Mobile Phone Paintball: A New Multiplayer Gaming Concept

by

Dongfan Kuang, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2011

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Dongfan Kuang

August 31, 2011

Permission to Lend and /or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Dongfan Kuang

August 31, 2011

Acknowledgments

I would like to thank my supervisor Dr. Kenneth Dawson-Howe for providing me the initial idea for this dissertation and for all his guidance and advices during the project.

I would also like to thank my family for standing by me in good and bad times throughout my studies. I love them and am forever grateful for their support.

DONGFAN KUANG

University of Dublin, Trinity College

September 2011

Mobile Phone Paintball: A New Multiplayer Gaming Concept

Dongfan Kuang

University of Dublin, Trinity College, 2011

Supervisor: Dr. Kenneth Dawson-Howe

The advanced hardware configuration combined with convenient use, rich UI and efficient application distribution system makes the current generation of smartphones a good platform for developing new mobile applications. The goal of this paper is to investigate a new mobile game input interface by developing a multi-player game called “Mobile Phone Paintball”, which uses camera, GPS as well as compass information as game input.

Before the game starts, players will be required to wear a specially designed hat with easily-distinguishable color to indicate which team they belong to. In order to win the game, players use their cell phone as a “gun” and attempt to take photos of their enemies to “kill” them. To implement this game, a color detection algorithm is developed to detect hats in the photo. GPS and compass information are used to help players find their enemies. A game server is also built to allow players to exchange information with each other.

Taking advantages of mobile phone camera, GPS and compass as game inputs can make the game more interactive and interesting, thus giving players a better gaming experience.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview and Motivations

With the world-wide proliferation of mobile technology, especially the rapid expansions of the smartphones market in recent years, smartphones are now playing an important role in people's daily life. The current generation of smartphones have remarkably powerful processors, high graphics quality and all kinds of build-in sensors and radios. They are an attractive platform for developing various kinds of game applications. The build-in sensors and radios in these smartphones make it possible to develop games using motion gestures and location information as game input to give player a better game experience, but most of the current mobile games didn't take the advantage of these new technologies and still using keyboard or touch screen as game input.

The goal of this project is to investigate and develop a new input interface based on mobile phone camera and other built-in sensors. To achieve this goal, a multiplayer Mobile Phone Paintball game, using mobile phone built-in camera, GPS and gyroscope sensors as game input to shoot other players, is supposed to be developed in my project. With the use of camera and GPS in the game, players could interact with the real world.

The framework implemented in this paper is made up of two key components: a game server running on a Mac and a game client application running on an iPhone. The game

server is responsible for managing all clients that are connecting to it. After the connection is established between the game server and game clients, the game clients can send request to game server and receive requested data from it. The game server has a database to store the players' data and a request handler to deal with the request. This database will be updated based on the clients' request.

In order to start a game, a player need to launch the game client application in his or her mobile phone and then login to the game server by using a player name. If login is successful, the player will enter the game lobby. A list of players that are currently connected to the game server will be displayed in here. Players in the game lobby need to choose a team they want to join and set their status to ready. The game will start when all players are ready.

The rule of Mobile Phone Paintball is the same as real paintball game. In a real paintball game, players use their paintball guns to aim and shoot their enemies. In order to kill an enemy in Mobile Phone Paintball, the players need to get close enough to the enemy and then use the camera in their mobile phone as a paintball gun to shoot the enemy. During the game, game clients will send GPS coordinates and heading of the current player to the game server. The game server will respond by sending back other players' GPS coordinates and heading. The game clients then could use these location data to help players find their enemies.

Due to the complex situation of outdoor environment and varies kinds of clothes the players could wear, it could be very difficult to directly recognize enemies through the camera. The way we used to solve this problem is to design special kind of hats for each team. Mobile Phone Paintball support three teams compete with each other at the same time, so three different kinds of hats are used in this game. Each kind of these hats is painted with two easily-distinguishable colors. When playing the game, players need to choose special hats to wear according what team they are in. Then the enemy detection problem is simplified to detect hat with two colors through the camera.

A triangle area in front of a player is defined as the sight range of the player. An enemy can

only be killed by a player if he or she is both detected by the player's mobile phone camera and also in the player's sight range. To decide whether there are enemies in the sight range of a player, we calculate distance and the bearing from current player to the other players by using their GPS coordinates and heading data which can be obtained from the game server. The game will end when there is only one team left.

1.2 Dissertation Layout

Chapter 2 reviews the state of the art that is related to this project. Chapter 3 shows the design and implementation details of the Mobile Phone Paintball game. The implementation results are shown in Chapter 4. The evaluation results are described in Chapter 5. Finally Chapter 6 gives the reader a conclusion and related future work.

Chapter 2

Background & Related Work

In this chapter we provide an overview of human tracking, face recognition and object recognition technologies in Section 2.1. Sections 2.2 - 2.5 reviews the state of the art with regard to the different kinds of mobile phone applications that are related to our project. Section 2.6 provides a brief introduction of iPhone and iOS.

2.1 Vision

2.1.1 Human Detection and Tracking

Detecting humans in images is a big challenge due to the variable pose, different illumination, and complex background. Generally speaking, detect a human in a static image is much more difficult than in a video. Detect a human in a video can be treated as moving object detection. This could be achieved by subtract background from image. But in a static image there is no motion information could be used to subtract background. So, feature extraction and classifier design become two important steps for detect human from a static image.

Dalal et al. [1] proposed a human detection algorithm by using histograms of oriented gradients (HOG). The method is based on evaluating well-normalized local histograms of

image gradient orientations in a dense grid. The basic idea is that local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions. So, the image window is divided into small spatial regions (cells), for each cell accumulating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. Then by accumulating a measure of local histogram "energy" over somewhat larger spatial regions (blocks) and using the results to normalize all of the cells in the block to generate the HOG descriptor. The last step is tiling the detection window with a dense (in fact, overlapping) grid of HOG descriptors and using the combined feature vector in a conventional SVM based window classifier to give human detection chain.

Rather than detect human in a static image, many practical applications require the camera moving to realize continuous tracking and expand monitoring range as well as reduce the cost. But the motion of the camera caused the movement of the whole background, compared to static background, it's much more complex and challenging to detect and track object real-time, especially when tracking non-rigid objects like human beings. Hu et al. [2] obtained a good tracking performance using joint spatial-color feature and improved Meanshift algorithm, but the algorithm computational cost was too high and it couldn't solve the problem when the target was seriously occluded.

From the view of the state of arts, there is no perfect algorithm for human detection to be adaptive to tough situations such as heavy shadow, sudden light change, tree shaking and so on. Most systems of human detection and tracking work fine in the environment with gradual light change, however, they fail to deal with the situation with sudden light change, and moving background.

2.1.2 Face Recognition

Many different frameworks have developed for solving the problem of face recognition, for example: Template Matching, Local Feature Analysis (LFA) and Principal Component Analysis (PCA). The choice of a particular solution is governed by its suitability in a

particular application.

Face recognition based on template matching [3] represents a face in terms of a template consisting of several masks enclosing the prominent features e.g. the eyes, the nose and the mouth. Matching is usually carried out by a correlation score computed from the pixel intensities of these masks taken from the reference, with the query image. Local feature analysis (LFA) [4] method of recognition is based on analyzing the face in terms of local features, e.g., eyes, nose, etc. In PCA method [5], also known as Eigenface method, face images are projected onto the so called eigenspace that best encodes the variations among known facial classes, and recognition is achieved by carrying out match of these projected feature vectors.

Face recognition from a single image is a challenging task because of variable factors like alterations in scale, location, pose, facial expression, occlusion, lighting conditions and overall appearance of the face.

2.1.3 Object Recognition

Due to the expensive computation and low recognition rate, human detection and face recognition are rarely used in mobile phone applications. The alternative method is recognizing objects by their color, shape and feature points.

Berg et al. [6] proposed recognition in the framework of deformable shape matching, relying on a new algorithm for finding correspondences between feature points. This algorithm sets up correspondence as an integer quadratic programming problem, where the cost function has terms based on similarity of corresponding geometric blur point descriptors as well as the geometric distortion between pairs of corresponding feature points. The algorithm handles outliers, and thus enables matching of exemplars to query images in the presence of occlusion and clutter. Given the correspondences, we estimate an aligning transform, typically a regularized thin plate spline, resulting in a dense correspondence between the two shapes. Object recognition is then handled in a nearest neighbor framework where the distance between exemplar and query is the matching cost

between corresponding points.

Sande et al. [7] studies the invariance properties and the distinctiveness of color descriptors in a structured way. First, taxonomy of invariant properties is presented. The taxonomy is derived by considering the diagonal model of illumination change. Then, the distinctiveness of color descriptors (SIFT, HSV-SIFT, HueSIFT, OpponentSIFT, W-SIFT, rgSIFT, Transformed color SIFT) is analyzed experimentally using two benchmarks from the image domain and the video domain. The benchmarks are very different in nature: the image benchmark consists of photographs and the video benchmark consists of news broadcast videos. Based on extensive experiments on a large set of real-world images and videos, the usefulness of the different invariant properties can be derived. The conclusion is the best choice for a color descriptor is OpponentSIFT when no prior knowledge about the dataset, object and scene categories is available. The corresponding invariance property is shift-invariance. When such knowledge is available, WSIFT and rgSIFT are potentially better choices. The corresponding invariance property is scale-invariance.

The purpose of [8] is to arrive at recognition of multicolored objects invariant to a substantial change in viewpoint, object geometry and illumination. Assuming dichromatic reflectance and white illumination, it is shown that normalized color RGB, saturation S and hue H, and the newly proposed color models c1c2c3 and l1l2l3 are all invariant to a change in viewing direction, object geometry and illumination. Further, it is shown that hue H and l1l2l3 are also invariant to highlights. But the existing color constancy methods require special priori information about the observed scene (e.g. the placement of calibration patches of known spectral reflectance in the scene) which will not be feasible in practical situations, so, a change in spectral power distribution of the illumination is considered to propose a new color constant color model m1m2m3. To evaluate the recognition accuracy differentiated for the various color models, experiments have been carried out on a database consisting of 500 images taken from 3-D multicolored man-made objects. The experimental results show that highest object recognition accuracy is achieved by l1l2l3 and hue H followed by c1c2c3, normalized color RGB and m1m2m3 under the constraint

of white illumination. Also, it is demonstrated that recognition accuracy degrades substantially for all color features other than m1m2m3 with a change in illumination color.

In [9] an approach was proposed to measuring similarity between shapes and exploits it as a basis for category-level object recognition. In order to solve the correspondence problem, a descriptor is attached, the shape context, to each point. The shape context at a reference point captures the distribution of the remaining points relative to it, thus offering a globally discriminative characterization. Corresponding points on two similar shapes will have similar shape contexts, enabling us to solve for correspondences as an optimal assignment problem. Given the point correspondences, transformation is estimated that best aligns the two shapes; regularized thin-plate splines provide a flexible class of transformation maps for this purpose. The dissimilarity between the two shapes is computed as a sum of matching errors between corresponding points, together with a term measuring the magnitude of the aligning transform. Recognition is treated in a nearest-neighbor classification framework as the problem of finding the stored prototype shape that is maximally similar to that in the image. Results are presented for silhouettes, trademarks, handwritten digits, and the COIL data set.

Chang and Krumm [10] proposed an object recognition approach with color co-occurrence histograms. In their study, the color co-occurrence histogram (CH) is used for recognizing objects in images. The color CH keeps track of the number of pairs of certain colored pixels that occur at certain separation distances in image space. The color CH adds geometric information to the normal color histogram, which abstracts away all geometry. We compute model CHs based on images of known objects taken from different points of view. These model CHs are then matched to sub-regions in test images to find the object. By adjusting the number of colors and the number of distances used in the CH, we can adjust the tolerance of the algorithm to changes in lighting, viewpoint, and the flexibility of the object. We develop a mathematical model of the algorithm's false alarm probability and use this as a principled way of picking most of the algorithm's adjustable parameters. We demonstrate our algorithm on different objects, showing that it recognize objects in spite of

confusing background clutter, partial occlusions, and flexing of the object.

