PROJECT REFERENCE GUIDE ON

FOOD - CHAIN
A Location Based Mixed Reality Experience

Prepared By: ASHWINI KUMAR SAHU

<table>
<thead>
<tr>
<th>Supervisor</th>
<th>Dr. Adrian David Cheok, Associate Professor, School of Computer Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>School of Electrical and Electronic Engineering, NTU</td>
<td></td>
</tr>
</tbody>
</table>
ABOUT THE PROJECT

‘FOOD-CHAIN’ is an ongoing project by the Interaction and Entertainment Research Center (IERC), Nanyang Technological University, Singapore in collaboration with the Interactive Media Laboratory, University of Southern California, Los Angeles. This is one of the several projects based on Mixed / Augmented Reality Experience being undertaken by the IERC. Started in December 2005, this project is still under progress at both the sides and was recently exhibited at the Advancement in Computers and Entertainment (ACE) ’06 Conference held at Hollywood, California. The core project development divisions at the IERC include the Hardware development, graphics and communications development, Game Server & AI development and the Hardware Interfacing teams.

This project is expected to be completed by the end of 2006 and put on displays and exhibitions as well as sent to conferences by the beginning of next year.
ABSTRACT

Computer gaming and entertainment has been a major dominant force in the recent years due to increasing attractiveness and increased level of interaction among players and the gaming environment, AI etc. However it differs radically from physical gaming by the use of virtual environments and defining specific properties for them rather than using physical objects, our sense of space and spatial relations between objects. But, most computer oriented games nowadays focus the user's attention mainly on the computer screen or specific 2D/3D virtual environments, and players are bounded to remain within the virtual environment and interact with the help of keyboard, mouse or gaming hardware like joysticks etc. So as seen, this greatly reduces the level of interaction with external environment as well as social interaction. The need to extend digital gaming beyond standard desktop environments is well recognized by CHI experts through the introduction of new forms of gaming. Tangible mixed reality gaming is one of the new forms of gaming that was introduced to assume a prominent role in fusing the exciting interactive features of computer gaming with the real physical world. An attempt has been made to model a Biological Food-chain through a location based Mixed reality experience.

In short, FOOD-CHAIN is an augmented reality game where 3 characters form the game-play with 2 of them being players and the 3rd one being an AI player. Together, the 3 characters form part of a biological food-chain and are in pursuit of ones immediately lower in the food-chain. The augmented reality creates an experience of physically interacting with virtual objects and creatures in the real environment, which does to a very good extent increase the level of interaction between the gaming environment and the players.

1. INTRODUCTION
With the advent of computing technology, gaming has taken a major position in the digital entertainment industry. Previously, before the widespread usage of computers in entertainment, games were designed and played out in the physical world with the use of real world properties, such as physical objects, our sense of space and spatial relations. In such physical games, interaction between players consisted of 2 vital elements: player to physical world interaction and player to player interaction. Presently, computer games provide a much higher level of attractiveness to players due to the possibilities of illusionary and imaginative virtual worlds with computer graphics and sound. Moreover the goals of computer games are typically more interactive than traditional games, and the optimal level of information complexity also provoke the players’ curiosity. Hence its an optimal combination of fantasy, challenge and curiosity which motivate people to opt for computer games.

However, one of the major disadvantages that still exist with computer games is the limitation in social and physical interaction. Natural interaction such as physical movement, behavioral states or interactions with the physical world are completely lost. The players’ attention is still focused mainly on the virtual environment which constitutes the entire game arena. Apart from this the level of physical interaction and interaction with the physical world is also constrained due to the usage of keyboard and mouse for all gaming purposes.

To address this issue, Tangible mixed reality / Augmented reality gaming developed as an exciting new area of Human-computer interaction which aims at fusing the dynamic interactive features of computer gaming with the real physical world with the sole aim to increase the level of interaction in gaming as well as to make the game ‘life-like’. Following this section is a brief introduction on Tangible Mixed Reality and Augmented Reality technologies and their applications in gaming and entertainment systems.
1.1 Tangible Mixed Reality in Gaming and Entertainment

Work on tangible mixed reality computing started to appear in the early 90’s with the introduction of Computer-Augmented Environments [*] that have visioned the merging of electronic systems into the physical world instead of attempting to replace them as in virtual reality environments. With the rapid advancement of computing technologies, some entertainment systems these days are capable of seamlessly integrating real and virtual worlds and provide a tangible interface with mixed reality objects. Compared to conventional computer games, tangible mixed reality gaming presents to its players more compelling experiences of physical and visual interactions.
Definition of some important terms to be encountered in Tangible Mixed Reality Application Development:

- **Mixed Reality** – It is the merging of real world and virtual worlds to produce a new environment where physical and digital objects can co-exist and interact. Probably a specialization of augmented reality and interactive media.

- **Augmented Reality** - Augmented reality (AR) is a field of computer research which deals with the combination of real world and computer generated data. At present, most AR research is concerned with the use of live video imagery which is digitally processed and "augmented" by the addition of computer generated graphics. Advanced research includes the use of motion tracking data, fiducial marker recognition using machine vision, and the construction of controlled environments containing any number of sensors and actuators. **Augmented reality** is an environment that includes both virtual reality and real-world elements. For instance, an AR user might wear translucent goggles; through these, he could see the real world, as well as computer-generated images projected on top of that world. Azuma defines an augmented reality system as one that
  - combines real and virtual
  - is interactive in real time
  - is registered in 3D

- **Virtual Reality** - Virtual reality (VR) is a technology which allows a user to interact with a computer-simulated environment. Most virtual reality environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays, but some simulations include additional sensory information, such as sound through speakers or headphones. Some advanced and experimental systems have included limited tactile information, known as force
feedback. Users can interact with a virtual environment either through the use of standard input devices such as a keyboard and mouse, or through multimodal devices such as a wired glove, the Polhemus boom arm, and/or omnidirectional treadmill. The simulated environment can be similar to the real world, for example, simulations for pilot or combat training, or it can differ significantly from reality, as in VR games.

