abstract

Computer games as a special way for entertaining have been growing rapidly within the last twenty-two years. Classic games such as Angry Bird and Call of Duty are quite popular all over the world. The quality of a game is one of the most important criteria to tell if a game is successful. Therefore, good game designers know how to break the traditional patterns in terms of game idea, game structure, and game story. They are paying more and more attention to how to offer their players a new thrilling experience, and how to enhance the interactions between players and games to increase players' sense of pleasure is what game designers are pursuing.

This paper is centered on research called "Generating Customized, Intelligent, and Dynamic Narratives" published by Jon A. Preston, Jeff Chastine, and Jeff. Greene [4]. Their research paper illustrates an idea of how to make games fun and interesting. The idea is "playing the game without having to actively engage with the application." Through this method, players can build a special relationship with the game they are playing and get a story that only belongs to them. This research paper inspires me to think about how I am able to help enhance the interactions between the real world and 2D games for players.

Generating Customized, Intelligent, and Dynamic Narratives

Yu Feng
Southern Polytechnic State University
November 2012

Table of Contents

1. Introduction

With the help of new technologies, we are now able to play very high quality video games that we could never imagine before. These new technologies help the games become more and more real. Some of the games even bring players' illusions and fantasies to life as if they were really in the games. Most of these kinds of games include fantastic gaming environments, realistic character design, and exciting music. Although players are much more easily attracted by these fabulous gaming components, we know clearly that these are not the only rules by which to judge a game's success. Other important elements still need to be considered such as whether the game is fun, whether the game is easy to play and so on [6]. That is why certain games, such as text-based ones, are very popular despite the lack of graphics. One example is the Zork series, which allows players to use a keyboard to enter commands such as "go north" or "open the door." Text-based games can be interesting and creative.

The game the research deals with, Parallel Journey was created by Jon A. Preston, Jeff Chastine, and Jeff Greene. [4] This is also a text-based game. The story is generated dynamically based on players' actions in the real world. Before introducing this game, I would like to start with defining interactive narrative. J.Porteous, M.Cavazza, and F.Charles [5] point out that "Interactive narrative is a form of digital entertainment in which users create or influence a dramatic storyline through actions, or acting as a general director of events in the narrative. Interactive storytelling is a medium where the narrative, and its evolution, can be influenced in real-time by a user."

Their theory discusses how to create a variety of actions in the gaming world that directly influence the game story itself. In order to explain details of this research project,

we first introducing design level for client application for the gaming world and for the system of generating narratives and then go more deeply into the implementation level of the game via the client-side, the server-side, the generating-of-narratives-side, and the auto-sending-of-narratives-by-email-side.

2. Chapter Two

2.1 Design of Client-Side Application

The purpose of designing a client-side application is to transmit players' information, such as locations and time, to the server-side as well as to provide a way for players to better interact with the gaming world and with other participants.

Therefore, in the current version of the client-side application, the design is very simple and the function is limited. It has only two separate views: one for login and the other for game-play. The login view only consists of several inputs, text areas, and a signal login button. When the player finishes inputting the correct information, it jumps to the game-play view. Figure 2-2 shows the game-play view in the current version. However, it holds too little information for players to handle. Therefore, a new version of the game view was created (Figure 2-3), and hopefully it can replace the current one in the future.

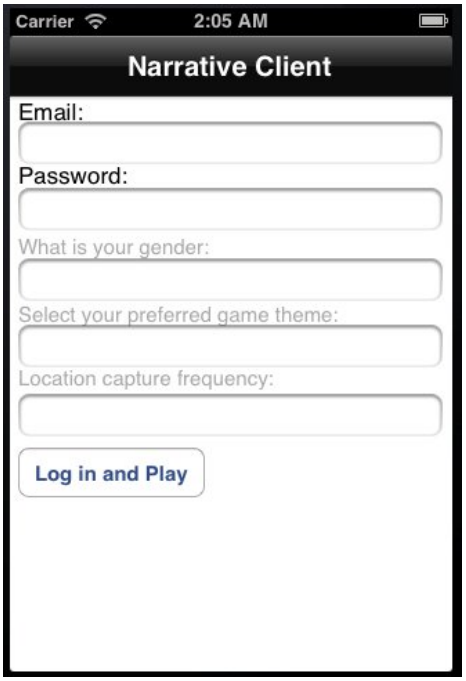

Figure 2-1: Login View

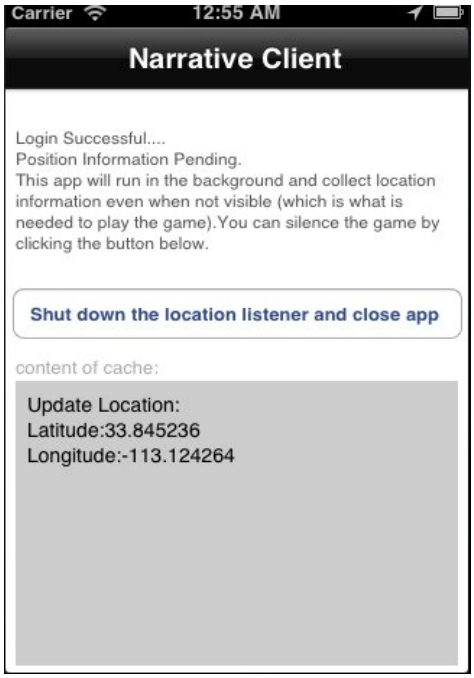

Figure 2-2: The current game-play view

The new version of client-side application will keep the current login view since it is simple and easy for players to use. The changes focus on the game-play view. The design idea came from the current mobile device, iphone. We all know that iphone's interface is a great design sample due to its simplicity and efficiency. Therefore, the design of the game-play view is simple, but it still contains enough information. According to the user interface design principles [10], the following principles are especially important:

- Know your user
- Pay attention to patterns
- Stay consistent
- Use visual hierarchy
- Provide feedback
- Be forgiving
- Empower your user
- Speak their language
- Keep moving forward

Based on these principles, I designed the following game-play view:

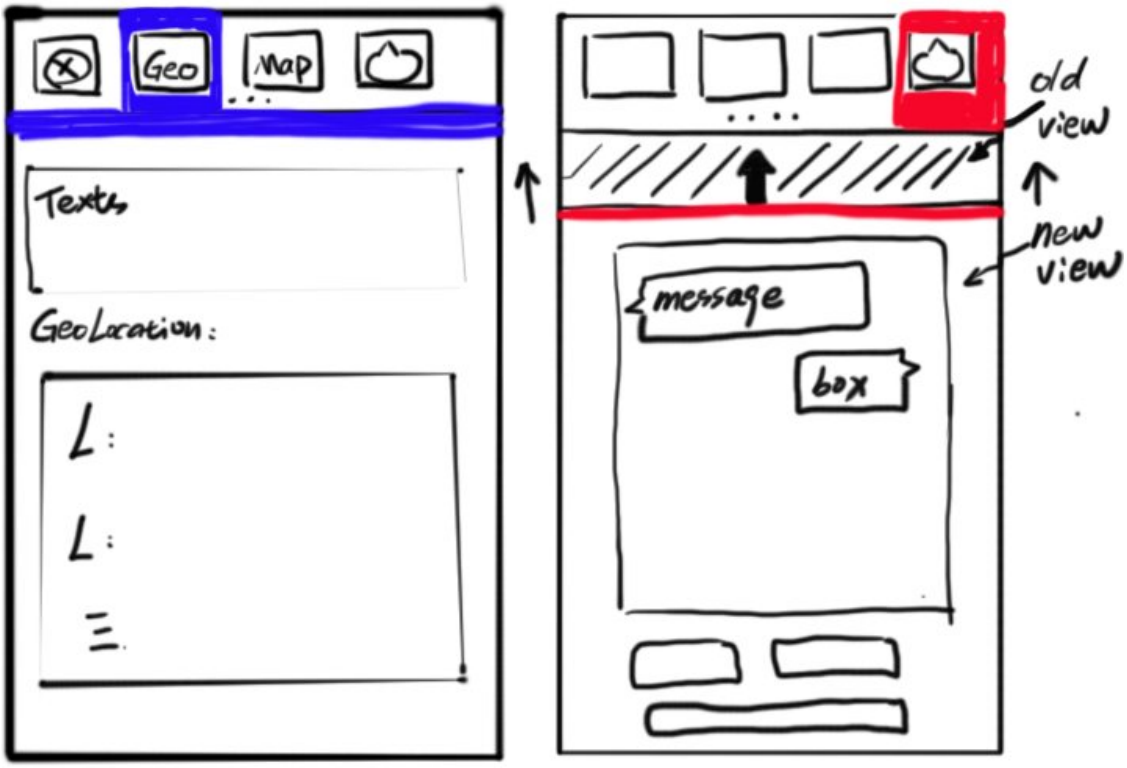

Figure 2-3: New game-play view draft

As we can see in Figure 2-3, icons are lined up in one row. Although there are only four icons visible for the players right now, they can slide the view to the left or to the right to display more icons. Each icon specifies only one function related to this game. For example, when the user presses the “Geo” button, the new view will slide from the bottom to the top (Figure 2-3) without covering up the buttons. User interface principles [10] were applied to this process. As mentioned earlier, some icons were created to look like those of popular interface devices such as the iphone and Android. This was done in order to maintain consistency since most of the users are familiar with the Android/IOS device. It is hard for the users to get familiar with new interfaces if they are already used to their current ones. What is more, the user clicks the button which brings up a new view

with the intended purpose to provide fast feedback. In addition, the function of each icon also needs to be defined. Some of the following buttons are used:

Figure 2-4: the new icons

The first one is a help button which is used to tell the new player how to use the application. The second button is used for changing language since this application may be used by different people. The third one is a map button that provides 2D/3D maps for the gaming world. The fourth one is a message button which allows two or more players to communicate and trade. Last but not least is a settings button. This button is used for setting reminders, adjusting sound volume, and other related functions.