2.2 Location Based Mobile Phone Application

2.2.1 Global Positioning System (GPS)

The GPS is a satellite-based radio navigation system designed financed, developed and operated by the U.S. department of defense as a military navigation and positioning system, but soon opened it up for civilian use. The object was to design and deploy an all-weather, 24h, global satellite-based navigation system to support the positioning requirements of the US armed forces and its allies [11]. GPS is based on a system of 24 satellites that orbit the earth. A GPS receiver can triangulate its position using Time of Arrival (TOA) methods as long as it knows the exact locations of the satellites, and what the distance to at least three or four satellites is within 300 kilometers. The satellites broadcast this information in a certain rate of frequency. It usually takes a GPS receiver as long as several minutes to achieve the mobile station location fix by starting without any knowledge about the GPS constellation's state [12]. GPS has a number of major restrictions, such as limited coverage in urban environments and inside buildings due to the requirement that cellular users be in "view" of the satellites, and a slow location acquisition time.

Assisted GPS systems overcome some of these limitations by using the cellular connection to transmit remotely-collected satellite navigation data from the base station to the cellular device. Using this fixed infrastructure can reduce acquisition time to less than 5 seconds, and possibly provide indoor accuracy to within 50m. However, as with the previous system, multipath and the lack of a line of sight measurement can degrade performance, and indeed A-GPS can be less accurate than GPS [13][14].

2.2.2 Wi-Fi Positioning

Wi-Fi is another attractive positioning technology due to the widely deployed Wi-Fi access

points (APs) and the growing number of Wi-Fi enabled mobile devices on the market. Wi-Fi positioning uses terrestrial based Wi-Fi APs to determine location. The density of APs in urban areas is so high that the signals often overlap, creating a natural reference system for determining location. Wi-Fi positioning software identifies the existing Wi-Fi signals within range of a Wi-Fi enabled mobile device and calculates the current location of the device. Wi-Fi positioning does not require that a connection be established to the Wi-Fi network: the Wi-Fi signals are only recorded in the form of their unique MAC address and signal strength at a particular location [15][16]. This allows Wi-Fi positioning to use potentially very weak signals, as well as encrypted signals, without having to establish a connection and it has excellent coverage and performance indoors.

2.2.3 Cellular Positioning

Positioning of mobile phones is an easy task for mobile operators, since they always know in which part of their network their customers' mobile phones are connected to [17]. One common cellular positioning technique is Time of Arrivals (TOA). TOA technology is depending on a triangle form; when it works, it needs at least three cell phone bases to receive the cell phone signal. After the signal arrives at the bases, each of the bases calculates the arrival time and then we can get the cell phone location. This technology is much more suitable in the city because there are more bases, which mean more accuracy [18].

2.2.4 iPhone Positioning

One criticism of the original iPhone was the fact it had no built-in A-GPS capability. This changed with the release of the 3G iPhone on 11 July 2008, which included a hybrid positioning system consisting of A-GPS, Wi-Fi and cellular positioning. The positional accuracy was characterized as 10 meters for A-GPS, 30 meters for Wi-Fi positioning and 500 meters for cellular positioning.

From a user's perspective, the positioning system of the iPhone switches seamlessly

between the three positioning modes. When a reliable A-GPS position fix is available, the location service publishes latitude, longitude, altitude and an estimate of positional error. When a reliable A-GPS position fix is not available the positioning mode is switched to Wi-Fi, and when no reliable Wi-Fi position fix can be obtained the positioning mode is switched to cellular. For both Wi-Fi and cellular positioning, the location service publishes latitude, longitude and an estimate of positional error but without altitude [9].

2.2.5 Related Works

Feeding Yoshi [20] tries to integrate playing a mobile game into the everyday life of the player. The goal is to find food for a creature named Yoshi. The localization in the game is performed by exploiting the WLAN infrastructures in cities where protected WLANs are Yoshis and open WLANs are used by the player to collect food. The WLAN access points are not used for communication with the game server. The player has to manually upload the game results.

One of the most well-known location-based services is PlaceLab [21]. In their experiment, LaMarca et al developed a system that was used for tracking wireless radio networks (W-LANs, Bluetooth etc.) and mapping them to the GPS coordinate system. In the calibration phase, when GPS enabled mobile terminal discovered a wireless network, it logged identifier of the discovered wireless networks with GPS coordinates to the server. Thus, when another user without GPS device was moving around already calibrated area, these coordinates were used as landmarks to locate the user.

Laasonen et al [22] used cell based positioning in GSM network to recognize users' important places from his/her daily routines. In their approach, GSM cells where user visited was logged to the database with a timestamp. This data was utilized for recognizing user's important places, the ones where she spent most of her time. The system was also able to determine user's relative position with respect to the recognized places, such as, if the user was between home and the workplace. However, the system didn't provide mechanism for assigning user's location to any fixed coordinates or landmarks. Thus, the

system was not able to tell whether the user is, for example, on the 11th or 16th street.

One reason for the location-based games to become more popular is the increasing amount of location-technology that is available for the consumers. However, there are still many problems with the location technologies. One big problem is GPS only work in outdoor, so if the player goes inside a building, the GPS signal may probability lost. Another problem is the accuracy of the localization data is depends on the environment a lot, especially in rough weather conditions, such as heavy rain.

2.3 Gesture Based Mobile Phone Application

Accelerometer combined with compass and gyroscope can interact with an application by using tiling, rotation and movement of the device as input. One big advantage of this interaction technique is its independence of the surround environment.

By using the mobile phones equipped with accelerometers, developers can use the gestures as a game input. There are many relevant work related to gestures recognition for accelerometer mobile. Choi et al. [23] used accelerometer data acquired from a mobile phone's built-in accelerometer to recognize digits from 1 to 9 and five symbols written in the air. During their study, they achieved a 97% average recognition rate. This was done by using a Bayesian network algorithm with its computation performed in the PC. Benbasat et al. [24] developed a system called ReachMedia which makes use of users' gesture data as input to interact with the system. In their approach, an axis-by-axis variance window with preset thresholds was used to detect the start and end of a gesture. In [25] a mobile application tool based on an 3-axis accelerometer, called SensorFall, was presented to detect and notify the acceleration caused by a fall. Accelerometers and gyroscope also could work together to estimate a user's facing. Hoseinitabatabaei et al. [26] presented an approach called uDirect which makes use of built-in accelerometer and gyroscope sensors in mobile phones to calculate the user's facing direction. Their algorithm is independent of the initial orientation of the device, which gives user higher space of freedom.

2.4 Augmented Reality Applications

Augmented reality (AR) is a term for a live direct or an indirect view of a physical, real-world environment whose elements are augmented by computer-generated sensory input, such as sound or graphics. A more concise definition is: Augmented Reality (AR), also known as "Mixed Reality" is where virtual and real objects appear together in a real time system. Location-based games are a subset of augmented reality games.

One good example of mobile augmented reality is the project proposed by Petri Honkamaa [27]. In this project, he use feature tracking for estimating camera motion when user turns the mobile device and examines the augmented scene, which uses GPS for defining the viewing location and Google Earth KML-files for defining the augmented object and its placement.

Another example is a foot-based mobile Interaction game proposed by Volker Paelke [28]. In this project Volker use the camera of current video capable mobile devices to detect motion and position of the player's foot to affect the input that is required for such games. The camera is used to detect "kicking" movements of the user's feet. When a collision between the user's "kick" and an interaction object shown on the screen of the mobile device is detected, a corresponding interaction event for the application is generated.

2.5 Multiplayer Mobile Phone Games

Multiplayer games or online games have been appeared in mobile phone for several years. A study of Nokia shows that 45% of the mobile game players play multiplayer mobile games at least once per month [29]. One of the biggest reasons for multiplayer game not being more popular in earlier day is the price of data traffic in mobile network. The latency in mobile network is another big problem for the games that requires fast player-to-player reaction. But now most of these problems are solved or in the way of being solved, i.e., the operators offer more multiplayer friendly data traffic price and the 3G networks helps the network latency and disconnections a lot.

A lot of the multiplayers games are use short-range communication technologies, such as Bluetooth or infrared. Bluetooth is a wireless protocol utilizing short-range communications technology facilitating data transmission over short distances from fixed and mobile devices. It is a standard and communications protocol primarily designed for low power consumption, with a short range (power-class-dependent: 1 meter, 10 meters, 100 meters) based on low-cost transceiver microchips in each device [30]. Bluetooth makes it possible for these devices to communicate with each other when they are in range. Bluetooth networks are ad-hoc and are often chaotic as many devices move in and out of Bluetooth range. A Bluetooth implementation requires a method of searching for devices, and finding out the capabilities of those devices. Once a device discovered another device then they can open a connection and exchange data. Any Bluetooth device initiating a connection is called the master whereas the device responding to the connection is the slave device. A master device can establish a piconet with 7 slave devices at most [31].

The huge advantage of multiplayer Bluetooth is it does not require any dedicated server. The player who initiates the game first gets to act as a Host and the other who is requesting to be a part of the game acts as a client, no Wi-Fi access points or Internet connectivity is required.

Some multiplayer games are use Wi-Fi or 3G connections to share the data globally and which helps in having multiplayer games. Developers can connect a large number of mobile games with a single server and share data among the players. The problem in this kind of connection is that it requires Client –Server communication using GSM. A client-server game consists of the individual game clients connected to a central server computer via the Internet. Game play is handled by having each user's game client communicate with the server. The server is responsible for passing information on to the other users. For instance, when one user moves, the user's computer sends a message to the server. The server then sends messages to the other players to inform them of a change in game state [32]. These technologies can provide faster data transform speed and don’t have the limitation of distance that appears in short-range communication technologies. Real-time

multiplayer gaming achieved by using Wi-Fi connection is usually faster than use 3G connection. The drawback of this technology is that 3G mobile phones are costly and the users of the mobile phone have to pay for the Wi-Fi connection. So this technology is expensive in terms of operation and communication.

2.6 iPhone

2.6.1 Introduction

The iPhone is a touchscreen smartphone brand designed and marketed by Apple. Unlike the traditional cell phone, the user interface of iPhone is built based on the device's multi-touch screen. The iPhone 4 is the most recent generation of iPhone with a high-resolution display, FaceTime video calling, HD video recording, a 5-megapixel camera, and build-in A-GPS, gyroscope and accelerometer sensors. In additional to be used as a communication tool, an iPhone 4 could function as a video camera, a media player, an Internet client or even a portable game console.

2.6.2 iOS

iOS is Apple's mobile operating system that runs on iPhone, iPod touch and iPad devices. This operating system manages the device's hardware and provides the frameworks to allow developer to implement their own applications.

The user interface of iOS is based on multi-touch gestures such as swipe, tap, pinch, and reverse pinch. All of these gestures are predefined in the iOS multi-touch interface. The build-in gyroscope and accelerometers are also used by some application to response the motion gestures of the device.

The iOS is consisted with four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer, which are shown in Figure 2.1. Each of the Core Services layer and the Cocoa Touch layer has an Objective-C framework that is

especially important for developing applications for iOS. They are the core Cocoa frameworks of iOS [33].

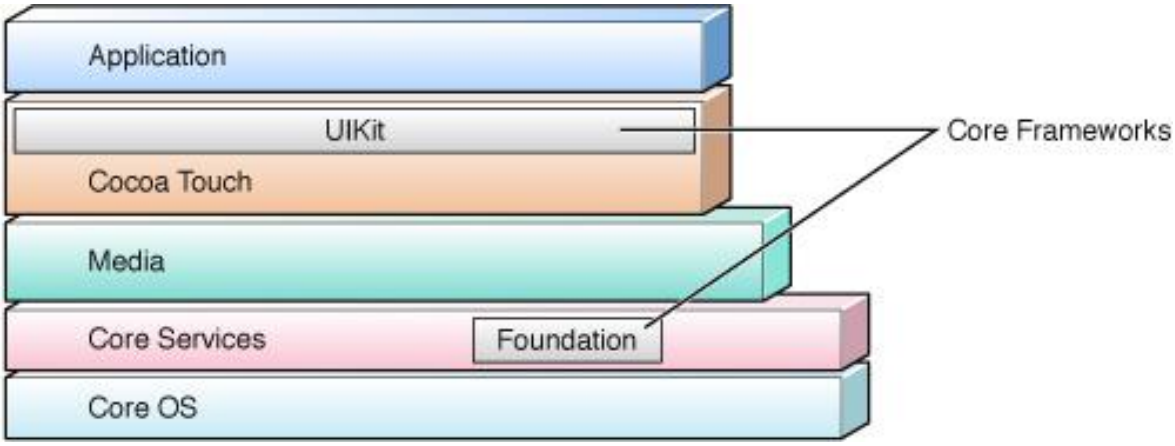


Figure 2.1 Cocoa in the architecture of iOS [33]

2.6.3 Developing for the iOS

Apple requires the use of iOS Software Development Kit (SDK) to develop applications for iPhone. The iOS SDK comes with all of the interfaces, tools, and resources needed to develop iOS application [34]. The key components of iOS SDK include XCode Tools that support iOS application development, iOS simulator that simulates the iOS technology stack and iOS developer Library that provides the reference and conceptual documentation to developers.