In 1994, Paul Milgram and co-workers [1] in their work titled “A Taxonomy of Mixed Reality Visual Displays” identified how augmented reality and mixed reality work are related. The real world and a totally virtual environment are at the two ends of this continuum with the middle region called Mixed Reality (MR). Augmented Reality (AR) lies near the real-world end of the line with the pre-dominate perception being the real-world data augmented with computer generated objects. Augmented Virtuality on the other hand is used to identify systems which are mostly synthetic with some real world imagery added such as texture mapping etc.

![Milgram’s Realty-Virtuality Continuum](image)

**Figure 1: Milgram’s Realty-Virtuality Continuum**

1.2 Indoor Vs. Outdoor AR Gaming
Currently there are a wide variety of different augmented and mixed-reality entertainment applications being developed which vary in approaches and objectives. However, the focus still lies on Indoor AR Application development as compared to outdoor AR applications, primarily because of the different position tracking technology used in both types of applications. Tracking is essential for all AR Applications, including games because virtual objects need to augment into the real world objects correctly for optimal AR effects. Having a low degree of accuracy in an AR system can make the system very confusing to the user because of the incompatibility of the real and virtual world objects which will be of no use to the user.

Most of the Indoor AR Games and applications in place nowadays, like AR Hockey, Aqua Gauntlet, and Tiltpad Pacman use camera-based physical marker tracking and computer graphics to generate virtual objects in real–space as well as control the interaction between players and the virtual objects. But as the case is, the level of physical interaction with the real-world is much reduced in such games. On the other hand, outdoor AR games depend primarily on the position tracking by integrated GPS unit as well as orientation tracking using special orientation trackers like *Intersense Inertia Cube*. AR Quake was the 1st outdoor AR Game where the erstwhile popular AR game Quake was converted to an AR environment for the player using a wearable computer, GPS Tracking and a digital compass for direction, thus allowing the user to search enemies and objects by simply moving and looking around in open space. Other outdoor games like Human Pacman allow multiple users to play in the outside AR environment, emulating the game play like the original arcade version of Pacman.

*FOOD-CHAIN* is typically a combination of an outdoor and indoor AR Gaming experience where one of the players (the **Terran**) plays the game in the open space moving around and collecting AR objects like health pills and special powers as well as consuming AI prey from specific ‘eco-systems’. Hence the Terran player constitutes the
outdoor component of the game. His position and location are tracked using a GPS Unit connected to the wearable computer present with him as well as an InertiaCube Orientation Tracker for estimating the 3 DOF orientation of the Terran player. In the indoor part, the 2\textsuperscript{nd} player namely the \textbf{Avian} plays the game in front of a 3D Stereoscopic display in the lab. He involves in the gameplay through the Avian Controller PC which has a 3D Representation of the game arena and positions the Terran, Avian as well as the AI Predator in it. To make a more realistic experience to the \textbf{Avian} player he is provided with active force feedback jackets and weighted shoulder pads that are able to create the feeling of lift-off and landing when the Avian player takes off the ground or is forced to land.

### 1.3 Aims and Research Issues of FOOD-CHAIN

The main aim of the FOOD-CHAIN Project is to create a 2-player Augmented Reality Game which will involve partly an indoor and an outdoor AR game component and gameplay. Some of the key achievements being aimed at are:

1. The application will be played in the physical world with one player and in the virtual world with another player.

2. The point of view of the outdoor player is completely positioned in the virtual game space hence making the game coherent for the virtual world player. Similarly, sound effects and real-time communication between the virtual world player and the real-world player makes the game coherent for the real-world player.

3. Relevant information is displayed for the physical world player (To be called Terran henceforth) as augmented reality via a HMD. This includes the ‘Virtual Ecosystems’ where the Terran is supposed to look for preys and special powers. For the Virtual-world player (To be called Avian henceforth) the same information is displayed on
the stereoscopic display in front of him about the exact positions of itself and all the other players in the game arena.

4. The players are mobile in the context of the game arena and are able to traverse the game space.

5. The application is a combination of real and virtual environments along with virtual players & participants (AI Predator and AI Preys) apart from human players.

To achieve these aims, we are investigating a number of research issues in the areas of user interfaces, tracking and conversion of existing desktop applications to AR environments. User-interfaces for AR applications which simultaneously display both the physical world and computer generated images require special attention. The choice of screen colors for the virtual objects like ‘Ecosystems’ and ‘AI Preys and Special Powers’ need attention to the lighting conditions and external backgrounds as they are being combined with the background which are outdoor images of a campus. The keyboard and mouse functions should be replaced with special buttons to perform the tasks required by the Terran player.