2.2 Design of Gaming World

As the game developers mentioned in their research paper, their game Parallel Journey successfully implements Szilas' proposed architecture of defining descriptions and event chronology [12]; However, only sending locations and moving in spaces are not enough to provide fun for players.

Providing more interaction between players and the real world will be necessary in future designs. With this intention in mind, I combined some of my own ideas with the research of certain resources and books. According to Julian [11], Nicholas [3], Jeannie [7], and David and Rusel [9], the following activities would be beneficial to consider in the future version of Parallel Journey.

- **Players can collect virtual items in the real world --** When players go into specific area like rivers, forests, and libraries, they will get virtual items like water, wood and books, and then save these data into the server for future using.

- **Players can create a new item based on collected items --** After players have collected different items, they can use these items to create a new item. For instance, a sword is created by + 1 wood and + 1 book. After players create new items, they can equip themselves with their items in the virtual world whenever they want.

- **Players can increase their personal skills --** If we have the players' personal portfolios, we can let players choose what kind of skills they want to increase. For example, players can increase their knowledge, muscle, and running skill in the real world and then we can change those real skills to virtual numbers and store them into players' personal portfolios for future playing/fighting.

- **Player versus Player --** Who wins or who loses can be determined by the players' skills; their equipment, and their luck.

- **Player versus Professor/Enemy/Enemies --**
    - If this game includes a real world classroom with a professor, we can let the professor give the player one or more questions which may be related to classes. If the player answers correctly, he/she will win an item; if not, he/she will lose an item.

    - If the player challenges an enemy/enemies, we can set a specific real world area to hide one or more enemies. When players go into that area, the battle event will be triggered by the system. The winning condition also could be determined by the player/players' skills, their equipment, and their luck.
- **Players can do instant chatting and trading --** Players can chat with each other instantly and trade virtual items when they meet with each other in the real world.
- **Players can complete different missions --** Creating different missions to help players increase their skills or to give them new items. The missions could be hidden and not told to the players or the missions could tell them what will happen by creating a calendar like WoW.
- **Awarding system and Ranking system --**
    - We can create an awarding system by using points as money in the game world. For instance, the players completing one mission grant +10 points in knowledge skills; the players attending class grant +3 points in knowledge skills, and so on.
    - The ranking of each player can be determined by points in the virtual world.

Adding these new game elements will make this game more fun and meaningful. We need to keep in mind that the purpose of the game is to facilitate fun, so as long as the new game elements are fun for players, we should consider adding them in the future version of Parallel Journey. However, if we do not have an interactive interface, that

would allow players to use their mobile devices to implement actions in the virtual world, creating these new game elements would become useless. Using the current version of the interface is far from enough, which is why a new mobile interface which allows players to interact more in the virtual world was designed. For example, An instant chatting and trading system in the future is necessary, so a message icon (Figure 2-4) was added to the client interface. When other players are nearby, the interface will provide a window to remind players to talk to or to trade items with other players. If all these interactions are simple, short and engaging [2], then they are worth adding into Parallel Journey.

Besides adding new interaction elements, the game developers also mentioned that using events generated in "... the system to result in instructions that modify a 3D graphical world," [8] would change the text-based narrative to a graphical representation of the events in the virtual world. In addition, they refer to the work of Douglas and Gratch [1], which "established a formal, functional notation/grammar for defining time relations between events as a means of creating dramatic tension." It would be possible to inject this formalism into a narrative generation algorithm. In short, using this method could help designers control time so as to generate deep and meaningful narratives.

All of these different designs for the client application and the gaming world that we have presented are mainly for players to engage more in the game rather than to just send their location and move from one space to another. As long as these new game elements are helpful for players, we would consider adding them in the future.

2.3 Design of Narrative Generator System

The Parallel Journey is designed by Jon A. Preston, Jeff Chastine, and Jeff Greene [4]. As they point out the two main ideas of Parallel Journey are first "a game that utilizes the GPS-tracking technology inherent in mobile phones to tell an engaging story." and the second "a player could 'play' the game without having to actively engage with the application."

In order to generate engaging stories based on their own ideas, they can use GPS with an engine that could generate and parse narrative strings based upon player location across a given space and time. For example, after the system keeps tracking a player during a game day, his/her locations are sent to an engine which includes different states to specify each location and a variety of real world events such as "player X enters a building," or "player X passes player Y." After the engine gets the location of that player, it will dynamically generate a persistent, emergent narrative based on real-world movements in real-time.

In the following chapters, we present more details of the system architecture, implementation, and the application for auto sending narratives by email. Section 3 presents the process of developing the client and the server elements of the system. Section 4, 5, 6 concern the implementation of client, server, and generation of the narrative. Section 7 is the implementation of auto-sending narrative by email.

3. Chapter Three

3.1 Process

Before we start the game, we need to get the players' signature and the name of the virtual avatar. This virtual avatar will also appear in the narrative.

After the players complete the paper work for participation in the Parallel Journey research, then they are ready to begin. First, the players need to install the application for the client-side. Then the players need to input their information regardless of whether Android or IOS device. After the players click the “Log in and play” button, the application will send the information to the server-side to verify the players' information. If the players have already registered for the game, the server will give them participants IDs which are unknown even to the participants; otherwise, the application will give them an error. This system not only identifies events as they occur, but also delivers the narratives for each player at the end of each day.

When the players successfully login to the game, the client device will start to send their GPS locations (latitudes and longitudes) to the server. The server will keep receiving each of the player's locations and storing them for the sub-system to generate the narratives when the game day is finished. (See figure 3-1. On page 19)

The software of the sub-system had already identified several real world “events” that would require narrative strings such as:

- entering the game world (while infected)
- entering the game world (while uninfected)
- entering a building
- leaving a building

- passing by an infected player (when not infected)
- passing by an uninfected player (when infected)
- passing by an infected player (when also infected)
- passing by an uninfected player (when uninfected)

These event states help to produce narratives strings that can be combined and thus form a cohesive whole. We do not need to consider the genre, theme, and individual descriptive details. All of these can be altered by the system. All the players need to care about is picking what kind of theme they want at the beginning. Different kinds of themes will generate different kinds of styles; the same theme could also generate a different narrative. For example, I picked the cyberpunk theme twice but the results are different:

First time:

I ran ancient protocols for hours, searching through vast hordes of broken software. The vendors in the hub offered their software and protocols at half price, citing Quantum Zero sales as hurting their business. One dealer in broken code suggested that a free 'runner had discovered a cure for Quantum Zero and may have left it somewhere in Essentia. It took hours to hunt through the data streams in the ancillary core, but I found nothing but garbled data and worthless junk....

Second time:

The vendors in the hub offered their software and protocols at half price, citing Quantum Zero sales as hurting their business. One dealer in broken code suggested that a free 'runner had discovered a cure for Quantum Zero and may have left it somewhere in Essentia..

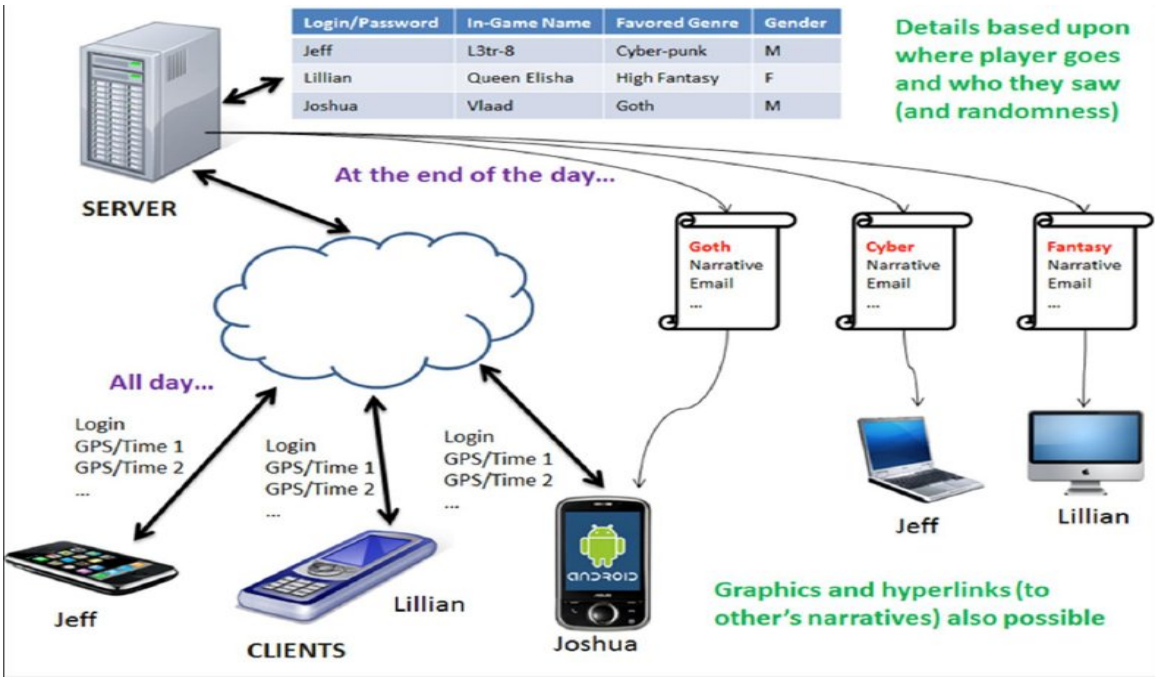

Figure 3-1: Overview of system

4. Chapter Four

4.1 The Client-Side application for IOS

When the players wish to start the game, they simply login to the application using their phones. Although in this project we only designed client application for Android and IOS devices, other devices can run the client as long as they include GPS/location services. After players provide their email, password, gender, and select the desired refresh/update option, they are ready to start the game (See figures 2-1 and 2-2 on page 9). The refresh/update option allows the user to choose how often their location is sent to the server. This function is very useful because the battery cost of GPS needs to be considered. The more frequently the location updates, the faster the battery is going to be consumed. Therefore, the client-side application allows the user to select their update

frequency as once per second, once per minute, once every five minutes, once every twenty minutes, and once an hour.