The programming language for iOS SDK is Object-C. The Object-C language is an object-oriented programming language that extended from the standard ANSI C language.

Chapter 3

Design & Implementation

In this chapter, we first describe the concept of the game in Section 3.1. The design and implementation details of the game framework are illustrate in Section 3.2 and Section 3.3.

3.1 Overview

The aim of this project is to investigate and implement a multiplayer mobile game that uses mobile phone camera and other build-in sensors (i.e. GPS, gyroscope) as game inputs. In order to achieve this goal, a multiplayer mobile game named Mobile Phone Paintball is developed in this project. To play this game, players need to first install the game application to their smartphones and then attempt to "kill" their enemies during the game. The game rule is almost the same as a real Paintball game. The difference is in the Mobile Phone Paintball game players use the build-in camera in their mobile phones as paint ball guns to aim and shoot their enemies. The video frames that are obtained from the camera together with the location and heading data that are collected from GPS and gyroscope sensors will be processed in real-time to detect whether an enemy is killed.

The framework we designed in this paper is consists of two main components:

- A game server running on a Mac that responsible for exchange data between players
- Game client application running on players' iPhone to detect enemies and decide whether an enemy is killed.

In order to make the game working, the following goals need to be achieved:

- Payers can communicate with each other by connecting to the game server.
- The game client can detect enemies by processing the video frames from the mobile phone camera.
- Once an enemy is detected through the camera, the game client knows whether this enemy can be kill based on the orientation and the heading of the enemy.

3.2 Game Server

3.2.1 Framework

Many iOS applications use Hypertext Transfer Protocol (HTTP) to communicate to a web server, because it is easy to implement and well-supported. However, HTTP is not efficient due to it requires new connections to be established at each time when a client wants to send data to the server. In addition, by using HTTP connection the server can only require the clients to pool rather than send clients data at any time. In this project, instead of using HTTP sockets, we choose to use Transmission Control Protocol (TCP) sockets. The advantages of doing this are listed as follows:

- By using TCP, we can send exact data by customize our own protocol.
- Once a TCP connection is created between a server and a client, it will only be closed when the clients want to stop the communication or when the server gets stooped.
- The server can send data to connected clients at any time after TCP connections is created.

The Game server in this project is implemented by using a networking engine named Twisted [35]. Twisted is an event-driven networking engine written in Python that support a large number of protocols (including HTTP, NNTP, IMAP, SSH, IRC, FTP, and others). It is designed based on the reactor pattern, which stars with a loop, waits for events and

then reacts to the events.

During the game, the game server is responsible for following tasks:

- Listen for incoming game clients connections
- Handle requests that are sent from game clients
- Send required data back to clients

As shown in Figure 3.1, Game clients use TCP to connect and communicate with the game server. All requests send by game clients will be processed by a request handler on the server side. In order to handle these requests, the server needs an event handler to define the format of data that are sent back and forth between the game server and game clients. The game server also needs a database to store and manage to players' data during the game.

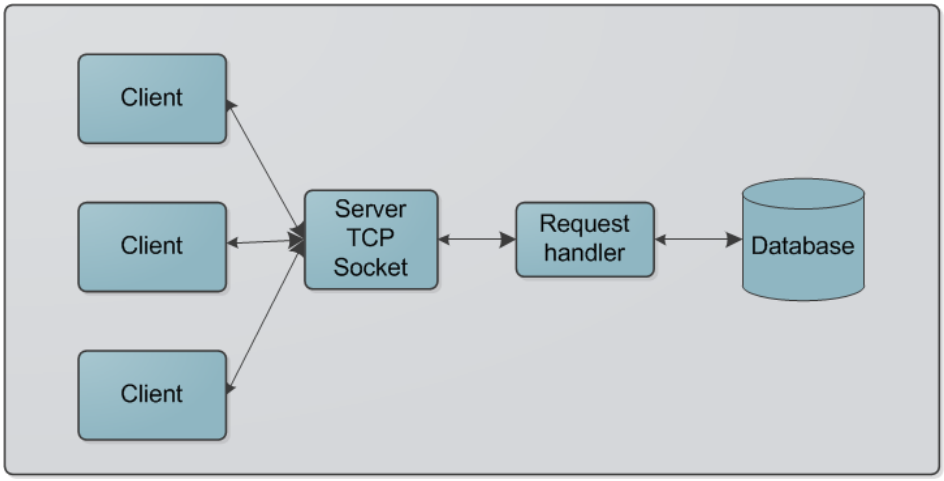


Figure 3.1 Structure of the game server

3.2.2 Database

A game database is designed to store and manage the players' data. During the game, this database will be updated based on the request send by the game clients. The structure of this database is described below:

- ID: every player will be given a unique ID when he or she login the game server

- Name: the name given by the player when he or she join the game
- Latitude: stores the latitude of current position of the player
- Longitude: stores the longitude of current position of the player
- Accuracy: stores the current accuracy of the GPS sensor
- Heading: stores the current orientation of the player
- Team: stores the team number the player is belong to
- Ammo: stores the number of left ammo of the player
- Alive: indicates whether a player is still alive or has been killed
- Ready: indicates whether a player is ready to start a new game

We choose to use SQLite to create and manage our game server database. SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine [36]. It allows accessing the database using a nonstandard variant of the SQL query language. Sqlite3 is a DB-API 2.0 interface for SQLite databases. Python comes with sqlite3 by default. In order to use the sqlite3 module in Python, we just need to import the sqlite3 library at the beginning of the program. To access the database, we must first create a *connection* object that represents the database and then create a *cursor* object and call its *execute*() method to perform SQL commands to manage the database [37].

3.2.3 Request Handler

A simple format of request is used to exchange data between the game server and game clients, which is a string that always included in a pair of square brackets. The string between the square brackets is always starts with a command and followed by a ":" and then contents. Contents usually contain multiple kinds of data that are separated by ";". The following example shows the format of the message:

[Command: data1; data2; data3; ……]

After received a request, the game server will deal with it based on what kind of command is in it and update its database according to the contents of request. The processes of

handling a request are described as follows:

- Split the string to find out the command and the contents
- Split the contents to find out the data
- Update the database based on what the command is and the data in the contents
- Generate a response message with the same format and broadcast it to the other clients

The request type that is used between the game server and game clients is described as follows:

- *[newPlayer: playerName]:* this request is sent to the game server from game clients when a new player want to join the game. The server will search the database to check if there is already a player with the same name existing in the database. If not, this new player name will be added into the database and the game server will broadcast this message to other players. Otherwise, the game server will send back an error message to tell the game client player name already exists.
- *[getAllPlayers]:* tell the game server to search the database to find all the players that are currently in the game. The game server will send back a message contains names of these players.
- *[ready: playerName; teamNumber]:* tell the game server the player is ready to start a new game. The game server will search the database, change the "ready" property of this player to true and set the "Team" property of this player to the teamNumber , then broadcast this information to other players.
- *[update: playerName; heading; latitude; longitude; gpsAcurracy]*: tell the game server to update the location information of the player. The game server will first search the database to find this player and update his or her location information and then broadcast this information to other players.
- *[playerDie: plaeyrName***]**: tell the game server the player has been killed. The game server will search the database to find this player, update his or her "alive" property to false and send this message to other players.

3.3 Game Client

The game client runs on players' iPhone. As shown in Figure 3.2, when launches the game clients, players need to login to the game through login interface. if login successful, players will enter the game lobby interface. The game will start after all the players chose their team and

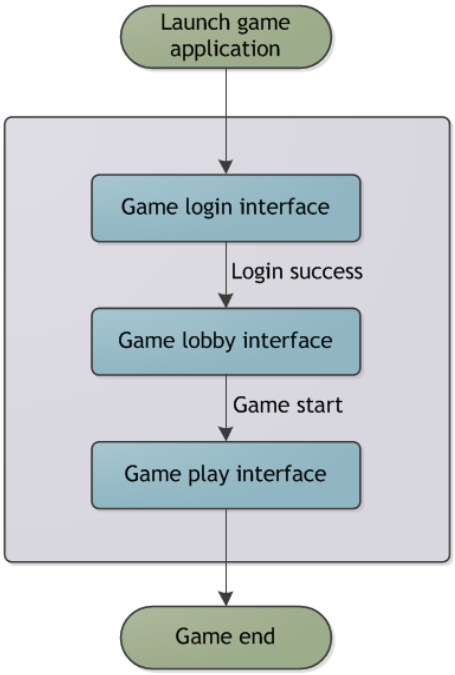


Figure 3.2 Three main interfaces in the game client

set their status to ready in game lobby. Once the game starts, clients will begin to obtain players' current GPS coordinates and orientation from the mobile phone's build-in sensors and send these data to the game server. At the same time, the game server will send the updated GPS and orientation data to all the players. Game clients also obtain video frames from the mobile camera and process them in real time. A player could only be killed by an enemy when he or she standing in front of this enemy (calculated by using GPS coordinates and orientation data) and also captured by the enemy's mobile phone camera. If an enemy is detected and killed, this information will send to the game server and at the same time, the game server will send the updated game data to other players.

There are three main interfaces in the game client: login interface, game lobby interface and game play interface. When the game client is launched, it will first make TCP connection to the game server. Then the player can join to the game through the game login interface, choose a team to join through the game lobby interface and play the game through game play interface.

3.3.1 Communicate with Game Server

We use streams to create a socket connection to the game server on iOS. There are three important classes related to steams that are included in the Foundation framework of iOS:

- *NSStream*: an abstract class for objects representing streams. Its interface is common to all Cocoa stream classes, including its concrete subclasses *NSInputStream* and *NSOutputStream*. It provides an easy way to read and write data to and from a variety of media in a device-independent way.
- *NSInputStream*: a subclass of *NSStream* for reading input stream.
- *NSOutputStream*: a subclass of *NSStream* for writing output stream.

The steps of initializing a network communication are described as follows:

- Step 1: Use *CFStreamCreatePairWithSocketToHost* method to create readable and writable streams connected to a given TCP/IP port of a particular host.
- Step 2: Create an *NSInputStream* object and an *NSOutputStream*. Sign the two *NSStream* object that created by *CFStreamCreatePairWithSocketToHost* method to them.
- Step 3: Set the delegate of the *NSOutputStream* object and *NSOutputStream* object to themselves to allow them receive notifications.
- Step 4: Sign both of these streams have to the run loop so that they can continuously send and receive data.
- Step 5: Open these two steams so they can start to read and receive data.

Once the network communication is established, we can send request to the game server by

write data into the *NSOutputStream* object. To message data from game server, we just read data from the *NSInputStream* object. But the message send from the game server is in byte format, we need to perform the following steps to convert the massage from byte to string:

- Step 1: Read bytes data from the input stream
- Step 2: Define a buffer and collect these bytes into it
- Step 3: Convert the buffer into a string

In order to deal with the messages that are received from the gam server, a request handler is also implemented in game client side. This request handler is almost the same as the request handler that are described in Section 3.22. The difference is the request handler in game server always broadcasts the updated data to other players once it done with a request, but the request handler in game client dose not sends back messages to the game server or broadcast the message to other players after it done with a request.