The tracking of the Terran player in the game space is another important issue to be looked into. We need a fairly accurate 6DOF tracking system for the Terran player. Tracking is required over the campus environment where the Terran player is free to move. We are using a Garmin GPS Unit which has an accuracy of upto 50m. Apart from this we also use an InertiaCube Orientation Tracker for tracking the orientation of the Terran player. Another issue that needs to be looked into is the design of the interface for the Terran player. The player’s movement needs to be calibrated with the game server as well as with the virtual representation of the game arena at the Avian player’s side. Moreover both the Terran and the Avian should be provided with suitable key / button options to control their actions and powers.
1.4 Progress of the Project

FOOD-CHAIN is an ongoing project and is not complete at the time of writing of this reference. At present both the teams have made considerable progress in the design and development of the different modules of the game. Some of the important modules being worked on currently at NTU (Terran player and Game server) are:

1. **Game Server** – The main game server of the application is being developed with the architecture in place and programming going on for the same. The entire application consists of 2 pc’s with the Avian player’s application and the game server running on 1 pc and the Terran player’s application and subsequent processing being handled by the wearable computer with the Terran.

2. **Tracking system** – The tracking system for the Terran player is being currently worked on. The Garmin GPS unit and the InertiaCube-3 Orientation tracker have already been interfaced with the wearable computer and the data gathering and analysis through these instruments is being developed at the moment.

3. **Audio and Communication between the players** – The audio conversation between the Terran and Avian needs to be specially built with positional effects in place. This is being currently investigated and worked on. A testbed UDP based VOIP application is already in place to test for positional sound effects and build other audio features.

4. **RFID ‘Ecosystem’ Tracking** – This is another part which is under progress, involving the RFID tag based tracking of the ecosystems on campus by the Terran player. The RFID Tag reader has been constructed and been tested successfully using RFIC tags. Currently, the Bluetooth data transfer of the RFID reader to the wearable computer is being worked upon.
2. PROJECT DESCRIPTION: FOOD - CHAIN

2.1 Logline and Description
FOOD-CHAIN is an augmented reality game, where one player attempts to catch the other while avoiding AI predators. One player (the Terran) is outdoors, running around the USC Campus, while the other (the Avian) flies around a 3D model of the USC campus which serves as the game-arena.

FOOD-CHAIN is a 2-person mixed reality experience in which each player is both predator and prey in a hybrid virtual/physical space. The Terran is outdoors, carrying a pair of AR binoculars through which he can search for virtual ‘Ecosystems’; the other player, Avian is indoors, flying around a virtual model of the outdoor space.

The Game Concept – A Biological Food-Chain: This game has been conceptualized to make players be aware of the Food-chain around us and feel to be one with it, which forms such a vital part of nature. The Avian Predator is just above the Terran Prey on the Food-chain; it is trying to catch and consume the Terran, while avoiding a virtual AI Predator to whom it is a prey itself. Meanwhile, the Terran tries to catch and consume virtual AI Prey in the ‘Ecosystems’ on campus while avoiding the attentions of the Avian Predator. Thus the 2 players have a specific, non-symmetrical relationship, even as both players experience the dual nature of being both the hunter and the hunted.

A number of media technologies are used to create a rich shared experience for the 2 players. Their positions are tracked, and they can communicate verbally with each other at all times. The Terran is on with a wearable computer, and carries a pair of see-through stereoscopic AR binoculars. He also wears an Orientation Tracker on his head to mark his orientation. He is tracked by GPS, and wears open-air headphones, as well as a jacket with active force-feedback shoulderpads that “pinch” his shoulders whenever he is caught by the Avian predator. An RFID reader worn around the ankle, allows the Terran to be precisely located when he steps on unique RFID tags that are embedded in various locations on the ground also serving as the ‘Ecosystem’ locations.
The Avian is indoors, experiencing the shared space as a large stereoscopic projection. Its full body position and orientation are tracked by an optical system, allowing it to navigate the virtual environment by swooping and gliding. It wears a jacket with weighted shoulderpads that are tethered to the ceiling; whenever it ‘takes off’, the shoulderpads are lifted slightly to create the relative feeling of weightlessness; conversely, when she runs out of energy and is forced to land, the full weight of the shoulderpads presses down on the player.

Outdoors, a number of positions on the ground are physically marked with large symbols. These represent locations where the Terran can find AI Prey. At the center of each of these symbols is an RFID tag. When the Terran positions himself on these symbols, his precise position is known. At these times, he can look through the AR binoculars to see into the ‘Ecosystem’ existing at that place and see virtual creatures superimposed over a video background of his location. He catches these creatures without moving, by shooting out his tongue, in the style of a frog.

2.2 Game Architecture and Design

The FOOD-CHAIN has been designed to accommodate 2 human players (The Terran and the Avian) and 1 AI player called the AI Predator. A brief description of the design of each of the players follows:

1. **Predator POV (AVIAN)** – The Avian Predator flies around the 3D Model and tries to locate the Terran Prey, who is represented by a small 3D icon on the 3D Map. The Avian Predator can then swoop down on the Terran Prey and ‘consume’ him. The goal of the Avian Predator is to catch the Terran Prey 3 times, in which case the Avian Predator succeeds in eating the Terran Prey. However, the Avian Predator must avoid being caught by an AI Predator who is trying to eat him.
2. **Prey POV (TERRAN)** – The **Terran** Prey runs around a section of the USC Campus that has been physically marked with symbols on the ground that represent ‘Ecosystems’. These ecosystems can only be located through the use of the AR (Augmented Reality) binoculars. A 3 DOF Orientation Tracker is also mounted on the backpack of the Terran allowing the Terran’s orientation to be precisely sensed. When the Terran Prey reaches an ‘Ecosystem’, he looks through the AR Binoculars and sees small digital AI Preys running around which he catches by snapping them with his tongue. To catch the AI Preys in the ecosystem, the Terran aims at them using his AR binoculars and presses a specific button so that his tongue extends and catches the creatures. However, the Terran Prey must not linger at the spot otherwise he will soon become visible to the Avian Predator.