What is more, the user is able to select their desired theme which includes horror, high-fantasy, and cyberpunk. However, these themes are not dependent upon users' actions in the real world. The narratives are based upon user actions that are independent of theme.

Once players finish filling in the login information and clicking the log in button, the client starts to connect with the server-side via the *GCDAsyncSocket* application which is a TCP/IP socket networking library used for support IOS devices. This application also supports full delegate, queued non-blocking reads and writes with optional timeouts, automatic socket acceptance, TCP streams over IPv4 and Ipv6, and TLS/SSL. Therefore, it is very powerful and useful. We simply provide the IP address and the port number of the server to API, and then call the *connectToHost* function. After that, we wait for the application to give us the response. If it successfully connects, the client begins to sending the user information with the following format by calling *writeData* function:

LOGIN:email:password:theme:gender:ParallelJourney

Once the client information is matched with the server side, the server will send back a UID for the user. Then the client is allowed to update the positions of the user at regular intervals by using the following format:

ID:time:longitude:latitude:ParallelJourney

Currently, the interface of the client only displays the current GPS location and specifies how many updates have been transmitted to the server. I hoped in the future, more

functions can be added into the client interface in order to provide a better way for players to interact with the game. Then the users can put their phone in pocket and 'ignore' their application thereafter or run the application in the background and meanwhile use other applications.

4.2 The Server-Side

The server is a place to hold all of the players' GPS locations and timestamps so that all of this information can be pushed into a sub-system which is used for generating narratives. The server not only stores this information, it also helps to accept a reduced set of data from the client and can parse that information efficiently.

The server begins by loading a list of players' information which contains players' email addresses, passwords and unique IDs. Here is the format for players' information:

Email address	Avatar name	UID
yfeng2@spsu.edu :	*gamemaster* :	*42*

Then the server sets up *TcpListener* which is a class that listens on the specified port. Once a client connects, the server verifies that the client is legitimate and returns the UID of the player. The following pseudo-codes implement the function of legitimizing and returning UID for players and the function of sending players' locations and timestamps:

```
try
 {
    //read the file from the client-side and save them into a string called data

   //if the data variable include "ParallelJourney" and "LOGIN", it means this data is for client to login.
    if (data.Contains("ParallelJourney"))
    {
       if (data.Contains("LOGIN"))
```

```
            {
                //scan the file to find the matched information and send back ID to that client

               // If the file does not include that client information, then send -1 back to  that
client
               if (ID not includes the client infor)
               {
                  ID = "-1";
               }
            }
            else
            {
               //if the client sending the data that does not include "LOGIN", that means the data
are the locations and time for that client. In that case, all we need to do is storing the locations
and time into a new file.
            }
         }
      } ...
```

Figure 4-1: pseudo-codes

As the players move around in the real world, the client will keep sending a single string of information containing the UID, the coordinates of the player, and the timestamps of the event to the server. Both login data and GPS updates for all players are appended to a common file in the order they are received (See figure 4-1). Here is an example of the server data that was saved in a file:

(UID:Date:Latitude:Longitude:Distance filter: ParallelJourney)

42:login:cyber:male

42:1346792119959:33.7068594:84.89133933333333:10.0:ParallelJourney

...

In order to handle multiple users sending the requests and data concurrently, the server uses a threaded architecture. However, requests for data appear frequently throughout the day and can prevent legitimate requests from being satisfied. To prevent this problem, a thread is spawned for each connection – preventing intentional (or unintentional) hanging from malformed data. Dead or hanging threads are cleaned up periodically by the server.

It should be noted that all of the coordinates from the players are recorded into the server-side even if some of them are invalid or useless for generating the narratives. What is more, due to data passing and saving data, server security should also be addressed in the future.

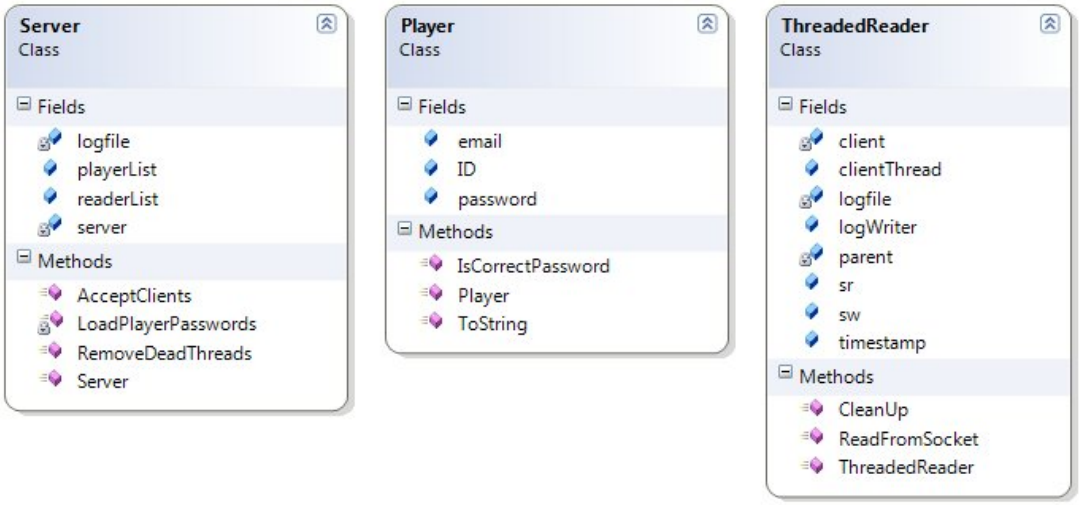

Figure 4-2: UML diagrams for server-side class

4.3 The Event Generator and Visualization Tool

The raw location with UID data captured by the players in the previous step is subsequently processed by an event generator. The primary responsibility of this part is to convert the raw GPS data into events that are written to a file and then to generate the file with narratives for each player.

In order to generate the narratives, only providing raw locations with UID data to the event generator system is not enough. The following files also need to be loaded into it:

1. **Narrative Elements file**: This file includes theme, building, variation, event and text output.

Parts of data in narrative elements file:

Theme	*Building*	*Variation*	*Event*	*Text Output*
Cyber	*J*	*1 (Intro)*	*Enter*	*I found an ancillary data...*
Fantasy	*J*	*1 (Intro)*	*Enter*	*The Ruins of Kazagard had...*
Horror	*J*	*1 (Intro)*	*Enter*	*The barn had looked...*

2. **Players file**: This file includes reloading player information which are UID, in game name, theme, and gender.

Parts of data in players file:
67:Vlaad:fantasy:male
41:The Nameless One:horror:male
98:Shambling Horror:horror:male
42:Mad Hatter:cyber:male
...

3. **Interaction file**: This file includes theme, variation, player infected, other player infected and description.

Parts of data in interaction file:

Theme	*Variant*	*I'm Infected*	*You're Infected*	*Description*
Cyber	*1*	*1*	*0*	*I saw <X> and...*
Fantasy	*1*	*1*	*0*	*<X> passed by...*
Horror	*1*	*1*	*0*	*<X> passed by...*

4. **Preloaded player information file**: This file includes players' email addresses, passwords, and UID numbers.

Parts of data in player information file:
yfeng2@spsu.edu:gamemaster:42
...

As mentioned previously, all of the narratives generated are decided by different states:

- **login**: which records the ID, theme and gender of the players.
- **entered**: the player has entered a building or other physical space, such as campus.
- **stayed**: a meta event that determines the length of time in a space.
- **exited**: the player has left a physical space.
- **near**: the player has encountered another player; this event contains the IDs of both players.

The reason why the system adds different states is because “during each step of the simulation, the system interpolates the location of each active player to determine if any of the events above have occurred. Interpolation is necessary because it is highly unlikely that the time step of the simulation matches the frequency at which the players sent their positions (i.e. the simulation runs at approximately 60 Hz and simulates at approximately 20 times faster than real time.)” In short, the states are used for allowing the system to easily recognize each player during the game.

In addition, the physical structures of campus were encoded as planar data relative to the footprint of the structure and were recorded into a text file. A point-inside-polygon algorithm was used to generate the entered and exited events, and near events was based upon direct 2D point distance. That's how the sub-system generated each of the events.

Although catching the locations for each player seems easy to implement, debugging is a critical problem when testing the system. Thus, having a visualization tool to see the location of players is necessary. Using this tool not only helps authors to easily debug during the testing of the system, it also provided authors insight in how players behaved in the real world (See figure 4-3).