3.3.2 Database

A database is also designed and implemented in game client side to store and manage the players' data. The structure of this database is the same as the database in the game server that is described in Section 3.2.2. iPhone SKD also has built-in sqlite3 library as Python, so we use this library again to access the database in game client. The method we used to manage the database is described as follows:

- *(Boolean)createDatabase***:** this method will be called if the database did not exist. It will create an empty database.
- *(Boolean) updatePlayerData:(PlayerData *) data***:** search the database to find the particular player and update his or her data
- **(***Boolean) deletePlayer:(NSString *) playerName***:** search the database and delete the particular player
- *(Boolean) deleteAllPlayer***:** delete all players' data in the database
- *(Boolean) insertPlayer:(PlayerData *) data***:** insert a new player to the database

- *(NSArray *) readPlayer*: read all players' data from the database and store them into an NSArray
- *(Boolean) checkWinOrLoose*: search the database to find enemies that are still alive, if all enemies are dead, return true, otherwise return false
- *(Boolean) setPlayerReady:(NSString*) playerName withTeamNumber: (int) teamNumber*: search the database to find a particular player. Set the "ready" property of this player to true, and set the "team" property to the corresponding team number
- *(Boolean) closeDatabase*: close the connection to the database when exiting the game client

3.3.3 Game Interfaces

3.3.3.1 Game Login Interface

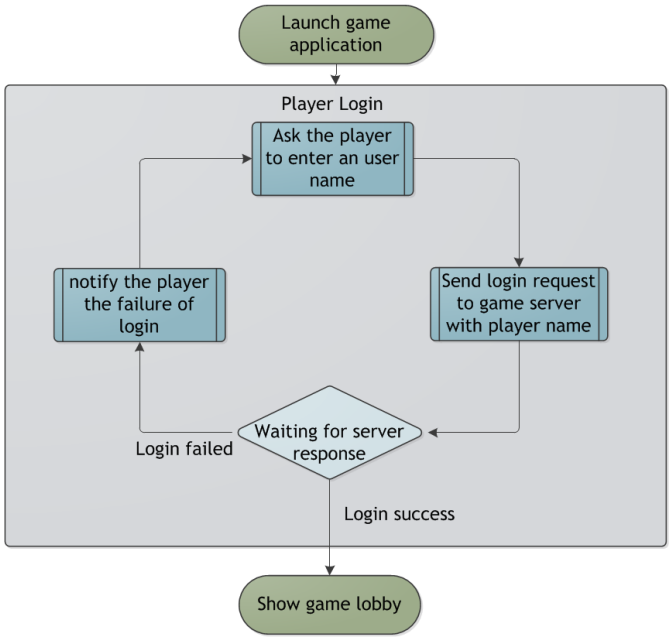


Figure 3.3 Processes of login to the game server

When players launch the game application, they need to first login to the game server through the game login interface. The login process is described as below (Figure 3.3):

- Step 1: require the player to enter a user name and press *Login* button.
- Step 2: send login request *[newplayer: playerName]* to game server.
- Step 3: if game server responds a login success message, show game lobby interface and stores the player's data into the database. Otherwise, ask the player to login again.

3.3.3.2 Game Lobby Interface

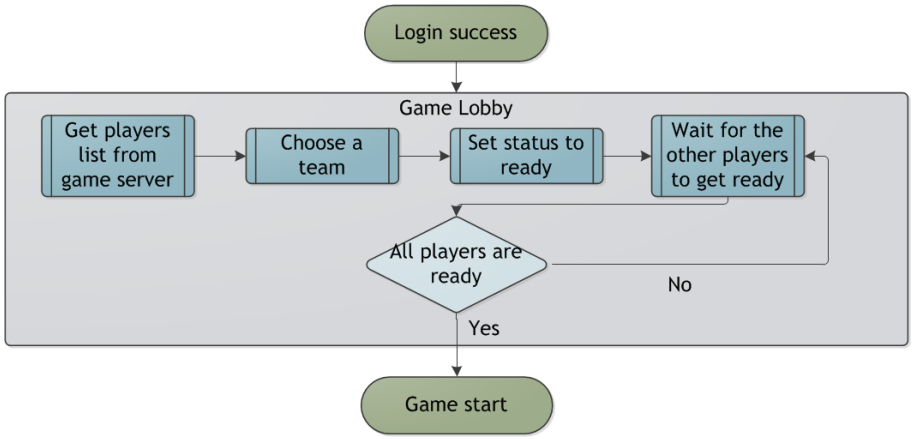


Figure 3.4 Processes of start a game

After logged into the game server, the game client will show the game lobby interface. The list of players that currently logged into the game server will be displayed in this interface. The players can check other players' status, choose a team they want to join and change their status to ready. To start a game, players need to perform the following steps (Figure 3.4):

- Step 1: Choose a team to join and then set their status to ready by press *Ready* button.
- Step 2: A set status to ready request *[ready: playerName; teamNumber]* will send to game server after the player pressed the "ready" button, the game server will update its database based on this request and then send the updated data to all the players.
- Step 3: After all players have chosen their teams and set their statuses to ready, press *Start* button to start the game.

3.3.3.3 Game Play Interface

The game player interface is consists of three layers: a camera preview layer, an AR layer and a heads-up display (HUD) layer. The details of these three layers are described as follows:

- Camera preview layer: it is the bottom layer of the game play interface. The video frames captured from the iPhone's camera are displayed in this layer in real-time. So that the player can see the video inputs of their iPhone's camera from the screen.
- AR layer: this layer is displayed at the top of the camera preview layer. Once the game starts, an enemy detection algorithm is performed on every input frame. The result of the enemy detection will be shown on the AR layer. A transparent gray mask will be displayed at the top at the enemies to notify the player enemies have been detected.
- HUD layer: the HUD layer is displayed over the AR layer. The included features in the HUD are:
    - Virtual radar: display all the nearby players at the relative position as dots with different color.
    - Ammo: show how much ammo is left.
    - Enemy number: show the number of live enemies.
    - GPS accuracy: show the current GPS accuracy.
    - Enemy detection: tell the player whether an enemy has been detected.

As shown in Figure 3.5, once the game starts, the game client will show the game play interface. The player's current GPS coordinates and heading will be collected from the build-in GPS sensor and send to the game server. Every time when the game server receives a new coordinates for a player, it will updates its database and send this new coordinates to the other connected game clients. So, the database in both of the game server and game clients stores the up-to-date coordinates and heading data of all players. These data will be used by the game client to detect the nearby players, which can be done by calculating the distance from current player to other players. The players that are within

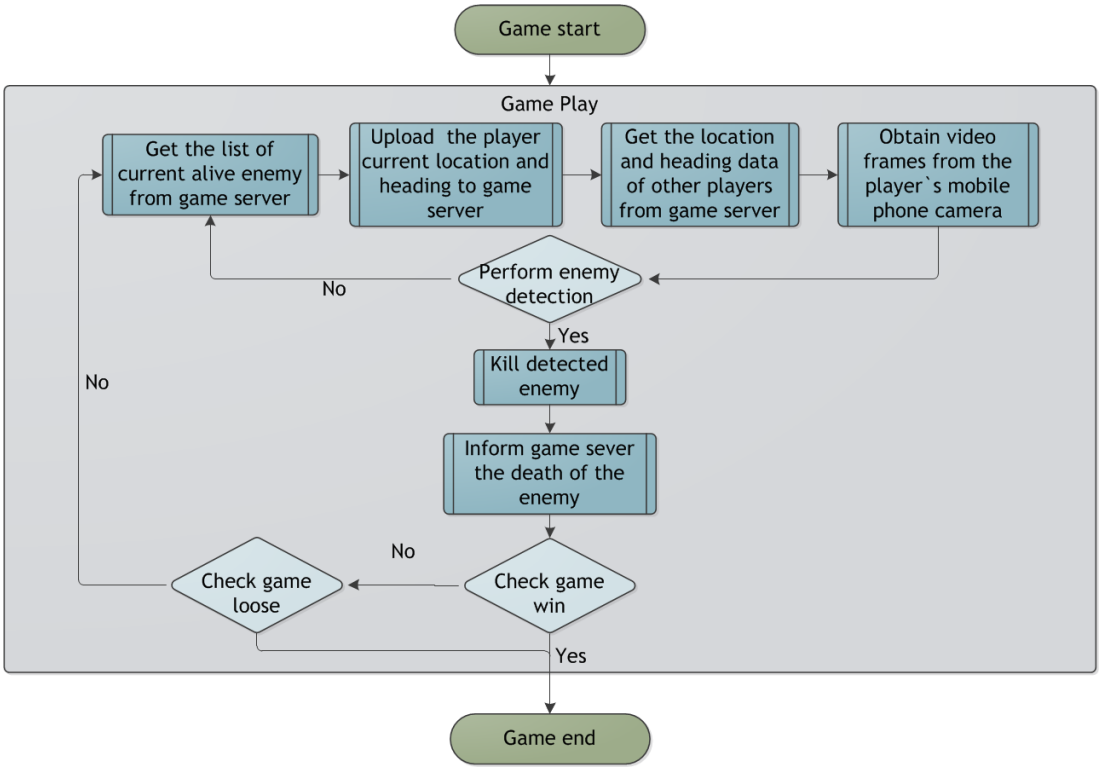


Figure 3.5 Processes of game playing

a certain distance to the current player will be marked as nearby players. The detected nearby players will then be displayed onto the virtual radar in the HUD.

The video frames obtained from the iPhone's camera will also be processed in real-time to perform enemy detection algorithm. If an enemy is both detected in the video frames and within a certain distance in front of the player, this enemy could be killed by the player.

3.3.4 Player Location Positioning

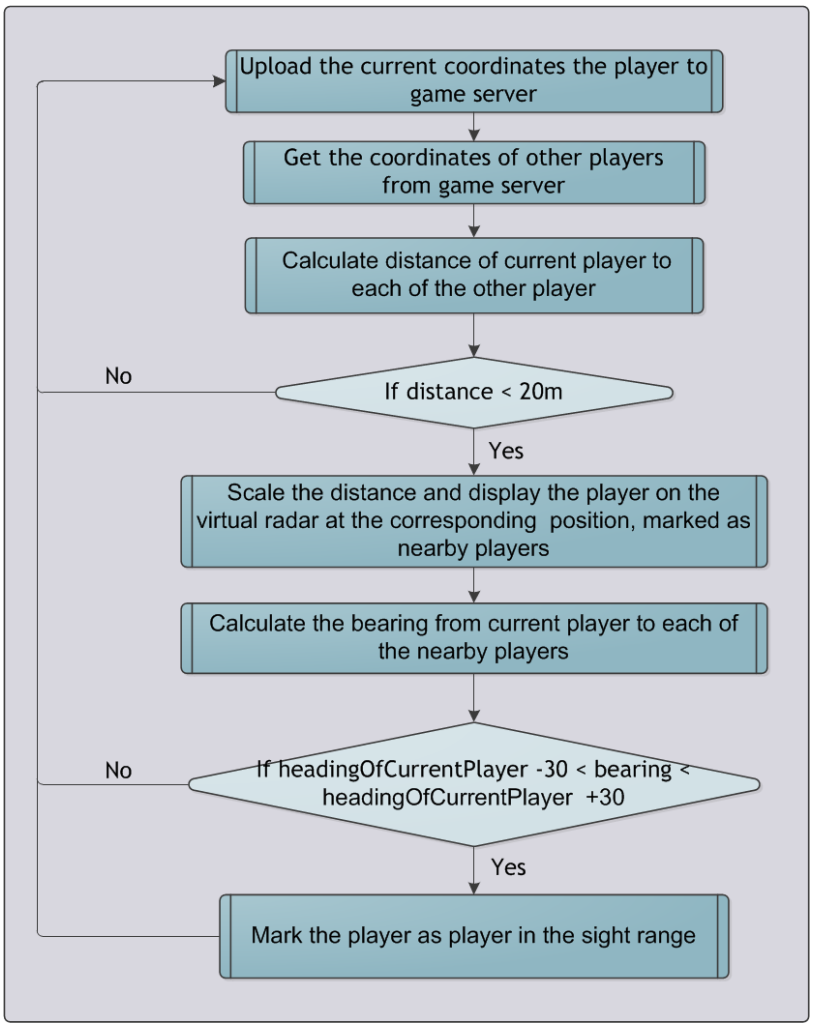


Figure 3.6 Processes of locating a player

In a real paintball game, if players want to kill one of their enemies, they must first get close enough to this enemy and then aim the enemy by using their gun. The processes to kill an enemy in Mobile Phone Paintball game is almost the same. In this game, the players must within a certain distance to the enemy and use their mobile phone camera as paintball gun to aim the enemy. In order to implement these functionality, the position and heading of each player must been known. IPhone 4 has built-in GPS and gyroscope sensors, so it can provides these data during the game if user enables the location service in the phone. Every time when a new current location data is given by the location server, it will be sent

to all the other players through the game server. Each of the game client stores a list of the nearby players and a list of the players are current in the sight range of the current player. The distance and bearing of the other players to current player will be calculated every time as the players' locations are updated. The players that are currently within a certain distance will be added to the nearby player list. And also if their bearing angles to current player are within a certain range, they will be added to the list of players that are current in the sight range of the current player. The processes of detecting an enemy are described in Figure 3.6.