3. **AI Predator** – The **AI** Predator is a bot programmed to chase after the **Avian** Predator. It walks along the ground trying to catch the Avian Predator when it isn’t flying. The AI Predator can also eat the Terran Prey if the Avian Predator player is not participating in the game.

The game software architecture that has been developed has been presented below:

The main Data structure and Objects of the Food-Chain Game are:

**Protocol**

The server will be designed using TCP/IP Protocol.

**Platform** - Windows XP

The basic data needed to be transferred between Server/Avian and Server/Terran.

**Client Type** - Avian, Terran

**AI Predator Flag** - True for Avian, False for Terran

**Terran Positions**
Terran Status - *Visible Flag*: True for visible, false for invisible; *Life number*: Initial number is 3, increase one point when capture 5 preys in an ecosystem, the up-limit is 8; *Captured flag*: Last 10 seconds (for shoulder pressure)

**Ecosystem Status**

*Label; Captured prey number; Active Flag*: will be active if Terran is in the particular ‘Ecosystem’

**The Data Structure Design is as follows:**

```c
enum PLAYER_TYPE;  //players’ type used for server to verify which client is connected
{
    AVIAN, TERRAN;
}
typedef int ClientId;
typedef int ClientType;
typedef int Lifepoint;
typedef bool PlayerStatus;

typedef struct
{
    float position;
    float angle[3];
} ClientLocation;

typedef struct
{
    bool status;  //true - active; false - inactive
    int num;      //prey being captured
} EcosystemStatus; // Data Transfer between Terran and Server
```
typedef struct {
    float position;
    bool flag; // true for Avian, false for Terran, update from server
} PredatorData;

typedef struct {
    LifePoint point; //initialize according to client type and update from server
    PlayerStatus status; //update from each client
    Bool captureFlag; //true for being captured, update from server
} PlayerStatus;

typedef struct {
    ClientId id; //set by client and update to server when connect
    ClientLocation loc; //update from each client in real-time
    PredatorData predator;
    PlayerStatus player;
} ClientData; // transfer between Terran/Avian and the Game Server

2.3 Game Terms

Some of the main terms used in the FOOD-CHAIN game are described as follows:
1. **Avian Predator** – This player flies around the 3D Model of the USC campus, trying to catch the Terran Prey character. He must also avoid being eaten by the AI Predator which is above him in the Food-Chain.

2. **Terran Prey** – This player runs around the USC campus seeking ‘Ecosystems’ in order to feed, while avoiding being eaten by the Avian Predator.

3. **AI Predator** – A virtual creature that is hunting the Avian Predator.

4. **Ecosystem** – An oasis-like position on the USC campus that has been overlaid with digital information that the Terran Prey can see. When the Terran Prey reaches the Ecosystem by stepping on certain RFID markers on the ground, he can see and then consume virtual prey that are visible at that location.

5. **AI Prey** – Virtual AI creatures that appear only when the Terran Prey reaches an Ecosystem.

---

### 3. HARDWARE & SOFTWARE DESIGN

#### 3.1 Hardware Design & Implementation
FOOD-CHAIN features a centralized client-server architecture that is made up of 3 main entities namely, the Game server, the Terran’s wearable computer and the Avian’s system. An overview of the system is shown in Figure 4. Communication between game clients and server is maintained using Wireless LAN (IEEE 802.11 b/g) / Mobile 3G Network.

![Figure 4: An Overview of the System Design of FOOD-CHAIN](image)

Figure 5 shows an actual game play of FOOD-CHAIN, involving the real person playing the Terran’s role, the first person’s (Terran) view and the 3D virtual representation of the game-space. This has been taken from another similar game known as Human-Pacman, but FOOD-CHAIN will be very similar in terms of design and implementation. In an outdoor environment, the system uses Real-time Kinematic GPS technology to track the position of the Terran. The position coordinates of the Terran is transmitted to the server every 1 sec, which will further be transmitted to the Avian’s system, so that the position of the Terran and Avian is correctly reflected in the 3D Virtual environment at the Avian’s side.
Since, we are currently focusing in the development of the Terran player’s system at NTU, hence we shall focus on the Terran’s wearable computer design in detail.

### 3.1.1 Wearable Computer Hardware Design

Figure 6 shows an overview of the wearable computer system to be used by the Terran system. A description of the major components of the system is as follows:

1. **Laptop** – We are using an Acer Travelmate 4001LMi Notebook pc with a Pentium 4 Processor of 1.5 GHz and an ATI Radeon 9700 Graphics Card. A high-end graphics card is necessary for the frame rate and the augmented reality display to be acceptable (about 30 frames per second). The price, number of USB ports and external network interface card capacity etc. This is the same system that was used for the wearable computer design for the Human-Pacman Project.
2. **USB Cameras** – Logitech 4000 USB cameras support upto 640 x 480 pixels of resolution and upto 30 Hz of frame rate. Hence they are ideal for use in the wearable computer system to act as a video input for the surroundings to be displayed on the AR binoculars / HMD. The camera is mounted on the AR binoculars and is powered by the USB port hence no external source is required.