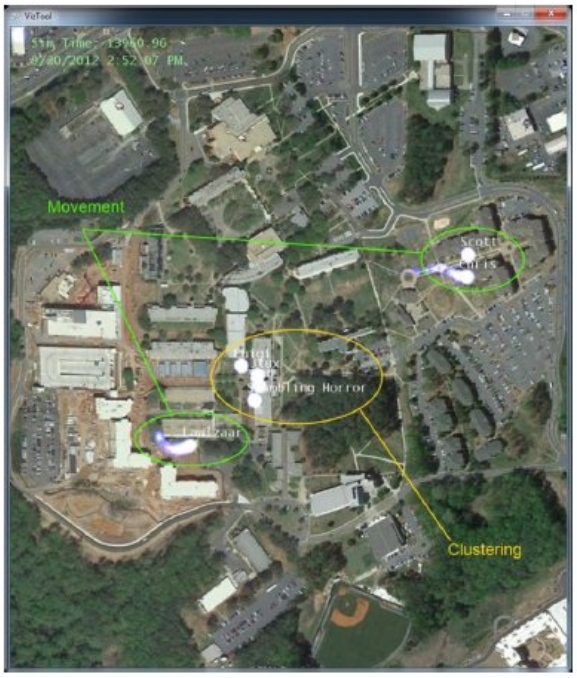

Figure 4-3: Visualization Tool

4.4 Generating the Narratives

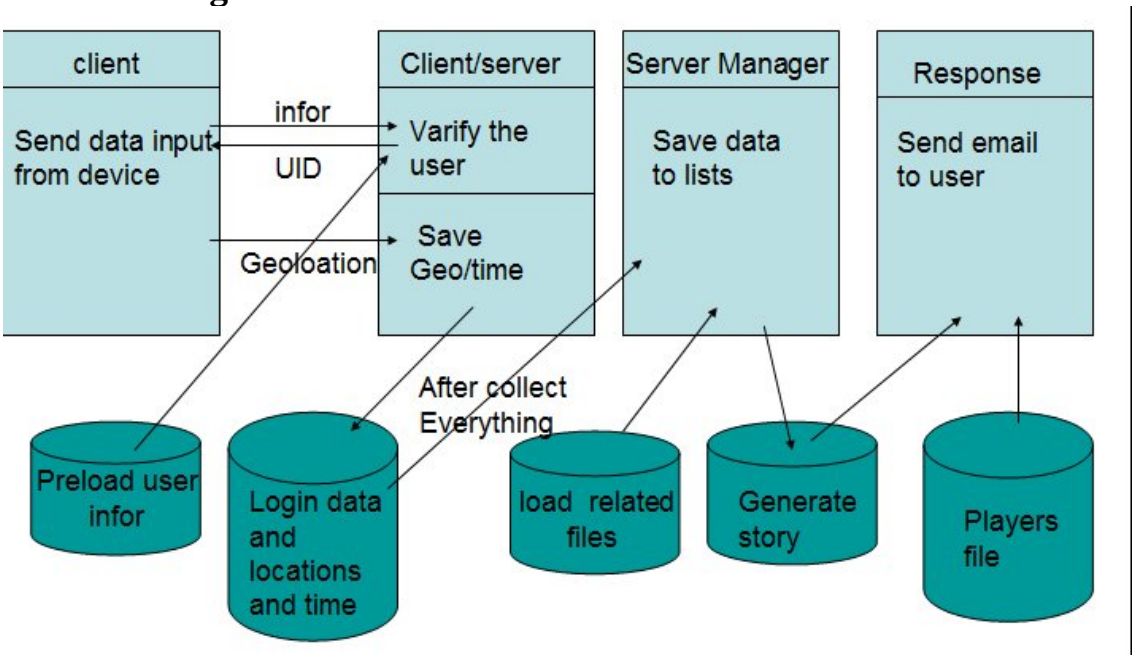

Figure 4-4: process of generating the narratives

As noted earlier, the events generated rely on different states like *enter*, *stay*, and *leave*. However, having only these states is not sufficient to generate dynamic narratives so that players can have their own story when the game day finish. Therefore, the system adds three themes and each theme has three variations. The different themes are used for generating different style of stories and additional three variations within each theme are used for generating different descriptions when the player returns to the same location. Apparently, more random descriptions per theme per location would be better, but having three themes and three variations for each theme is sufficient for testing because right now we have six places for the players to move around. Each place has its own description for *enter*, *stay*, and *leave* as well as three variations for each *enter*, *stay*, and *leave* state. Therefore, one player may have more than thirty possible descriptions in each place, which is enough for testing.

As you can see, different themes with the same location can have three different descriptions. For example, one player may want a Cyberpunk themed narrative, and upon entering location A would receive the “The energy soaked into ….” whereas another player who chooses the Horror theme, upon entering location A would receive “Inside the manager's office...” It is important to mention here that even though players choose different themes, the interaction between players will not be affected because the theme is independent of mechanics and actions within the game system.

When the players see each other in the real world, their information is sent to the system and will be treated as a “near” event to the event list of the day; as a result, the narrative will have a description of the player seen by the user. As you can see in finger 4-4, which is an example of interacting with another player, there are “<>” tags to fill in each of the descriptions. The tags are used for switching gender and pronouns when players interact with each other. What is more, the interaction events are also separated by themes. Additionally the system also offers multiple “near” elements per theme to allow randomness in the narratives.

NARRATIVE ELEMENT TEMPLATES

Examples of Staying in a Location

Cyberpunk	*Horror*	*High-Fantasy*
The energy soaked into my avatar and I lost myself in the experience.	Inside the manager's office of the concrete factory there was a working radio. I turned it on and listened to the crackling emergency broadcast. It urged people to stay in their homes and warned of the infectious bite of the undead.	I lingered amongst the stones, reflecting on the Shades and what had become of Stormworld.

Examples of Interacting with Another Player

Cyberpunk	*Horror*	*High-Fantasy*
I saw <X> and something about <POS-PRONOUN> avatar made me feel lightheaded and hungry. I followed <P-PRONOUN> for a bit, my vision spooling, before finally losing sight of <P-PRONOUN>.	<X> passed by me and I could smell <POS-PRONOUN> fear and terror. I followed <P-PRONOUN> as long as I could, but <PRONOUN> seemed to be moving faster and faster.	<X> passed by me and I could smell <POS-PRONOUN> fear and revulsion. It bolstered me to continue onwards.

Figure 4-5: table

In the code side of generating the story, there are four different classes which I mentioned before to handle each of the different functions. (See figure 4-5) The following pseudo-codes in figure 4-6 show how to generate a dynamic narrative:

```
public Game()
{
   //In the constructor method, it calls functions which uses  to loading files either
  generated in pervious step or generated before the game begins.

   ReadLocationDataFile("NarrativeElements.csv");
   BuildPlayers("players.txt");
   ReadInteractionsDataFile("interactions.csv");
   ReadEventDataFile("events.txt");
}

//In ReadLocationDatafile class, it simply loads the NarrativeElements.csv file and saves
into location list.
public void ReadLocationDataFile(string filename)
{
        while (if the file is not end)
        {
           //read the file
          //add into location list
           AddLocationData(theme, location, variant, eventType, description);
        }
        //close the file
 }

//In player class, it loads data from players.txt and saves into player list.
private void BuildPlayers(string filename)
{
        while (if the file is not end)
```

```
            {
                //read the file
               //add into location list
                Players.Add(p);
            }
          //close the file
     }

    //In ReadInteractionsDataFile class, it loads data from interactions.csv file and saves the
contents into list
    private void ReadInteractionsDataFile(string filename)
    {
            while (if the file is not end)
            {
               //read the file
              //judge whether get infect and then add into list
               if (tokens[2] == "1") actorInfected = 1; else actorInfected = 0;
               if (tokens[3] == "1") otherInfected = 1; else otherInfected = 0;

               AddInteractionData(theme, variant, description, actorInfected, otherInfected);
            }
             //close file
    }


    //In ReadEventDataFile class, it loads the events.txt file and saves the data into event list.
    private void ReadEventDataFile(string filename)
    {
            while (if the file is not end)
            {
               //read the file
              //add into event list
               Events.Add(ev);
            }
            //close the file
    }
```