3.3.4.1 Calculate Distance and Bearing

In order to retrieve location data from the iOS, we need to first import the Core Location framework into the project. The Core Location framework uses the available hardware to determine the user's position and heading. Develops can sue the classes and protocols in this framework to setup and manage the delivery of location and heading events. In this project we used three important classes that are included in the Core Location framework**:**

- *CLLocationManager***:** this class defines the interface for configuring the delivery of location and heading related events. We can create an instance of this class with the parameters to start and stop the delivery of the location and heading events and to determine when to deliver them. Once a *CLLocationManager* instance is created, we can retrieve the most recent location and heading data from it.
- *CLHeading***:** a *CLHeading* object contains the heading data generated by a *CLLocationManager* instance**.** The heading data contains both of the computed values for true and magnetic north.
- *CLLocation***:** a *CLLocation* object contains the location data generated by a *CLLocationManager* instance. The location data contains the geographical latitude and longitude coordinates and altitude of the device's location with a value indicates the accuracy of these coordinates.

The processes of setup a location manage and receive location and data from it are

described as follows:

- Step 1: create a *CLLocationManager* instance called *locationManager* and then to determine whether location services are enabled by check the *locationServicesEnabled* if this instance. If the location services are disabled, display an alert to tell the player turn on the location services in their iPhone.
- Step 2: setup the delegate of the *locationManager* so we can receive location and heading events from it.
- Step 3: decide the minimum distance and angular change required to generate new location and heading event by setup the *distanceFilter* and the *headingFilter* property of the *locationManager***.**
- Step 4: setup the device orientation to use when computing heading data by sign an orientation value to the *headingOrientation* property of the *locationManager***.**
- Step 5: start to generate the location and heading event by calling the *startUpdatingLocation* and the *startUpdatingHeading* method of the *locationManager***.**
- Step 6: retrieve the most recent location and heading data by calling the *didUpdateHeading* and *didUpdateToLocation* method of the *locationManager***.**
- Step 7: every time when a new location or heading data is retrieved from the *locationManager***,** a request message *[update: playerName; heading; latitude; longitude; gpsAcurracy]* will be sent to the game server to tell it update the location and heading data for the player. The game server will broadcast this message to other players.

After known the location and heading data for current player and received these data of other players from the game server, we can use these data to calculate the distance and bearing from current player to other players. The *CLLocationManager* provides a method called *distanceFromLocation* to calculate and return the distance between two given location coordinates. But the *CLLocationManager* does not provide method to calculate the bearing angle of two coordinates. So we need to calculate the bearing by ourselves. The

formula of calculate bearing from two location coordinates is shown as below:

$$\theta = atan2(\ sin(\Delta long).cos(lat_2),\ cos(lat_1).sin(lat_2) - sin(lat_1).cos(lat_2).cos(\Delta long)\ )$$

In the above formula lat_1 and lat_2 is the latitude of two given locations and the $\Delta long$ is the differences between the longitude of the two given locations, The location and heading data we get from the *CLLocationManager* is in degree format, we need to first convert this data from degree format into radians format and then calculate the bearing by using the above formula. Since atan2 returns values in the range $-\pi$ to$+\pi$, but the range of the compass bearing is from 0 to2π. We need to normalize the result by add 2π to the result if it is small than 0.

3.3.4.2 Virtual Radar

The position and heading data given by iPhone is in latitude, longitude and degree format. In order to make these data easy to understand, an virtual radar is designed to be displayed on the screen of the players' mobile phones to illustrate the position of their nearby teammates or enemies (Figure 3.7).

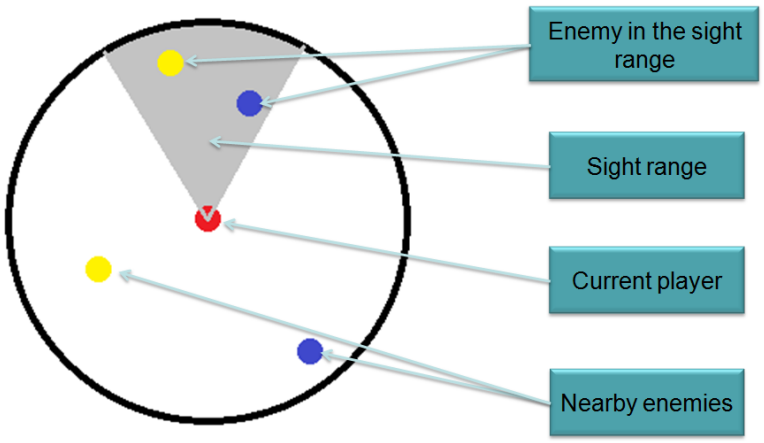


Figure 3.7 Design of the virtual radar

During the game, the center of the radar always indicates the current position of the player. All the nearby teammates and enemies will be displayed in a relative position within the virtual radar circle as dots with different colors. The gray fan-shape on the virtual radar

indicates the current heading and sight range of the player. Only the enemies that are in the player's sight range could be detected and killed by the player.

3.3.5 Detect Enemy from Camera Input

3.3.5.1 Enemy Detection by Using Hats

Only use location and heading information to kill an enemy is unreliable. For example, if an enemy is hiding behind an obstacle and a player is close enough and aim his or her phone to the direction of the enemy, this enemy would still be killed. The method we are using to solve this problem is take account of the vision. An enemy could only be killed be if he or she is not only in the sight range of the virtual radar, but also be captured by the camera of the iPhone.

Compared to human recognition and face recognition, object recognition by using color, shape and features is much suitable for mobile platform, as it requires less computation and usually with higher recognition rate. The solution for enemy detection in this game is require players to wear different kinds of specially designed hats depends on which team they are in. These hats are painted with two colors that are rare be seen in our daily life (Figure 3.8).

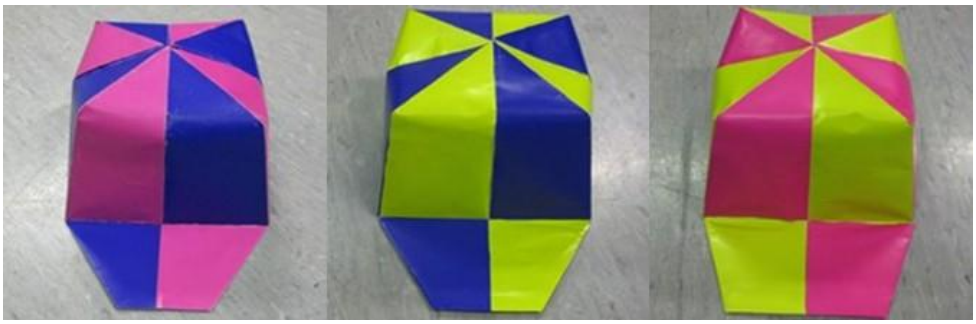


Figure 3.8 Three kinds of hats used in Mobile Phone Paintball

By using this method, the enemy detection process could be simplified to color detection. One efficient way to detect specific colors in an image is convert the image into HSV color space, so that it is less susceptible to different lighting condition of a similar color. The hat detection process can be divided into following steps:

- Obtain video frames from camera
- Convert these video frames into HSV color space
- Define thresholds values for hue and saturation based on the two colors on the hat.
- Try to find the hat in these frames by check the hue and saturation value of each pixel with these thresholds.
- If a hat is detected, tell the player an enemy is detected.

3.3.5.2 Using OpenCV to Process Images

In order to simplify the processes of hat detection, we choose to use a third party library called OpenCV. OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision [1]. The library has a lot of optimized algorithms for image processing. OpenCV is written by C++. Object-C is a super set of C, so it is compatible with Object-C. One problem is the image type that is supported by OpenCV is *IplImage*, but it is not supported by iOS. In order to solve this problem, we implemented a class called *ImageUtilities* to convert *IplImage* to the image type that is supported by iOS as well as convert iOS image into *IplImage* to perform OpenCV algorithm. The methods in class *ImageUtilities* are described as follows:

- *createIplImageFromUIImage*: convert an image from *UIImage* format that is supported by iOS to *IplImage* format.
- *createUIImageFromIplImage*: convert an image from *IplImage* format to *UIImage* format.
- *createUIImageFromRawData*: create an *UIImage* object from the raw data that is obtained from camera inputs.

3.3.5.3 Access Video Inputs from iPhone ' Camera

Now we have an image-processing library and an *ImageUtilities* to support its functionality. The next step is to capture video frames from the camera input. To access the camera video frames, we need to add the following three important frameworks into our project:

- AVFunndation framework**:** it provides an interface for managing and playing audio and visual media in the iOS application. It also contains an interface for media capture and access to camera.
- CoreVideo framework**:** it provides an interface to allow developers to access and manipulate individual video frames.
- CoreMedia framework**:** is provides a low-level C interface for managing and playing audio and visual media in the iOS application.

After import these three frameworks into the project, we can now start to capture video frames from iPhone's camera. The processes of starting camera capture are described as follows:

- Step 1: create an *AVCaptureSession* object to coordinate the flow of data from AV input devices to output.
- Step 2: create an *AVCaptureVideoPreviewLayer* object and add it into camera preview layer (described in section 3.3.5), so the players can see the video frames that are captured by the iPhone's camera from its screen.
- Step 3: get the default camera device of the iPhone by creating an *AVCaptureDevice* object and set its default device to camera. An *AVCaptureDevice* object represents a physical capture device that can provides input data.
- Step 4: create an *AVCaptureInput* object to use the camera. It the object is equal to NULL, tell the user cannot crate camera capture.
- Step 5: create an *AVCaptureVideoDataOutput* object to allow us to process uncompressed or compressed frames that are captured by the camera.
- Step 6: Create a dispatch queue to run the camera capture on.
- Step 7: Set the sample buffer delegate of the *AVCaptureVideoDataOutput* object that we created in step 5 to self, so we can get the video input from it. And then configure its compression settings by assign corresponding values to its *videoSettings* property.
- Step 8: set the quality level of the frames we want by assign corresponding values

to the *AVCaptureSession* object we created in step 1.

- Step 9: Set the input of the session as the *AVCaptureInput* object we created in step 4**.** Set the output of the session as the *AVCaptureOutput* object we created in step 7.
- Step 10: the final step is to call *startRunning* method of the session object to start capture the video inputs from the iPhone's camera.