3. **Intersense InertiaCube**[^3] – This is used to track the user’s head orientation (Yaw, pitch, roll) with a dynamic accuracy of 3° RMS. It is hence a 3-DOF orientation tracker. It has a low latency of 2 ms and high refresh rate of 180 Hz thereby allowing for the real-time update of the Terran’s orientation. It is powered directly via the USB port of the notebook using a USB to serial adaptor as the InertiaCube[^3] operates via a serial port.

4. **Bluetooth USB Dongle – Class 1** (Upto a range of 10m) Bluetooth communication is required between the RFID Tag Reader worn by the Terran and the wearable computer to transfer the RFID tag data (‘Ecosystem’ related information) that is read from the RFID Transponder tags on the ground by the RFID tag reader. It can be provided for 7 dedicated Bluetooth connections at the same time and hence makes the system more versatile. Its low power consumption is another advantage.
5. GPS Module – We are using the Triman GPS Module to track the position of the Terran player while he is moving on the USC campus searching for ‘Ecosystems’. The GPS module follows the NMEA standard for measuring of position data. The GPS module serves to acquire readings from the satellites to receive positional data that are crucial due to the nature of the game. It is generally broken down into 3 parts:

I. Serial communications module - This module takes care of the link between the GPS receiver and the com port that is reading data from it. It involves initializing com port settings, reading and writing through the com port, and closing the com port when the GPS module is not required. The serial communications module also acts as a listener to constantly receive data from the GPS receiver.

II. GPS Interpreting module - GPS data comes in different NMEA sentences and each is to be interpreted differently according to the sentence types. The interpreter module extracts the respective data into their respective data structures and makes them readily available for the other modules using these data.

III. XY Mapper module - The XY Mapper works as a data transforming module that takes in geodesic coordinates and maps them into Cartesian coordinates. It involves conversion of changes in geodesic coordinates to changes in X Y
distances in metres. The X Y distances are used to map positions in Cartesian coordinates which the game module uses to track players around the campus.

6. **RFID MicroReader with Bluetooth** – This is the RFID reader which is used to read the RFID tags embedded on the ground at specific locations to determine ‘ecosystems’. It connects to the wearable computer via a Bluetooth connection. A more detailed description of the RFID system is given later in the chapter while describing the RFID Module.

7. **Twiddler** – This device is used as a joystick for the **Terran Prey** to consume other digital AI preys inside the ‘ecosystems’ as well as collect special powers and objects while moving around. It contains several buttons which can be used to control actions like projecting out tongue, picking up object etc.
3.1.2 Current Progress of Modules

Currently, many of the modules are being developed. The Game server is being developed as well as the Terran’s side is in progress. The audio system between the Terran and Avian is also in progress along with the tracking modules of the Terran. Apart from this, the RFID Tracking modules are also being developed. 2 main modules, namely the sound system and the RFID Reader module are being discussed next. A short tutorial on the development and use of the modules are being emphasized. Further modules being developed can be added to this section with ease.

3.2 Description of Modules of the Wearable Computer – A Tutorial

This section will contain a detailed description of the different modules of the Terran’s side built till date. It will give an idea on the hardware and software design of the modules and serve as a tutorial for readers to get acquainted with the development of
the system. Further modules upon development as well as the current modules can be changed in the following sections as and when necessary.

3.2.1 The Audio Communication Module for the Avian and Terran players

The Avian and the Terran players have been designed to communicate with each other in course of the game. They will be able to talk to each other to increase the level of social interaction in the game as well as to make the game play more competitive and interesting. It thus makes the game more realistic in context of a computer game as it combines human voices with computer technology thus following the sole purpose for making a mixed-reality game.

The main purpose of the audio communication module here is to add in the Mixed Reality effect for the Terran. The Terran player though cannot see the Avian Predator, but it can hear the Avian’s voices corresponding to the distance between the two in the virtual game arena. Hence, depending on the distance between the Avian and the Terran on the 3D Virtual map or the gamespace, the voice data between the two sides will be modulated accordingly to have a positional effect on the sound. One of the elementary ways in which it is aimed to be done is by changing the volume of the sound heard by each side, depending on the distance between the two. In the later stages, more advanced sound processing can be done using specialized API’s like FMOD Ex 4.0.3 to create a more realistic 3D sound effect.

At present, we have developed an open-ended UDP based VOIP protocol, which can be used to send and receive microphone voice data on either sides, i.e. the Avian and the Terran’s sides. The main objective of this protocol is to capture microphone audio and encode it using a speech compression/de-compression codec (SPEEX in this case), and then encapsulate it inside a UDP packet. This is done using the FMOD Ex 4.0.3 API. After this a UDP connection is created between the two terminals in the network and the packets are sent over to either sides. Over at the
receiving end, the UDP packets are read and the data is decompressed using the SPEEX audio-codec, and let out through the soundcard output.

This tutorial will contain the software design and a brief description of the software codes used to build the VOIP protocol. We use a combination of UDP Protocol Stack of Visual C++ along with the FMOD API's system and sound objects to create and manipulate the audio communication.