Figure 4-6: pseudo-codes

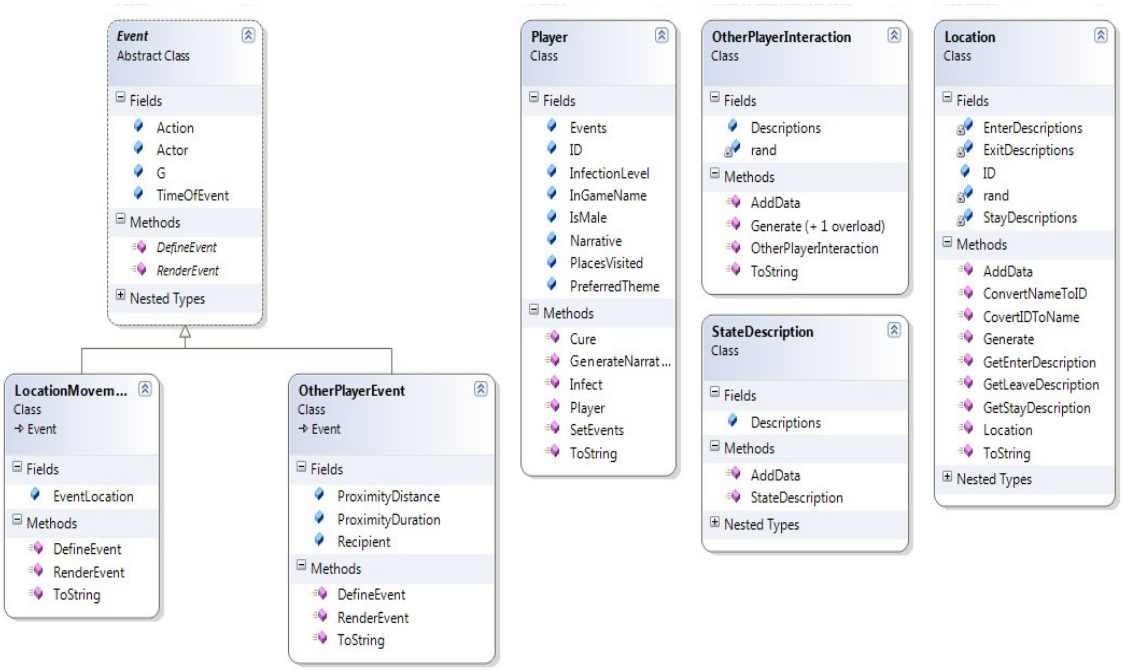

Figure 4-7: UML of generating narrative

4.5 Auto Sending Narratives by Email

After generating narratives for each player, there is an auto mail-sending application designed to handle auto sending emails to each player. This is a small application using *smtp* protocol to handle email sending. By initializing the *smtp* class and passing a current, usable IP address and port number, I then simply provide attachments and the contents of the email and send them asynchronously. (See figure 4-8)

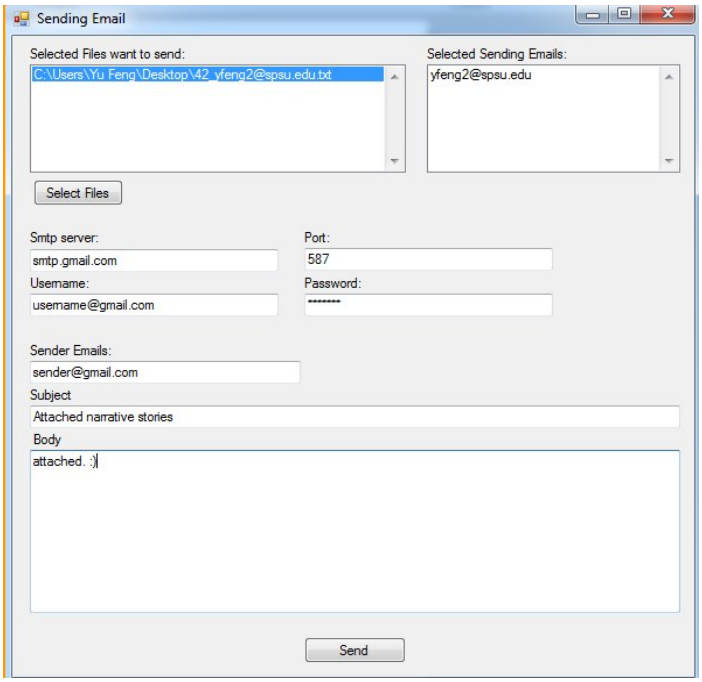

Figure 4-8: GUI for auto sending narrative story

5. Conclusion

Parallel Journey is a creative game. Unlike the traditional games that require players to stare at the computer screen and use a game pad or keyboard, players can do whatever they want in real life, for instance walking from one building to another. The essence of this game is to generate a dynamic narrative according to the places with the help of the server-side applications that the game designers created.

I feel really honored to participate in this project. It widens my mind and broadens my horizon. It was the first time that I knew games were not limited on a screen and a game pad. This game is so interesting because the design is so unique and the way of executing it is creative. The client-server architecture developed allows players to use their iphone/Android devices to connect and log their locations as they go about their day. Second, the narrative is generated dynamically and uniquely per player and customized to their experience throughout the day in the real world.

The more places the players visit, the more interactions the players have with the story the applications will help to create. In order to meet different players' needs, three game themes have been included.

When players end all the activities in a day, a story generated by the applications is sent to different players. Right now, this type of interactive games is only applied to Android and IOS platforms, and the interactions of the game by players are limited. Therefore, it is an inspiring beginning. I sincerely hope that in the future, this game could be expended to add variety of interactions between the players and the real world and bring more fun to the players. I also hope to continue to participate in the design and development in this game.

In the end, I would like to extend my gratitude to my professor who gave me this opportunity to participate during the design and development of this game. I feel really honored to be a member in this research group.

6. References

[1] Douglas, J. and Gratch, J., Adaptive Narrative: How Autonomous Agents, Hollywood, and Multiprocessing Operating Systems Can Live Happily Ever After, ICVS 2001, LNCS 2197, editors Balet O., Gobel S.,, and Torguet P., pp 100-109, 2001.

[2] Geiger, C., Paelke, V., and Reimann, C., Mobile Entertainment Computing, TIDSE 2004, LNCS 3104, editors Gobel S. et al, pp 142-147, 2004.

[3] Greene, Nicholas. (2012, Sep. 07). *Top 10 MMORPG Games for 2013*. Available: http://mmoattack.com/mmo-articles/top-10-mmorpg-games-2013.

[4] Jon A Preston, Jeff Chastine, Jeff Greene, "Generating Customized, Intelligent, and Dynamic Narratives," September 2012.

[5] J. Porteous, M. Cavazza and F. Charles (2010) Applying planning to interactive storytelling: Narrative control using state constraints.

[6] Lent, Michael van, "Interactive Narrative," University of Southern California.

[7] Novak, Jeannie, *Game Develop Essentials*, New York: Clifton Park, 2008.

[8] Park, J. and Lim, C., Mixed Reality Based Interactive 3D Story Composition Tool, TIDSE 2006, LNCS 4326, editors Gobel S. and Iurgel I., pp 181-186.

[9] Perry, David, Rusel DeMaria, *David Perry on Game Design*, MA: Boston, 2009.

[10] Preece, Jennifer, *Interaction Design beyond human-computer interaction*, NY: New York, 2002.

[11] Schoffel, Julian. (2012, May). *Why is WoW so popular?* Available: http://www.growlingdoggames.com/2012/05/why-is-wow-so-popular/.

[12] Szilas, N., Interactive Drama on Computer: Beyond Linear Narrative, Narrative Intelligence: AAAI Fall Symposium, editors Mateas M. and Sengers P., Menlo Park, California, AAAI, 1999.

Appendix A: A Sample of Generated Narrative

The Journey of Lawlzaar (Cyberpunk theme)

I spent a long time hunting through random files and data debris, coming up with nothing useful. It took hours to hunt through the data streams in the ancillary core, but I found nothing but garbled data and worthless junk. The time spent in the ancillary core revealed nothing important. No news about the Quantum Zero infection or background information on Aegis001. I found an ancillary data core and it looked to be fairly untouched. Datastreams were still glowing like a hundred incandescent spiderwebs. I pushed my way through one of the side access ports. At the back of the ancillary core there were a number of open terminals. I left through one of them. I found an ancillary data core and it looked to be fairly untouched. Datastreams were still glowing like a hundred incandescent spiderwebs. I pushed my way through one of the side access ports. I departed through an empty node. I found an ancillary data core and it looked to be fairly untouched. Datastreams were still glowing like a hundred incandescent spiderwebs. I pushed my way through one of the side access ports. Rune stopped me and we discussed the Quantum Zero infection. He didn't believe that it was as dangerous as all the 'runners seemed to think. Shambling Horror flew past me, leaving arcs and trails of light. Scott flagged me down and we looked at each other's avatars. He had decided upon leather wings, hooves, and a spined tail. Mine wasn't nearly as imaginative. At the back of the ancillary core there were a number of open terminals. I left through one of them. Luigi flagged me down and we looked at each other's avatars. he had decided upon leather wings, hooves, and a spined tail. Mine wasn't nearly as imaginative. Jtux flew past me, leaving arcs and trails of light.