3.3.5.4 Hat Detection Algorithm

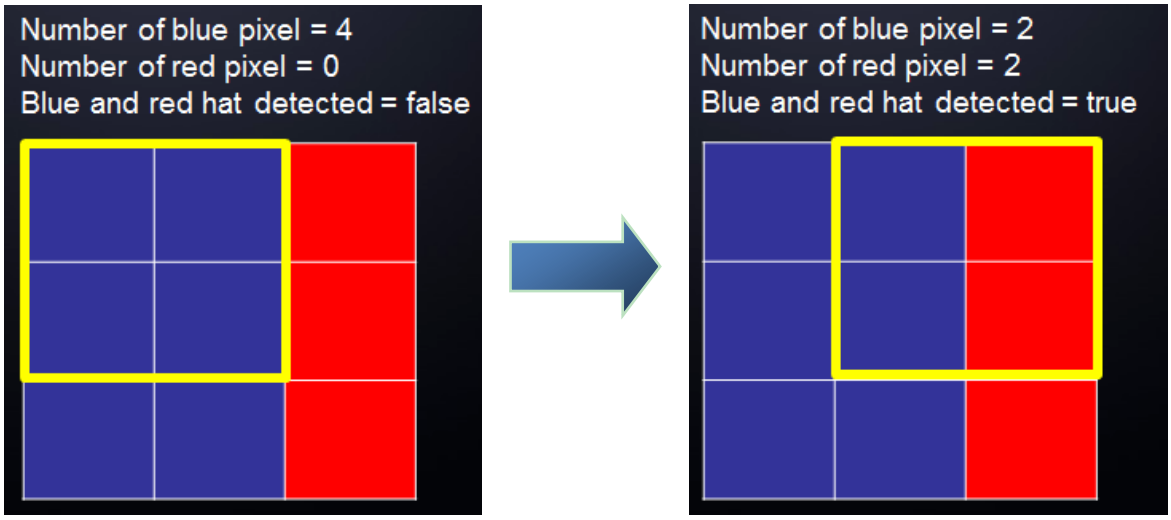


Figure 3.9 Example of detecting a red and blue hat

Once start to capture camera inputs, we can access the captured video frames by calling the *didOutputSampleBuffer* method. The image frames we get from this method are in raw data format. We need to convert the image raw data into *UIImage* by calling the *createUIImageFromRawData* method in the *ImageUtility* class. And then we need to convert the image from *UIImage* format to *IplImage* format. The next step is converting the image from RGB color space into HSV color space by calling the OpenCV method *cvCvtColor***.** After all of these steps have been done, we can perform the hat detection algorithm on the HSV image. From Figure 3.8 we can see there are three kinds color that are used to make the hat: red, blue and yellow. So the main idea of the hat detection algorithm is to detect these three colors in the image. We first define the hue and saturation

Algorithm 1 Hat Detection

```
Define a 2×2 mask
Define a square area for the input image to perform hat detection algorithm
Define hue and saturation threshold value for red, blue and yellow color
for each input HSV image
    mask.position = squreArea.startPosition
    redAndBlueHatDetected = false
    blueAndYellowHatDetected = false
    yellowAndBlueHatDetected = false
    while mask.position is within squareArea.boundary
        redPixelCount, bluePixelCount and yellowPixelCount equal to 0
        for each pixel in the mask
            Apply pre-defined red, blue and yellow color threshold
            if current pixel is red
                redPixelCoutn++
            end if
            if current pixel is blue
                bluePixelCount++
            end if
            if current pixel is yellow
                yellowPixelCount++
            end if
            If redPixelCount > 0 && bluePixelCount > 0
                redAndBlueHatDetected = true
            end if
            if bluePixelCount > 0 && yellowPixelCount > 0
                blueAndYellowHatDetected = true
            if yellowPixelCount > 0 && bluePixelCount > 0
                yellowAndBlueHatDetected = true
            end if
        end for
    end while
end for
```

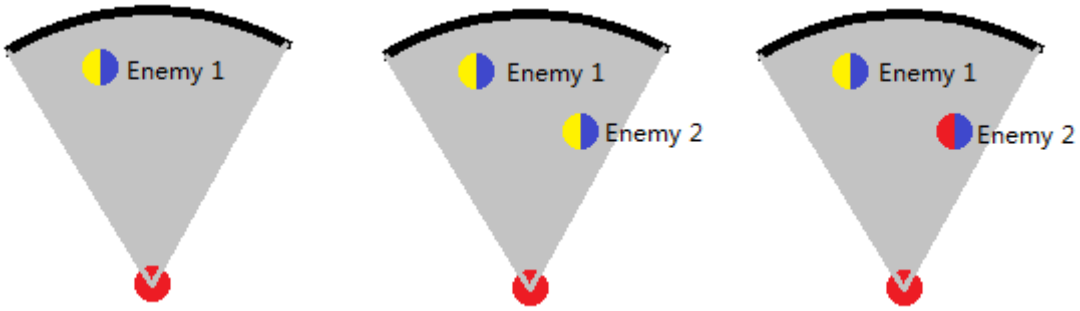


Figure 3.10 Decide which enemy will be killed

threshold for each of these three colors and then a 2X2 mask is defined. We move this mask within the input HSV image. For each new position of the mask, we count the number of red, blue and yellow pixels within it by using the pre-defined threshold. By analyzing the number of red, blue and yellow pixels in the mask, we can know whether a hat is detected and what kind of hat is detected. An example of detecting a red and blue hat using this algorithm is given in Figure 3.9. The detection algorithm is described in Algorithm 1.

3.3.5.5 Choose Threshold for Hat Detection Algorithm

We use three kinds of color to make the hats. In order to choose suitable hue and saturation values for detect the colors on the hats, we put these hats under different nature lighting condition and record the value of the hue and saturation values for these colors. We took 30 samples for each of the three colors and put the data into three charts to see the distribution of their hue and saturation value, these three charts are given in Appendix A. The hue and saturation thresholds for these colors are then decided depend on their hue and saturation value distributions.

3.3.5.6 Shooting an Enemy

An enemy can only be killed by a player if the specific kind of hat he or she wears is detected through the player's camera and this enemy is also in the sight range of the player. If there is more than one player in the player's sight range that are belong to the same team,

the enemy that is closest to the player will be killed. As the examples in Figure 3.11, if a blue and yellow hat is detected from the player's camera, *Enemy 1* will be killed in the left image, *Enemy 2* will be killed in the middle image and *Enemy 1* will be killed in the right image. If a red and blue hat is detected, the player can't killed the enemies in the left and the middle image as the type of hat detected is not match the enemy's team type in the player's sight range. But in the right image, *Enemy 2* can be killed by the player in this situation.

Chapter 4

Results

We will now present the results of our implementation. We first show the interfaces of the Mobile Phone Paintball in section 4.1. The result of hat detection and shooting detection are shown in section 4.1 and section 4.2.

4.1 Interfaces

We developed three main interfaces for the game: game login interface, game lobby interface and the game play interface. Once Players launch the game clients on that are installed in their iPhone, they need to first login to the game server in the game login interface. In the game lobby interface they can check the statuses of other players and choose a team they want to join and then they can start to play the game with the game play interface.

Figure 4.1 shows the screenshot of the game login interface. In the middle of this interface is a text input box. It is used for players to enter their player name they want to use in the game. After the players entered their name, they can login to the game server by press the **Join** button on the bottom of the interface. The game clients will then send a login request to the game server, if the game server sends back a login success message, plays will enter

the game lobby interface. Otherwise, the game clients will ask the players to login again.

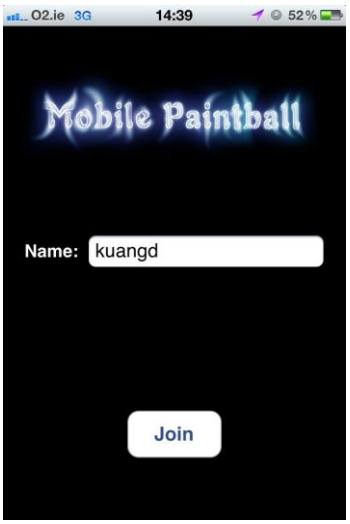


Figure 4.1 Game login interface

Figure 4.2 shows the screenshots of the game lobby interface. The statuses of other players that are currently on the game server are shown on the top the screen. In order to start a game, players need to choose a team they want to join and then press the **Ready** button to set their statues to ready. Every time when a player's status is changed, the information will be sent to the game server and then be updated to other players. The game can be started when all players set their statuses to ready.

Figure 4.2 Game lobby interface

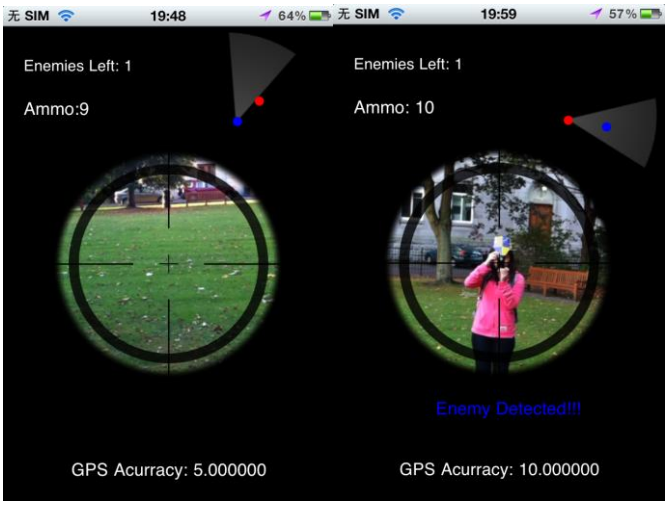


Figure 4.3 Game play interface

Once the game starts, the players can play the game through game play interface (Figure 4.3). The video inputs from camera will be displayed onto the circle area in the middle of the interface. A HUD is displayed over the interface to inform the players of their current statuses. A virtual radar is displayed in the upper right corner to show the nearby enemies. The gray triangle shape on the virtual radar indicates the current heading and the sight range of the current player.

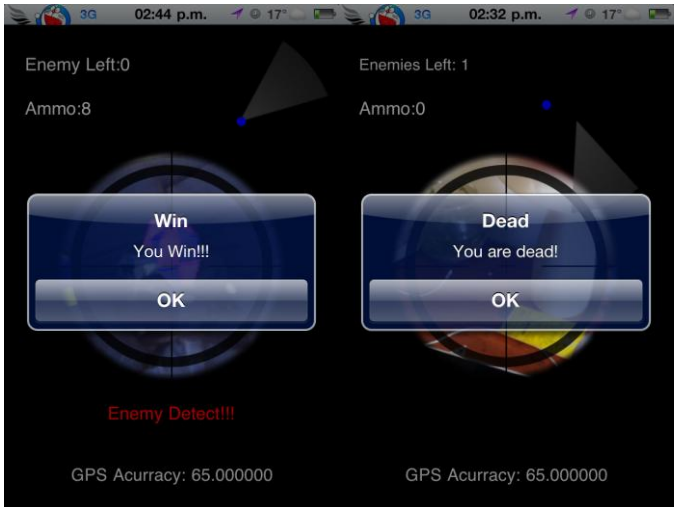


Figure 4.4 Game win and game lose screenshots

A player will win the game when all enemies are killed or lose the game when get killed by an enemy (Figure 4.4).

4.2 Hat Detection

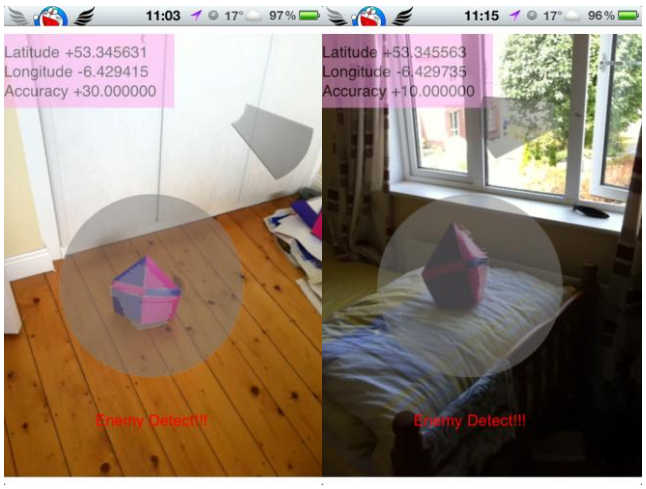


Figure 4.5 Detect hat under different natural lighting conditions

We first convert the video frames that are captured by the iPhone's camera from RGB color space into HSV color space and then use the pre-defined hue and saturation thresholds to detect the colors in the images. Figure 4.5 shows the hat detection result under different natural lighting conditions.

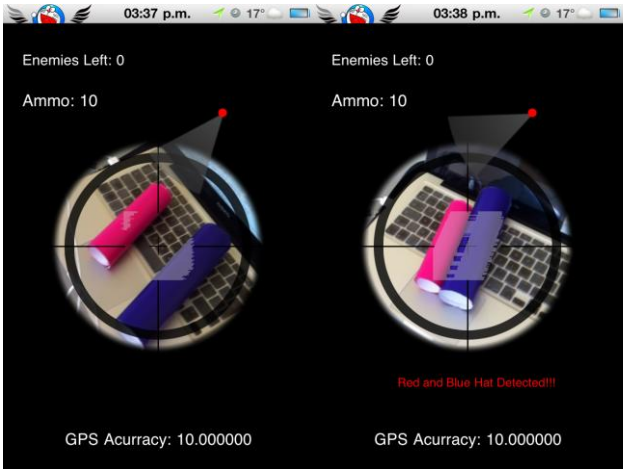


Figure 4.6 Special situation of detecting hat

Our hat detection algorithm actually detects the edge of the two colors on the hat. So even if there are two colors that are same with the colors on the hat, but they are not connected to each other, they will not be detected as a hat (Figure 4.6).