**SOFTWARE ARCHITECTURE**

**FMOD Ex Sound System:**

The FMOD Ex 4.0.3 sound system is a revolutionary new audio engine for game developers, multimedia developers, sound designers, musicians and audio engineers, based on the years of experienced of Firelight Technologies’ previous product FMOD. It also aims high - to push the boundaries of audio implementation for games and the like while at the same time using minimal resources and being scalable. Some of the key features of FMOD Ex are:

1. Suite of built-in DSP special effects

![Data Structure Design of the Audio Communication Module](image)
2. Sound designer focus and tool
3. Full 3D Sound support including various rolloff models, multiple listener support, occlusion and obstruction, stereo and support for multi-channel sounds played in 3D.
4. Virtual voices
5. Advanced streaming engine
6. Compressed sample playback
7. Object Oriented API supporting C, C++, C# and VB.

Please refer to the FMOD Ex 4.0.3 User Guide for further details on FMOD Ex usage.

** AudioManager Class:**

The prototype version of the module has been developed on **Ubuntu Linux 5.1.0**. As shown in Figure 9, the **AudioManager** class is the class that establishes the sound system as required by FMOD.

```cpp
class AudioManager {
    private:
        FMOD::System *system;
        FMOD_RESULT result;
        static AudioManager* audioManager;

        AudioManager();
        ~AudioManager();

    public:
        static AudioManager* GetUniqueInstance();
        FMOD::System* GetSystem();
};
```

**system** is the FMOD object which defines the whole environment where FMOD operates on audio functions. All the functions operate in the scope of the system object.

```cpp
result = FMOD::System_Create(&system);
ERRCHECK(result);

result = system->getVersion(&version);
ERRCHECK(result);
```

The above function creates the system object, and then checks the version of the current FMOD software installed. There are 3 sound output types available in Ubuntu, namely OSS, ALSA and ESD. The user has a choice of selecting any of the sound systems available in Linux. **AudioManager()** therefore initializes the system object of
FMOD and hence is the wrapper class for system. ~AudioManager() is the destructor which frees the FMOD system object. GetUniqueInstance() is one of the public methods of AudioManager class which returns a unique AudioManager object. GetSystem() is another public method of AudioManager which returns the currently available/active FMOD system object in scope.

**VoiceRecorder Class:**

As shown in Figure 9, the VoiceRecorder class is the class that initializes and runs the audio recording engine.

```cpp
public:
    VoiceRecorder();
    ~VoiceRecorder();
    void Start();
    void Stop();
    int GetRawBytes(char*);
```

These are the public methods defined in VoiceRecorder class. VoiceRecorder() is the constructor which is the initialization routine of the VoiceRecorder. ~VoiceRecorder() frees the VoiceRecorder. Start() and Stop() correspondingly start and stop the recording of data through the microphone. GetRawBytes(char*) is the function which performs the capture of raw data bytes from a stream of data during the recording. These raw data bytes are converted into UDP data packets and then encoded through SPEEX 1.0.5 and then transferred from the Terran to the Avian’s end and vice-versa.

```cpp
void VoiceRecorder::Start()
{
    recordpos = 0;
    lastrecordpos = 0;
    sound->getLength(&soundlength, FMOD_TIMEUNIT_PCM);

    result = system->recordStart(sound, true);  //true means the sound object will be overwritten every loop
    ERRCHECK(result);
}
```

As mentioned previously the Start() method starts the audio recording from the audio device on the system. It gets the length of a particular audio sample in PCM time-units as shown in
Then the FMOD system starts recording a sample into the sound object. FMOD_OK is returned by these inbuilt functions upon success. ERRCHECK(result) checks for the error in the executions of the methods if any. This is followed by the GetRawBytes(char* rawBytes) method which captures the current audio sample and extracts the audio data. With this function we get access to the RAW audio data, for example 8, 16, 24 or 32bit PCM data, mono or stereo data, and on consoles, vag, xadpcm or gcadpcm compressed data.

system->getRecordPosition(&recordpos); //Retrieves the current recording position of the record buffer in PCM Samples

blocklength = (int)recordpos - (int)lastrecordpos;

**Locking and Unlocking of the sound object ::**

//Lock the sound to get access to the raw data

sound->lock(lastrecordpos * sizeof(short), blocklength * sizeof(short), &ptr1, &ptr2, &len1, &len2);

/* *4 = stereo 16bit. 1 sample = 4 bytes. */

/*
Write it to disk.
*/

if (ptr1 && len1)
{
    memcpy(rawBytes, ptr1, len1); //copying into memory the raw data bytes extracted
}
if (ptr2 && len2)
{
    char *dest = rawBytes + len1;
    memcpy(dest, ptr2, len2);//Destination location of audio data for the particular sample
}

/*
Unlock the sound to allow FMOD to use it again.Release previous sample data lock
*/

sound->unlock(ptr1, ptr2, len1, len2);
**Codec Class:**

As shown in the block structure in Figure 9, the Codec class performs the encoding and decoding of the audio data using the SPEEX 1.0.5 Codec implementation.

```c
int Codec::Encode(short* in, char *encodedArray, int length) {
    int numBytesEncoded;
    // Amplify and convert to float for encoding
    float volume = 40;
    for (int i=0; i<length/sizeof(short); i++)
        floatBuffer[i] = 40.0*static_cast<float>(in[i]);

    // Encode the voice data
    speex_bits_reset(&encoderBits);
    speex encode(encoderState, floatBuffer, &encoderBits);
    int nBytes = speex_bits_nbytes(&encoderBits);

    int encodedBytes = speex_bits_write(&encoderBits, encodedArray, nBytes);
    printf("Encoded:%d\n", encodedBytes);
    return 0;
}
```