Appendix B: Code of the system

Client-Side Code: (IOS)

View Two (Game-Play View):
Interface Class:

```
#import <UIKit/UIKit.h>
#import "CoreLocation/CoreLocation.h"
#import "AppDelegate.h"
#import "ViewControllerNEW.h"

@class GCDAsyncSocket;
@interface ViewController3 : UIViewController <CLLocationManagerDelegate>
{
   NSString *_serverIP;
   NSString *_longitude_data_reveive;
   NSString *_latitude_data_receive;
   NSString* _tempOfUserID;
   ViewControllerNEW *_viewcontrollerObject;
   NSTimer *_timer;

   int serverPort;
   int filterCount;
   float userID;
   CLLocationManager *locationManager;
   GCDAsyncSocket *asyncSocket_view2;
   AppDelegate *app2;
}
- (IBAction)ShutDown:(id)sender;
-(void)getServerIP;
-(void)logInCheck;
-(void)startRead;
-(void)startRead;
-(void)locationUpdate:(NSTimer *)timer;

@property (weak, nonatomic) IBOutlet UITextView *ShowLocation;
@property (weak, nonatomic) IBOutlet UILabel *outPutLabel;
@property (nonatomic,assign) float userID;
@property (nonatomic,assign) int serverPort;
@property (nonatomic,assign) int filterCount;
@property (strong, nonatomic) CLLocationManager *locationManager;

@property (nonatomic,retain) NSString *longitude_data_reveive;
@property (nonatomic,retain) NSString *latitude_data_receive;
@property (nonatomic,retain) NSString *tempOfUserID;
@property (nonatomic,retain) NSTimer *timer;
@property (nonatomic,retain) NSString *serverIP;
@property (nonatomic,retain) ViewControllerNEW *viewcontrollerObject;
@end
```

Implementation Class:

```
#import "ViewController3.h"
#import "GCDAsyncSocket.h"

@interface ViewController3 ()

@end

@implementation ViewController3

@synthesize outPutLabel,locationManager,filterCount,serverPort,userID,ShowLocation;
@synthesize latitude_data_receive = _latitude_data_receive;
@synthesize longitude_data_reveive = _longitude_data_reveive;
@synthesize serverIP = _serverIP;
@synthesize timer = _timer;
@synthesize viewcontrollerObject = _viewcontrollerObject;
@synthesize tempOfUserID = _tempOfUserID;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
        // Do any additional setup after loading the view.

    outPutLabel.text = @"Login Successful....\nPosition Information Pending.\nThis app will run in the background and collect location information even when not visible (which is what is needed to play the game).You can silence the game by clicking the button below.";
    outPutLabel.numberOfLines = 0;
    outPutLabel.lineBreakMode = UILineBreakModeWordWrap;
    [outPutLabel setFont:[UIFont systemFontOfSize:12]];
    [self.view addSubview:outPutLabel];

    self.locationManager = [[CLLocationManager alloc]init];
    locationManager.delegate = self;
    app2 = (AppDelegate *)[[UIApplication sharedApplication]delegate];

    //accuracy
    locationManager.distanceFilter = 10.0f;
    locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters;
    _viewcontrollerObject = [[ViewControllerNEW alloc]init];

    //start update
```

```
    [locationManager startUpdatingLocation];

    _timer = [NSTimer scheduledTimerWithTimeInterval:[self.viewcontrollerObject
timeAccuracy] target:self selector:@selector(locationUpdate:) userInfo:nil repeats:YES];
    userID = [[self.viewcontrollerObject response]floatValue];
    _tempOfUserID = [[NSString alloc]initWithFormat:@"%.0f",userID];

}

- (void)viewDidUnload
{
    [self setOutPutLabel:nil];
    _longitude_data_reveive = nil;
    _latitude_data_receive = nil;
    _serverIP = nil;
    _viewcontrollerObject = nil;
    _tempOfUserID = nil;
    asyncSocket_view2 = nil;
    [self setShowLocation:nil];
    [super viewDidUnload];
    // Release any retained subviews of the main view.
}

-(void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation
*)newLocation fromLocation:(CLLocation *)oldLocation
{
  _latitude_data_receive = [[NSString alloc]initWithFormat:@"%f",
newLocation.coordinate.latitude];
  _longitude_data_reveive = [[NSString alloc]initWithFormat:@"%f",
newLocation.coordinate.longitude];

  ShowLocation.text = [[NSString alloc] initWithFormat:@"Update
Location:\nLatitude:%f\nLongitude:%f",newLocation.coordinate.latitude,newLocation.coordinate
.longitude];
}

//update location
-(void)locationUpdate:(NSTimer *)timer
{
    //everytime it update the data will save to the server
    //already connect to the server
    if((_latitude_data_receive != NULL) && (_longitude_data_reveive != NULL))
    {
        [self getServerIP];
        [self logInCheck];

        double date = [[NSDate date]timeIntervalSince1970];
        date = date * 1000;
        NSString * stringWithDateFormat = [NSString stringWithFormat:@"%.0f",date];
        NSString * stringWithDistanceFileterFormat = [NSString
stringWithFormat:@"%.1f",locationManager.desiredAccuracy];
```

```
        NSString* locationData = [[NSString
alloc]initWithFormat:@"%@:%@:%@:%@:%@:ParallelJourney\n",_tempOfUserID,stringWith
DateFormat,_latitude_data_receive,_longitude_data_reveive,stringWithDistanceFileterFormat];

        //write data to server
        NSData *requestData = [locationData dataUsingEncoding:NSUTF8StringEncoding];
        [asyncSocket_view2 writeData:requestData withTimeout:-1 tag:0];
        [asyncSocket_view2 disconnectAfterWriting];
    }
}

-(void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error
{
    [_timer invalidate];

    NSString *msg = @"Error Obtaining Location.";
    UIAlertView *alert;
    alert= [[UIAlertView alloc]
            initWithTitle:@"Error"
            message:msg
            delegate:self
            cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
    [alert show];

    //go back to login view
    [self performSegueWithIdentifier:@"goBackToMainMenu" sender:self];
}

-(void)getServerIP{
    @try {
        NSString* urlString = @"http://cse.spsu.edu/jchastin/research/ParallelJourney/serverIP.txt";
        NSURL *url = [NSURL URLWithString:urlString];
        NSData *dataOfUrl = [NSData dataWithContentsOfURL:url];

        NSString *dataConverToString = [[NSString alloc]initWithData:dataOfUrl
encoding:NSUTF8StringEncoding];

        //pass the infor to ip and port variables
        NSArray *splitArray = [dataConverToString componentsSeparatedByString:@":"];
        _serverIP = [splitArray objectAtIndex:0];
        serverPort = [[splitArray objectAtIndex:1]intValue];
    }
    @catch (NSException *exception) {
        [_timer invalidate];

        UIAlertView *alert = [[UIAlertView alloc]initWithTitle:@"Error" message:@"Connection
Error." delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
```

```
        //go back to login view
        [self performSegueWithIdentifier:@"goBackToMainMenu" sender:self];

    }
}

-(void)logInCheck{
    //connect
    asyncSocket_view2 = [[GCDAsyncSocket alloc] initWithDelegate:self
delegateQueue:dispatch_get_main_queue()];
    NSError *error = nil;
    uint16_t port = serverPort;

    if (![asyncSocket_view2 connectToHost:_serverIP onPort:port error:&error])
    {

        NSLog(@"error connection");
    }
    else
    {
        NSLog(@"connect...");

    }
}

-(void)startRead
{
        [asyncSocket_view2 readDataWithTimeout:-1 tag:0];
}

- (void)socket:(GCDAsyncSocket *)sock didConnectToHost:(NSString *)host port:(UInt16)port
{
        NSLog(@"socket:didConnectToHost:%@ port:%hu", host, port);
}

- (void)socket:(GCDAsyncSocket *)sock didWriteDataWithTag:(long)tag
{
    NSLog(@"did write data");
}

- (void)socketDidDisconnect:(GCDAsyncSocket *)sock withError:(NSError *)err
{
    if(err)
    {
        NSLog(@"error == %@",err);
    }
    else
    {
        NSLog(@"without error");
    }
}
```

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (IBAction)ShutDown:(id)sender {
     exit(0);
    [asyncSocket_view2 disconnectAfterWriting];
}
@end
```

View One: (Login View)

Interface Class:

```
#import <UIKit/UIKit.h>

@class GCDAsyncSocket;
@interface ViewControllerNEW : UIViewController
<UIPickerViewDataSource,UIPickerViewDelegate, UITextFieldDelegate>{
    NSMutableArray *list_gender;
    NSMutableArray *list_theme;
    NSMutableArray *list_accuracy;
    UIPickerView *pickView_gender;
    UIPickerView *pickView_theme;
    UIPickerView *pickView_accuracy;

    NSString *serverIP;
    int serverPort;
    NSString *select_theme;
@public
    GCDAsyncSocket *asyncSocket;
    BOOL isChangeToViewController3;
}
@property (retain, nonatomic) IBOutlet UIPickerView *pickView_gender;
@property (retain, nonatomic) IBOutlet UIPickerView *pickView_theme;
@property (retain, nonatomic) IBOutlet UIPickerView *pickView_accuracy;
@property (weak, nonatomic) IBOutlet UITextField *EmailTextField;
@property (weak, nonatomic) IBOutlet UITextField *PasswordTextField;
@property (weak, nonatomic) IBOutlet UITextField *textField_gender;
@property (weak, nonatomic) IBOutlet UITextField *textField_theme;
@property (weak, nonatomic) IBOutlet UITextField *textField_accuracy;
@property (nonatomic,retain) NSString *serverIP;
@property (nonatomic,retain) NSString *select_theme;
@property (nonatomic,assign) BOOL isChangeToViewController3;
@property (nonatomic,assign) int serverPort;

//appear pickerView
- (IBAction)touchDown_gender:(id)sender;
```

```
- (IBAction)touchDown_theme:(id)sender;
- (IBAction)touchDown_accuracy:(id)sender;

//button log in
- (IBAction)loginButton:(id)sender;
-(void)logInCheck;
-(void)getServerIP;
-(void)startRead;
-(void)passDataToServer;
-(void)setterOfTimeAccurary:(int)timeAccu;
-(int)timeAccuracy;
-(void)setterOfResponse:(NSString *)response_function;
-(NSString *)response;
@end
```

Implementation Class:

```
#import "ViewControllerNEW.h"
#import "GCDAsyncSocket.h"
#import "DDLog.h"
#import "DDTTYLogger.h"

static const int ddLogLevel = LOG_LEVEL_VERBOSE;
static float timeAccuracy;
static NSString *response;