4.3 Detect Shooting

Figure 4.7 Detect shooting

An enemy can only be detected by a player only if he or she is in the sight range of the player and also the hat type corresponding to his or her team type is detected through the video inputs (Figure 4.7).

Figure 4.8 Hat type does not match the team type

A player can't kill an enemy if the hat type detected from the video inputs is not match the team type of the enemy (Figure 4.8).

Chapter 5

Evaluation

This chapter examines the performance of the Mobile Phone Paintball. In section 5.1 we evaluated the hat detection algorithm by trying to detect the hat from different distance and under different lighting conditions. Section 5.2 analyses the results that are gestured during player positioning tests.

5.1 Hat Detection

we performed three tests that are related to hat detection to evaluate the performance of our hat detection algorithm: detect the hat in different backgrounds with same distance, detect the hat with different distance in same background and detect the hat with different lighting condition.

5.1.1 Hat Detection in Different Backgrounds with Same Distance

In this section we use sensitivity and specificity to measure the performance of our heat detection algorithm. Sensitivity and specificity are measurements that are used to analysis the performance of a binary classification test. Sensitivity measures the proportion of actual positives which are identified correctly in a test. Specificity measures the proportion

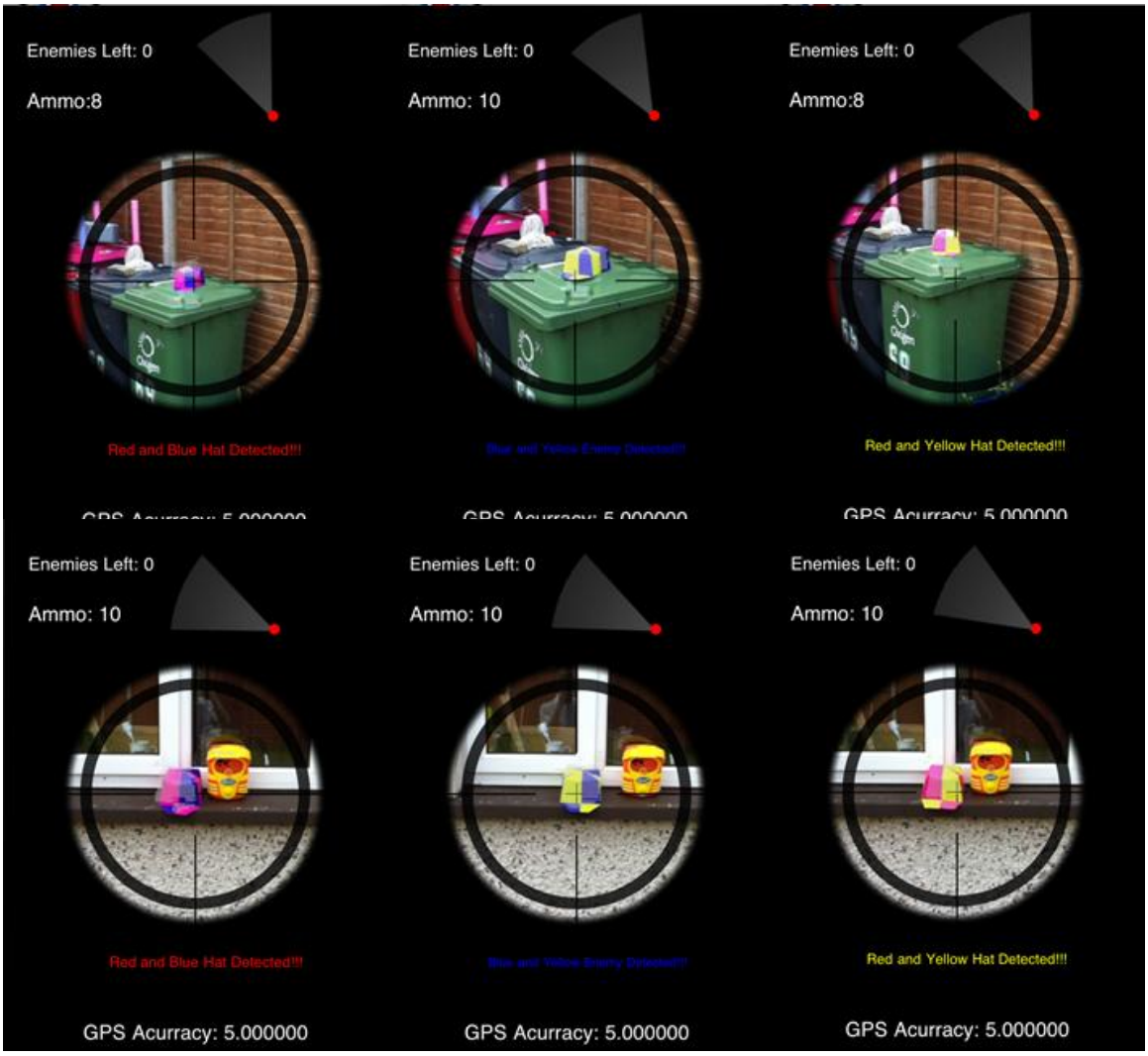


Figure 5.1 Hat detection result in different background with same distance

of actual negative which are identified correctly in a test. In this test, sensitivity means the percentage of a hat is correctly detected in an image with a hat and the specificity means the percentage of a hat is not detected in an image without a hat.

In order to calculate sensitivity and specificity, we first need to get familiar with four notions: true positive, false positive, true negative and false negative. The meanings of these four notions in this test are described as follows:

- True positive: a hat is detected in an image with a hat

- False positive: a hat is detected in an image without a hat
- True negative: a hat is not detected in an image without a hat or the wrong type of hat is detected in the image.
- False negative: a hat is not detected in an image with a hat.

The formulas that are used to calculate sensitivity and specificity are listed as follows:

$$sensitivity = \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives}$$

$$specificity = \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives + number\ of\ false\ positives}$$

We placed the hat in front of the different background and try to detect it from 2 meters away. And then we take away the hat to perform the same tests. We performed 30 tests with hat and 30 tests without hat (Figure 5.1).

In 30 times of tests with hat, the number of true positive results is 28 and the number of false negative results is 2. We can calculate the sensitivity as follows:

$$sensitivity = \frac{28}{28 + 2} = 93.33\%$$

In 30 times of tests without hat, the number of true negative results is 27 and the number of false positive results is 3. Then we can calculate the specificity as follows:

$$specificity = \frac{27}{27 + 3} = 90\%$$

From the result we find that the hat detection algorithm has good performance with different kinds of background. But we also find a shortcut of this algorithm is that it only detect a hat by its color without consider its shape or other features. This may lead to incorrect detection result when we trying to detect a hat that is in front of a colorful background. For example: a background has a big yellow area and also a big blue area, if we place a red and blue hat in front of it and try to detect the hat, the result of the hat detection algorithm may probably tell us a yellow and blue hat is detected. The way to

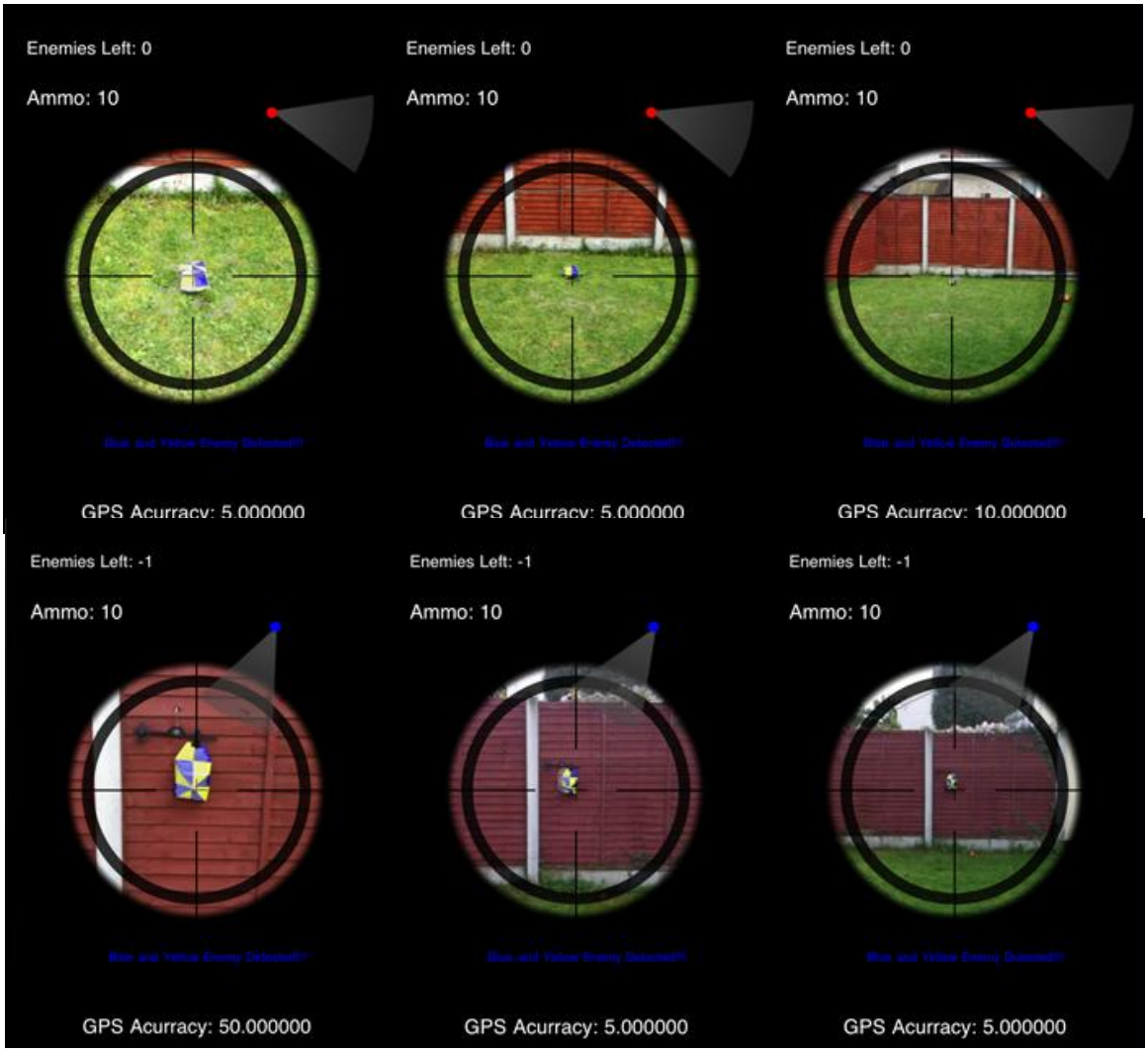


Figure 5.2 Hat detection result in same background with different kinds of distance (2 meters, 5 meters and 10 meters)

solve this problem is take account of both color and shape features of the hat in the hat detection algorithm.

5.1.2 Hat Detection with Different Distance in the Same Background

In this test we placed the hat in a same background and trying to detect it from 2 meters away, 5 meters away and 10 meters away (Figure 5.2). In 30 times of tests:

- Hat detected successfully in all three distance: 23 times (successful rate: 76.66%)

- Hat detected successfully in 2 meters and 5 meters: 27 times (successful rate: 90%)
- Hat detected successfully in 2 meters: 28 times (successful rate: 93.33%)

From these results we find that the successful rate of hat detection is decreases as the increase of the distance. The reason for this is as the distance between iPhone's camera and the hat is increasing, the hat will become smaller in the frames that are captured by the camera. This may lead the wrong result of the hat detection algorithm because it can't find the corresponding color of the hat from the input video frames. This problem is mainly due to the quality of the camera. If the camera could capture photos with higher resolutions, the hat could be detected in a longer distance.

5.1.3 Hat Detection under Different Lighting Conditions

In this test we attempt to find out how the natural lighting and artificial lighting can affect the result of hat detection algorithm.