In the above method, the raw audio bytes of `in` are converted into float format and then encoded using `speex_encode(encoderState, floatBuffer, &encoderBits)` method of SPEEX. `nBytes` stores the number of bytes encoded. Similarly `speex_decode()` performs the decoding of the encoded array at the receiving end.

```c
int Codec::Decode(char* encodedBytes, short *decodedBytes) {
    speex_bits_reset(&decoderBits);

    speex_bits_read_from(&decoderBits, encodedBytes, decoderFrameSize);
    speex_decode(decoderState, &decoderBits, outputBuffer);

    for (int i=0; i<decoderFrameSize; i++)
        decodedBytes[i] = static_cast<short>(outputBuffer[i]);
    // Explicitly cast the char into short in decodedBytes

    return 0;
}
```

`speex_bits_read_from(&decoderBits, encodedBytes, decoderFrameSize)` and `speex_decode(decoderState, &decoderBits, outputBuffer)` performs the decoding of the encodedBytes array.
**Transmitter Class:**

This class performs the wrapping of the raw audio data packets into UDP packets and also manages the sending over the UDP packets over the network. 

`Transmitter::Transmitter(const char* IP, int port)` is the constructor of the Transmitter class which creates the UDP port at the sending side.

```cpp
void Transmitter::SendPacket(char *data) {
    sendto(socketID, data, strlen(data), 0, (sockaddr*)&si_other, slen);
}
```

`sendPacket(char *data)` sends the `data` array over as a UDP packet.

**Receiver Class:**

This class has been created as a test purpose for testing the sending and receiving of UDP packets individually. Essentially same as the Transmitter class, it has the additional receive method for receiving the UDP packet.

```cpp
void Receiver::ReceivePacket(char* buf) {
    recvfrom(socketID, buf, 128, 0, (sockaddr*)&si_other, (socklen_t*)&slen);
}
```

**Terran Class:**

The main class of the whole system, the Terran class initializes and operates all the audio devices and network functions required for the transfer of audio data from one end to another.

```cpp
AudioManager *audio = AudioManager::GetUniqueInstance() – This method gets a unique and unused instance of the AudioManager class to function for this case.

FMOD::System *system = audio->GetSystem() – This method gets the current system object in scope and activates the system.
```
Transmitter *transmitter = new Transmitter("127.0.0.1",3000); - This method initializes the Transmitter object and creates the necessary UDP sockets ready for transmission.

VoiceRecorder *recorder = new VoiceRecorder(); recorder->Start(); - This initializes the VoiceRecorder and Start() starts the recording into the sound object.

numBytes = recorder->GetRawBytes(rawBytes);
if (numBytes>0)
{
    formattedBytes = reinterpret_cast<short*>(rawBytes); //type-cast rawBytes in char format to short
    codec->Encode(formattedBytes,encodedBytes,numBytes);
    printf("%d\n",strlen(encodedBytes));
    transmitter->SendPacket(encodedBytes);
}
Sleep(50);

The above code snippet comes within an endless loop and performs the encoding of the RawBytes obtained from the sound sample, and then after transmitting the audio bytes as UDP packets.

**Avian Class:**

Similar to the Terran class, the Avian class also initializes the various audio devices and network connections and sockets on the Avian’s side.

// Getting the UDP packets across the network and play them
for(;;)
{
    receiver->ReceivePacket(rawBytes);
    printf("Received packet\n");
    codec->Decode(rawBytes,decodedBytes);

    //Decode the char packet to Short(FMOD) packet
    for (int i=0;i<codec->GetDecoderFrameSize();i++)
        voipQueue.push((short)decodedBytes[i]);
    system->createStream(0, FMOD_2D | FMOD_SOFTWARE | FMOD_OPENUSER, &exinfo, &sound);
    system->playSound( FMOD_CHANNEL_FREE,sound, false, &channel);
}
The above code snippet in Avian class gives the primary difference between the Avian and Terran in the test stage. The Avian receives the UDP packets across the network, unwraps them to get the encoded audio data bytes and then after decodes it using the SPEEX codec again and plays it using the Avian's sound system.

Another important part that is still being worked upon is the **positional sound effects** in the receiving sounds. One simple way to do it as a prototype is to change the system volumes according to the distance between the Avian and Terran players in the virtual arena. This is being worked upon now and will be tested next.

### 3.2.1 The RFID ‘Ecosystem’ Tracking Module for the Terran

This is the second module being worked upon. To accomplish the Terran player to move around the USC campus searching for ‘ecosystems’, an RFID tracking system is being developed. It essentially consists of 2 important parts:

1. RFID Tag Reader with combined Bluetooth Promi
2. Bluetooth Dongle with a Serial Connection on the Terran’s wearable computer

The RFID Tag Reader structure is shown below in Figure 10. We use **Series 2000 Micro reader** by Texas Instruments to read the transponders placed under each display block. It reads which block it reads and sends it along with the player ID. It is fabricated like a thin pad, so that player can place the pad inside the shoe and circuit parts are placed securely inside a small pouch. We use the microcontroller PIC 16F76 to control the RFID reader & at the same time it communicates with the AI server via a dedicated Bluetooth connection.

After the RFID Shoe detects a particular tag, a Bluetooth Connection request is sent to the Bluetooth port of the wearable computer of the Terran. Once the Terran
computer accepts the Bluetooth connection, the RFID Tag data is transferred to the wearable computer by this serial connection. This RFID Tag data consists of the Ecosystem ID which helps the Game server to determine the location of the Terran on the game map. Moreover, this Ecosystem ID is also used to generate the Ecosystem and the Augmented Reality AI preys and objects in the particular ecosystem.