@implementation ViewControllerNEW
@synthesize EmailTextField;
@synthesize PasswordTextField;
@synthesize pickView_gender;
@synthesize pickView_theme;
@synthesize pickView_accuracy;
@synthesize textField_gender;
@synthesize textField_theme,select_theme;
@synthesize textField_accuracy;
@synthesize serverIP,serverPort;
@synthesize isChangeToViewController3;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
        isChangeToViewController3 = FALSE;

        EmailTextField.text = @"";
        PasswordTextField.text = @"";
    }
    return self;
}
```

```
- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField
{
    //change to password
    if([textField tag]==4)
    {
        textField.secureTextEntry = YES;
        [pickView_gender setHidden:YES];
        [pickView_theme setHidden:YES];
        [pickView_accuracy setHidden:YES];
        return YES;
    }
    else if(([textField tag]==1)||([textField tag]==2)||([textField tag]==3))
    {
        [textField resignFirstResponder];
        return NO;
    }
    else
    {
        return YES;
        [pickView_gender setHidden:YES];
        [pickView_theme setHidden:YES];
        [pickView_accuracy setHidden:YES];
    }
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField{
        [textField resignFirstResponder];
        return YES;
}

//picker view
-(NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;
{
    return 1;
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component
{
    if([pickerView tag]==1)
    {
        textField_gender.text= [list_gender objectAtIndex:row];
```

```
}
else if([pickerView tag]==2)
{
    textField_theme.text = [list_theme objectAtIndex:row];

    //send short string to server side
    if([textField_theme.text isEqualToString:@"Cyberpunk-netrunning"])
    {
        select_theme = @"cyber";
    }
    else
        if ([textField_theme.text isEqualToString:@"High Fantasy"])
        {
            select_theme = @"fantasy";
        }
        else
            if ([textField_theme.text isEqualToString:@"Survival Horror - Zombies"])
            {
                select_theme = @"horror";
            }

}
else if([pickerView tag]==3)
{
    textField_accuracy.text = [list_accuracy objectAtIndex:row];

    //send short string to server side
    if([textField_accuracy.text isEqualToString:@"Once per second"])
    {
        [self setterOfTimeAccurary:1.0f];
    }
    else
        if ([textField_accuracy.text isEqualToString:@"Once per minute"])
        {
            [self setterOfTimeAccurary:60.0f];
        }
        else
            if ([textField_accuracy.text isEqualToString:@"Once every 5 minutes"])
            {
                [self setterOfTimeAccurary:300.0f];
            }
            else
                if ([textField_accuracy.text isEqualToString:@"Once every 20 minutes"])
                {
                    [self setterOfTimeAccurary:1200.0f];
                }
                else
                    if ([textField_accuracy.text isEqualToString:@"Once every hour"])
                    {
                        [self setterOfTimeAccurary:3600.0f];
                    }
```

```
    }
}

//getter and setter of time accuracy
-(void)setterOfTimeAccurary:(int)timeAccu
{
    timeAccuracy = timeAccu;
}
-(int)timeAccuracy
{
    return timeAccuracy;
}


-(NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component;
{
    if([pickerView tag]==1)
    {
        return [list_gender count];
    }
    else if([pickerView tag]==2)
    {
        return [list_theme count];
    }
    else if([pickerView tag]==3)
    {
        return [list_accuracy count];
    }
    else
        return 0;
}

-(NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
forComponent:(NSInteger)component;
{
    if([pickerView tag]==1)
    {
        return [list_gender objectAtIndex:row];
    }
    else if([pickerView tag]==2)
    {
        return [list_theme objectAtIndex:row];
    }
    else if([pickerView tag]==3)
    {
        return [list_accuracy objectAtIndex:row];
    }
    else
        return 0;
}
```

```
#pragma mark - View lifecycle

// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad
{
    EmailTextField.delegate = (id)self;
    EmailTextField.returnKeyType = UIReturnKeyDone;
    PasswordTextField.delegate = (id)self;
    PasswordTextField.returnKeyType = UIReturnKeyDone;

    //default value
    timeAccuracy = 60.0f;
    select_theme = @"fantasy";

    //UIPickerView * 3
    //gender
    pickView_gender = [[UIPickerView alloc]initWithFrame:CGRectMake(0, 280, 320, 200)];

    pickView_gender.delegate = self;
    pickView_gender.dataSource = self;
    pickView_gender.showsSelectionIndicator = YES;
    pickView_gender.tag = 1;
    [pickView_gender setHidden:YES];

    textField_gender.delegate = (id)self;
    list_gender = [[NSMutableArray alloc]init];
    [list_gender addObject:@"male"];
    [list_gender addObject:@"female"];

    [pickView_gender selectRow:1 inComponent:0 animated:YES];
    [self.view addSubview:pickView_gender];

    //theme
    pickView_theme = [[UIPickerView alloc]initWithFrame:CGRectMake(0, 300, 320, 200)];

    pickView_theme.delegate = self;
    pickView_theme.dataSource = self;
    pickView_theme.showsSelectionIndicator = YES;
    pickView_theme.tag = 2;
    [pickView_theme setHidden:YES];

    textField_theme.delegate = (id)self;
    list_theme = [[NSMutableArray alloc]init];
    [list_theme addObject:@"Cyberpunk-netrunning"]; //cyber
    [list_theme addObject:@"High Fantasy"];//fantasy
    [list_theme addObject:@"Survival Horror - Zombies"];//horror

    [pickView_theme selectRow:1 inComponent:0 animated:YES];
    [self.view addSubview:pickView_theme];
```

```
    //frequency
    pickView_accuracy = [[UIPickerView alloc]initWithFrame:CGRectMake(0, 342, 320, 200)];

    pickView_accuracy.delegate = self;
    pickView_accuracy.dataSource = self;
    pickView_accuracy.showsSelectionIndicator = YES;
    pickView_accuracy.tag = 3;
    [pickView_accuracy setHidden:YES];

    textField_accuracy.delegate = (id)self;
    list_accuracy = [[NSMutableArray alloc]init];
    [list_accuracy addObject:@"Once per second"];
    [list_accuracy addObject:@"Once per minute"];
    [list_accuracy addObject:@"Once every 5 minutes"];
    [list_accuracy addObject:@"Once every 20 minutes"];
    [list_accuracy addObject:@"Once every hour"];

    [pickView_accuracy selectRow:1 inComponent:0 animated:YES];
    [self.view addSubview:pickView_accuracy];


    [super viewDidLoad];
}

- (void)viewDidUnload
{
    [self setEmailTextField:nil];
    [self setPasswordTextField:nil];
    [self setPickView_gender:nil];
    [self setTextField_gender:nil];
    [self setTextField_theme:nil];
    [self setTextField_accuracy:nil];
    self.pickView_gender = nil;
    self.pickView_accuracy = nil;
    self.pickView_theme = nil;
    self.serverIP = nil;
    self.select_theme = nil;
    [super viewDidUnload];
    // Release any retained subviews of the main view.
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
- (IBAction)touchDown_gender:(id)sender {
    //set tag==1
    [EmailTextField resignFirstResponder];
    [PasswordTextField resignFirstResponder];
    [pickView_theme setHidden:YES];
```

```
    [pickView_accuracy setHidden:YES];
    [pickView_gender setHidden:NO];
    textField_gender.text = [list_gender objectAtIndex:[pickView_gender
selectedRowInComponent:0]];
}

- (IBAction)touchDown_theme:(id)sender {
    //set tag==2
    [pickView_gender setHidden:YES];
    [EmailTextField resignFirstResponder];
    [PasswordTextField resignFirstResponder];

    [pickView_theme setHidden:NO];
    [pickView_accuracy setHidden:YES];
    textField_theme.text = [list_theme objectAtIndex:[pickView_theme
selectedRowInComponent:0]];
}

- (IBAction)touchDown_accuracy:(id)sender {
    //set tag==3
    [pickView_gender setHidden:YES];
    [pickView_theme setHidden:YES];
    [EmailTextField resignFirstResponder];
    [PasswordTextField resignFirstResponder];

    [pickView_accuracy setHidden:NO];
    textField_accuracy.text = [list_accuracy objectAtIndex:[pickView_accuracy
selectedRowInComponent:0]];
}

//connect with server
-(void)getServerIP{
    @try {
        NSString* urlString = @"http://cse.spsu.edu/jchastin/research/ParallelJourney/serverIP.txt";
        NSURL *url = [NSURL URLWithString:urlString];
        NSData *dataOfUrl = [NSData dataWithContentsOfURL:url];

        NSString *dataConverToString = [[NSString alloc]initWithData:dataOfUrl
encoding:NSUTF8StringEncoding];

        //pass the infor to ip and port variables
        NSArray *splitArray = [dataConverToString componentsSeparatedByString:@":"];
        serverIP = [splitArray objectAtIndex:0];
        serverPort = [[splitArray objectAtIndex:1]intValue];
    }
    @catch (NSException *exception) {
        UIAlertView *alert = [[UIAlertView alloc]initWithTitle:@"Error" message:@"Can not get
IP and Port" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
}
```

```
-(void)logInCheck{
      //connect
      NSLog(@"PROVO...");
      asyncSocket = [[GCDAsyncSocket alloc] initWithDelegate:self
delegateQueue:dispatch_get_main_queue()];
      NSError *error = nil;
      uint16_t port = serverPort;

      if (![asyncSocket connectToHost:serverIP onPort:port error:&error])
      {
         DDLogError(@"Unable to connect to due to invalid configuration: %@", error);