![Figure 10: RFID Tag Detection and Communication between Microcontrollers](image)

The RFID Shoe has been taken from a previously developed application called the **Age Invaders**, and its firmware flowchart is shown below.

![Figure 11: RFID Shoe Firmware Operation Flowchart](image)
After the initialization of the RFID Shoe and the tags, a serial COM port connection needs to be established with the wearable computer. The codes following describe the serial connection.

```c++
SERIAL_RESULT_TYPE CSrialConn::Open()
{
    char str[100];
    int i;

    for (i=0; i<m_nNumPort; i++) // In this case m_nNumPort = 1 since there is only one RFID Reader to connect to
    {
        sprintf(str, "\\\\.\\COM%d", m_pComPort[i]);

        // Creating the Port Handle which links the HANDLE object with the particular COM Port
        m_pPortHandle[i] = CreateFile (str, GENERIC_READ|GENERIC_WRITE, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
        if (m_pPortHandle[i] != INVALID_HANDLE_VALUE)
        {
            printf("COM%d opened.\n", m_pComPort[i]);
        }
        else
        {
            DWORD a = GetLastError();
            printf("ComPort opening error. %d\n", a);
            return SERIAL_CANNOT_INIT_PORT;
        }

        m_DCB.BaudRate = CBR_9600;     // set the baud rate
        m_DCB.ByteSize = 8;             // data size, xmit, and rcv
        m_DCB.Parity = NOPARITY;        // no parity bit
        m_DCB.StopBits = ONESTOPBIT;    // one stop bit

        COMMTIMEOUTS commTimeOuts;
        commTimeOuts.ReadIntIntervalTimeout = 50;
        commTimeOuts.ReadTotalTimeoutMultiplier = 50;
        commTimeOuts.ReadTotalTimeoutConstant = 50;
        commTimeOuts.WriteTotalTimeoutMultiplier = 50;
        commTimeOuts.WriteTotalTimeoutConstant = 50;
        int a = SetCommState(m_pPortHandle[i], &m_DCB); // Setting the Port Configuration using the Port Handle
        int b = SetCommTimeouts(m_pPortHandle[i], &commTimeOuts); // Setting the Port Time-outs
        if(!a || !b)
        {
            printf("Cannot configure port %d %d\n", a, b);
            CloseHandle(m_pPortHandle[i]); // Closing Port Handle
        }
    }
}
```
return SERIAL_CANNOT_INIT_PORT;

}
printf("Serial port COM%d successfully reconfigured.\n", m_pComPort[i]);
m_bConnected[i] = true;
}//FOR I

return SERIAL_OK;
}
Over here, a serial port is opened and configured on the Wearable computer. m_dcb is a port handle which is used to set the port properties like baud rate, byte size, parity and stop bits. COMMTIMEOUTS refers to the timing properties of the serial port like read interval timeout etc.

void CSerialConn::Read(int *buf, int i)
{
    unsigned long one[1];
    if (m_bConnected[i] == true)
    {
        int temp = 0;
        int count = 0;
        while(count < 2)
        {
            ReadFile(m_pPortHandle[i],&temp,1,one,NULL);
            if(one[0] == 1)  //1 byte read
            {
                if((temp >= 65 &&
                    temp <= 110)||temp == 33 || (temp >
                    48 && temp <= 57))
                    {
                        *(buf+count) = temp;
                        count++;
                    }
                else
                    count = 0;
            }
            Sleep(0);
        }
    }
    else
    {
        *buf = -1;
        *(buf+1) = -1;
    }
}
The above method is the Read(int *buf,int i) which reads from the Virtual serial port established between the wearable computer and the Bluetooth Promi on the RFID Reader. COM 6 or COM 7 are used for the same which are generally bound to the generic Bluetooth device of the computer.
As of now, some of the other testing on the RFID module is in progress to convert the data received from the RFID tag into forms which can be used to initiate the ecosystems, as well as update the Game Server of the location of the Terran.

![RFID Tracking Framework for the TERRAN'S Wearable Computer](image)

*Figure 12: RFID Tracking Framework for the TERRAN’S Wearable Computer*
4. GAME FLOW

A. Game Flow – Terran Prey

1. Terran begins quest to find ecosystems. When Ecosystem is reached, a noise occurs to alert the Terran prey that he is standing in the right place.

2. Terran’s AR readout tells him how many food creatures are in the area and how much time it has before it is visible to the Avian.

3. Terran begins to eat AI preys.

4. When Terran has eaten all the AI preys, he must run to the next ecosystem.

5. When running between ecosystems, the Terran is visible to the Avian Predator.

6. To win the game, the terran must feed from 5 different Ecosystems successfully.

7. The game can also end when the Avian Predator attacks the Terran 3 times.

A. Game Flow – Avian Predator

1. Avian Predator begins quest to find Terran Prey.

2. Avian Predator may begin to fly around the 3D model of campus. Meter displays how long he can fly for.

3. When Avian runs out of flight power he must rest on the ground for sometime.

4. Avian must avoid predation from AI Predator when it is resting on the ground.

5. Avian HUD (Heads Up Display) alerts him when the Terran Prey has left an ecosystem.

6. Avian then tries to catch the Terran.

7. If the Avian catches the Terran 3 times, the game is over.

8. The Avian Predator must avoid being caught by the AI Predator when it is resting or else the Terran wins the game.