      }
      else
      {
         DDLogVerbose(@"Connecting...");
        if(isChangeToViewController3 == FALSE)
        {
           [self passDataToServer];
        }
      }
}

-(void)passDataToServer{
   //already connect to the server
   if(([[textField_theme text]length]>0) && ([[textField_gender text]length]>0) &&
         ([[textField_accuracy text]length]>0) && ([[PasswordTextField text]length]>0)&&
([[EmailTextField text]length]>0))
   {
      NSString* requestStr = [[NSString
alloc]initWithFormat:@"LOGIN:%@:%@:%@:%@:ParallelJourney\n",[EmailTextField
text],[PasswordTextField text],select_theme,[textField_gender text]];

      NSData *requestData = [requestStr dataUsingEncoding:NSUTF8StringEncoding];
      [asyncSocket writeData:requestData withTimeout:-1.0 tag:0];

   }
   else
   {
      //push alert to tell to enter the information
      UIAlertView *enterInformation = [[UIAlertView alloc]initWithTitle:@"Error Login"
message:@"Enter Information into text field." delegate:self cancelButtonTitle:@"OK"
otherButtonTitles: nil];
      [enterInformation show];
      enterInformation = nil;
   }

   [self startRead];
}
```

```
-(void)startRead {
   [asyncSocket readDataWithTimeout:-1 tag:0];
   [asyncSocket disconnectAfterReading];
}

- (void)socket:(GCDAsyncSocket *)sock didConnectToHost:(NSString *)host port:(UInt16)port
{
        DDLogVerbose(@"socket:didConnectToHost:%@ port:%hu", host, port);
}

- (void)socket:(GCDAsyncSocket *)sock didWriteDataWithTag:(long)tag
{
        DDLogVerbose(@"socket:didWriteDataWithTag:");
}

- (void)socket:(GCDAsyncSocket *)sock didReadData:(NSData *)data withTag:(long)tag
{
        DDLogVerbose(@"socket:didReadData:withTag:");

   if(tag == 0)
   {
      response = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];

      NSString *temp = [[NSString alloc]initWithFormat:@"-1\r\n"];
      if([response isEqualToString:temp])
      {
         UIAlertView *cannotConnectToServer = [[UIAlertView alloc]initWithTitle:@"Error
Login" message:@"Can not login.\nPlease check the username or password" delegate:self
cancelButtonTitle:@"OK" otherButtonTitles: nil];
         [cannotConnectToServer show];
         cannotConnectToServer = nil;
         temp = nil;
      }
      else
      {
         //transfer to next view
        [self performSegueWithIdentifier:@"transferViewController3" sender:self];
         temp = nil;
      }


   }
}

//getter and setter for response
-(NSString *)response
{
   return response;
}
-(void)setterOfResponse:(NSString *)response_function
{
```

```
    response = response_function;
}

- (void)socketDidDisconnect:(GCDAsyncSocket *)sock withError:(NSError *)err
{
        if(err)
    {
        DDLogVerbose(@"socketDidDisconnect:withError: \"%@\"", err);
    }
    else
    {
        DDLogVerbose(@"socketDidDisconnect:withoutError");
    }
}

-(void)socketDidCloseReadStream:(GCDAsyncSocket *)sock
{
    DDLogVerbose(@"close the read stream");
}

- (IBAction)loginButton:(id)sender {
    [self performSegueWithIdentifier:@"transferViewController3" sender:self];

    //get IP AsyncSocket
    [self getServerIP];
    [self logInCheck];
}
@end
```

Auto Sending Narratives by email Code:

```
namespace SendingEmailAutomatically
{
    public partial class Form1 : Form
    {
        SmtpClient SmtpClient_my = null;
        MailAddress MailAddress_from = null;
        MailMessage MailMessage_my = null;
        Attachment Attachment_my = null;

        public Form1()
        {
            InitializeComponent();
            MailMessage_my = new MailMessage();

            //add mouse event
            this.listBox1.MouseDoubleClick += new
MouseEventHandler(listBox1_MouseDoubleClick);
```

```
        }

        #region setUp Smtp server
        private void SmtpClientInit(string ServerName, int Port, string UseName, string password)
        {
            try
            {
                string ip = null;
                IPAddress[] IpAddress = Dns.GetHostEntry(ServerName).AddressList;
                Ping ping = new Ping();
                PingReply pingReply = null;

                foreach (IPAddress IP in IpAddress)
                {
                    pingReply = ping.Send(IP);
                    if (pingReply.Status == IPStatus.Success)
                    {
                        ip = IP.ToString();
                        break;

                    }
                }
                SmtpClient_my = new SmtpClient(ip, Port);
                SmtpClient_my.Timeout = 2000;

                NetworkCredential NetworkCredential_my = new NetworkCredential(UseName,
password);
                SmtpClient_my.Credentials = NetworkCredential_my;

                SmtpClient_my.SendCompleted += new
SendCompletedEventHandler(SmtpClient_my_SendCompleted);

            }
            catch (SocketException E)
            {

                MessageBox.Show(E.ToString());
                return;
            }

        }
        #endregion

        #region initialize the attachment of files
        private void Attachment_myInit(string path)
        {

            if (!File.Exists(path))
            {

                MessageBox.Show("{0}No file.", path);
```

```
            return;
        }
        try
        {
            FileStream FileStream_my = new FileStream(path, FileMode.Open);
            string name = FileStream_my.Name;
            int size = (int)(FileStream_my.Length / 1024);

            if (size > 10240)
            {

                MessageBox.Show("File length cannot exceed 10M, your file size is{0}",
size.ToString());
                return;
            }

            FileStream_my.Close();

            Attachment_my = new Attachment(path, MediaTypeNames.Application.Octet);

            ContentDisposition ContentDisposition_my = Attachment_my.ContentDisposition;
            ContentDisposition_my.Size = size;
            ContentDisposition_my.FileName = name;
            ContentDisposition_my.CreationDate = File.GetCreationTime(path);
            ContentDisposition_my.ModificationDate = File.GetLastWriteTime(path);
            ContentDisposition_my.ReadDate = File.GetCreationTimeUtc(path);

            MailMessage_my.Attachments.Add(Attachment_my);
        }
        catch (IOException E)
        {
            MessageBox.Show(E.Message);
        }
    }
    #endregion

    #region after sending the email
    void SmtpClient_my_SendCompleted(object sender, AsyncCompletedEventArgs e)
    {
        if (e.Cancelled)
        { MessageBox.Show("Asynchronous operation has been canceled"); }

        if (e.Error != null)
        {
            MessageBox.Show(e.UserState.ToString() + "Sending Wrong." + e.Error.Message,
"Sending Wrong.", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
```

```
            MessageBox.Show("Sending Successful", "Sending", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

    }
    #endregion

    #region button1_Click
    private void button1_Click(object sender, EventArgs e)
    {
        //intialize stream
        System.IO.Stream myStream;
        OpenFileDialog thisDialog = new OpenFileDialog();

        thisDialog.InitialDirectory = "C:\\";
        thisDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
        thisDialog.FilterIndex = 2;
        thisDialog.RestoreDirectory = true;
        thisDialog.Multiselect = true;
        thisDialog.Title = "Please Select Source File(s) for Conversion";

        if (thisDialog.ShowDialog() == DialogResult.OK)
        {
            foreach (String file in thisDialog.FileNames)
            {
                try
                {
                    if ((myStream = thisDialog.OpenFile()) != null)
                    {
                        using (myStream)
                        {
                            listBox1.Items.Add(file);
                        }
                    }
                }

                catch (Exception ex)
                {
                    MessageBox.Show("Error: Could not read file from disk. Original error: " +
ex.Message);
                }
            }
        }
    }
    #endregion

    #region listBox1_MouseDoubleClick
    void listBox1_MouseDoubleClick(object sender, MouseEventArgs e)
    {
        int index = this.listBox1.IndexFromPoint(e.Location);
        string selectedFile = "";
```

```
            string[] stringOnlyWithFileName;
            string[] stringOnlyWithFileName_two;
            string emailAddress = "";
            char[] delims = { '_' };
            char[] delims_two = { '.' };

            if (index != System.Windows.Forms.ListBox.NoMatches)
            {
                selectedFile = this.listBox1.Items[index].ToString();

                //add to attachment
                Attachment_myInit(selectedFile);

                stringOnlyWithFileName = selectedFile.Split(delims);
                stringOnlyWithFileName_two = stringOnlyWithFileName[1].Split(delims_two);
                emailAddress += stringOnlyWithFileName_two[0] + "." +
stringOnlyWithFileName_two[1];

                listBox2.Items.Add(emailAddress);

                //add to sending list
                MailMessage_my.To.Add(emailAddress);
            }
        }
        #endregion

        #region sending message
        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                SmtpClientInit(textBox3.Text, int.Parse(textBox4.Text), textBox5.Text, textBox6.Text);
                MailAddress_from = new MailAddress(textBox7.Text);
                MailMessage_my.Subject = textBox1.Text;
                MailMessage_my.Sender = MailAddress_from;
                MailMessage_my.From = MailAddress_from;
                MailMessage_my.Body = textBox2.Text;

            }
            catch (ArgumentException E)
            {

                MessageBox.Show(E.Message);
            }


            string userToken = "Well!";
            if (SmtpClient_my != null)
            {
                SmtpClient_my.SendAsync(MailMessage_my, userToken);
            }
```

```
            else
            {

                MessageBox.Show("Email not sending. Smtp not initialized.");
            }
        }
        #endregion
    }
}
```