We placed the hat under different kinds of natural lighting conditions and try to detect it from 1 meter away (Figure 5.3). In 30 times of tests, the hat is detected successfully in 25 times. The successful rate is 83.33%. From these test we find the natural lighting actually affects the results of the hat detection algorithm. The hat detection algorithm failed mostly either when the natural lighting is too weak or too strong. This is because if the lighting is too weak or too strong, the hat image that is captured by the camera will become either too dark or too light so that the hat detection algorithm can't recognize the colors on the hat.

We also placed the hat under different kinds of artificial lighting conditions and try to detect the hat from 1 meter away. In 30 times of tests, the hat is detected successfully in 11 times and failed in 19 times. The successful rate is 36.67%. The result shows that the artificial lighting affects the result of the hat algorithm a lot. The reason for this is that the artificial lightings usually have their own color. These kinds of lightings can actually make the colors on the hat look different.

One way to reduce the effects of different lighting conditions is to use optimal hue and

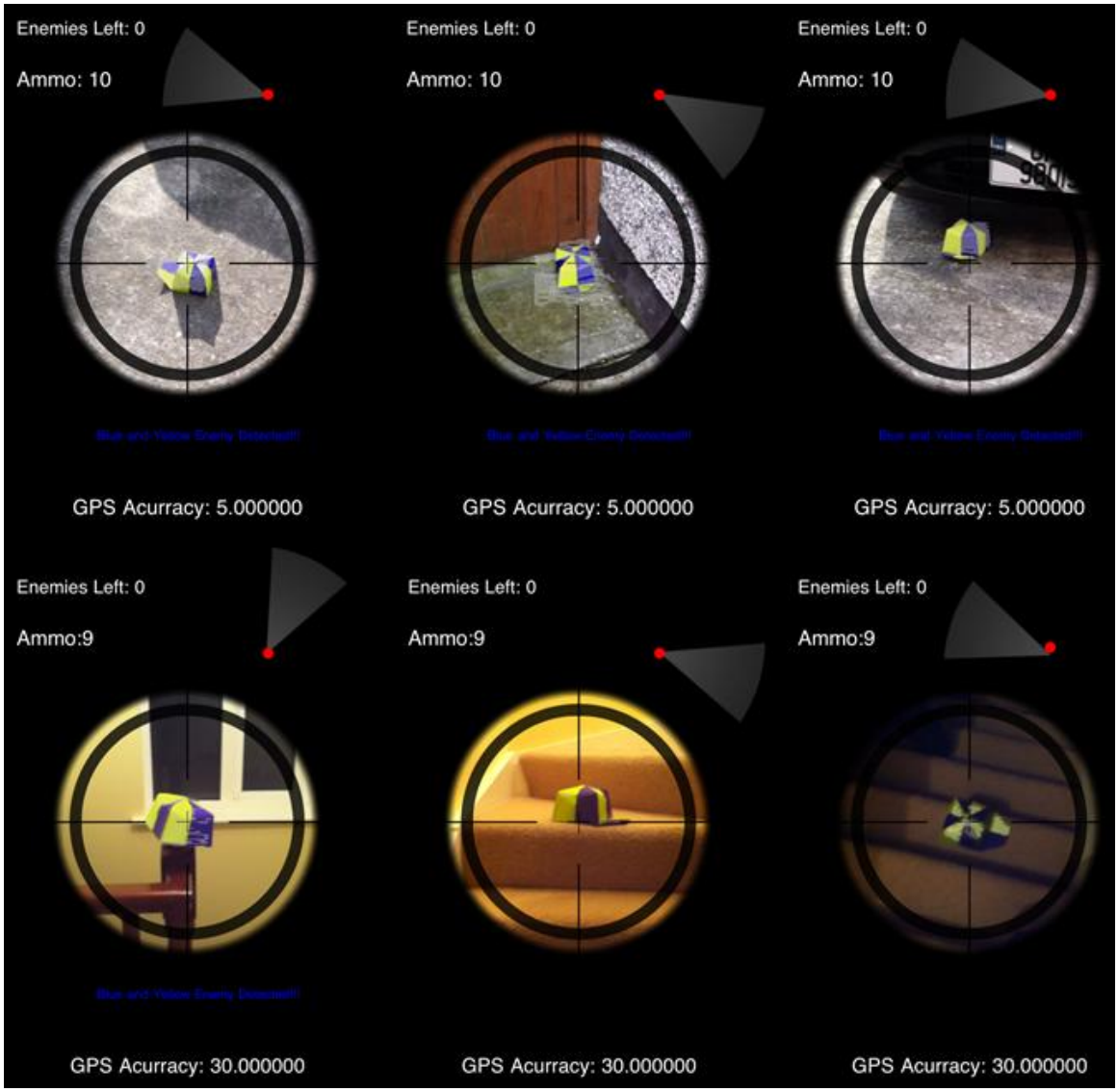


Figure 5.3 Hat detection under different natural lighting and artificial lighting conditions

saturation thresholds instead of using fixed threshold for different lighting conditions.

5.2 Player Positioning

In this section we assessed how the accuracy can affect the performance of player positioning by running the game client application on two iPhones at the same time. We place one iPhone on the ground and attempted to locate it by using the second iPhone from

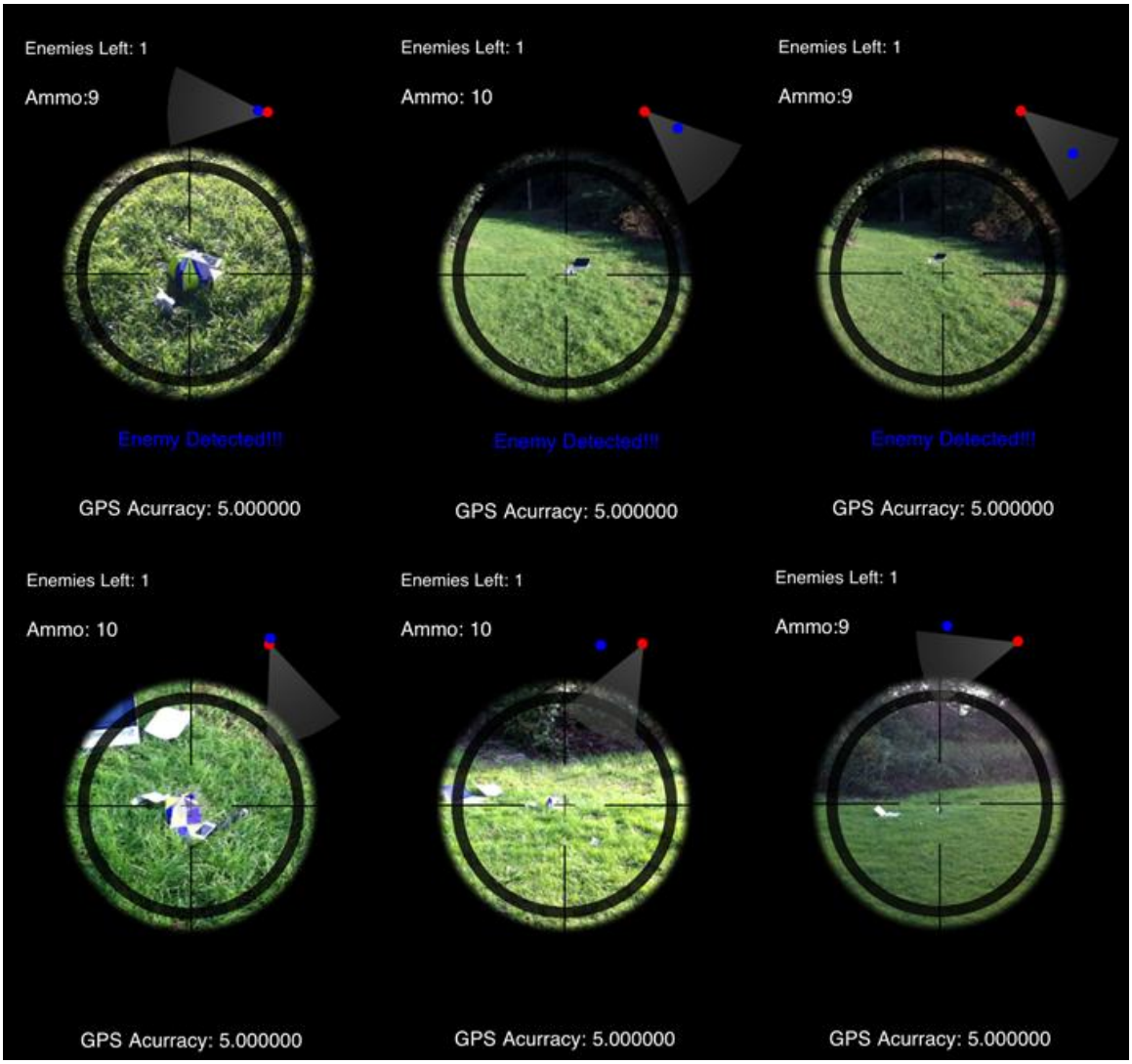


Figure 5.4 Player positioning from 2 meters, 5 meters and 10 meters away

2 meters, 5 meter and 10 meters. Also, for each distance we 3 performed the test from 3 different directions.

We performed this test in 10 different locations (Figure 5.4). The results of these tests are listed as follows:

- Player positioning from 2 meters away in 30 times of tests: success in 5 times (successful rate: 16.67%)
- Player positioning from 5 meters away in 30 times of tests: success in 10 times

(successful rate: 33.33%)

- Player positioning from 10 meters away in 30 times of tests: success in 17 times (successful rate: 56.67%)

From the result we can know the performance of player positioning is not good enough. This problem is mainly due to accuracy of the location coordinates we get from the GPS of the iPhone. In Section 5.1.2 we found out that the maximum distance for the hat detection algorithm to detect a hat successfully is around 10 meters. From the screenshots in Figure 5.4 we can see the GPS accuracy stayed in 5 meters at the most of the time. This is the highest accuracy we can get from iPhone's GPS sensor. This accuracy is good enough for navigation use, but the players in this game need to within 10 meters away from an enemy to allow the hat detection algorithm detect the enemy's hat. For 10 meters distance, 5 meters deviation is a serious problem as it can lead to positioning errors. But this GPS accuracy problem is actually a hardware problem that we can't do much about it from the software side.

Chapter 6

Conclusions

In this dissertation we proposed a new multiplayer gaming. The goal of this dissertation is to develop a multiplayer mobile phone game named Mobile Phone Paintball that uses camera and other built-in sensors (i.e. GPS and gyroscope) as game input. The game supports up to three teams to compete with each other at the same time. When players are playing the game, they use the cameras on their mobile phone as paintball guns to aim and shoot their enemies. The location coordinates and heading of the enemies are also collected to help players find them.

The framework of Mobile Phone Paintball is consists of two components: a game server and a game client application. The game server is written by Python using a networking engine called Twisted. It can connect and communicate with game clients through its TCP socket. A request handler is implemented to process the requests that are sent from game clients. A game database is also created by the game server to manage the players' data during the game.

The game client is an iPhone application. In order to play the game, players first need to install the game client application to their iPhone. When players launch the game application, they will be required to give a player name to login to the game server. A game lobby interface is design and implemented to allow players choose a team they want to join and check other players' statuses. When the game starts, the current location and heading

of each player will be collected from the iPhone's GPS and gyroscope sensors and then be uploaded to the game server. Every time when the game server receives a new location and heading data from a game client, it will send this data to all of the other game clients that are connect to it. Then, the distance and bearing from a player to other players can be calculated to help him or she gets close to the enemies. A triangle area in front of a player is defined as the sight range. Players can only shoot the enemies that are within their sight range.

When the game starts, the video inputs from the iPhone's camera will be displayed onto the screen. Every input frame will be processed to detect whether an enemy is captured by the camera. Due to the computation ability and memory issue of the iPhone, it is not suitably to use human tracking or face recognition technology to detect the enemies in this game. We solved this problem by designing three kinds of special hats, each kind of these hats has two colors on it which are rarely can be seen in our daily life. Every player needs to wear a specific kind of hat depends on what team they are in. A hat detection algorithm is implanted to detect the colors on these three kinds of hats from the video frames. The result of the hat detection and the distance and bearing information calculate from the location and heading data are then combined together to decide whether a enemy is killed after the player fired a shot. An enemy can only be killed by a player if he or she is both detected from the video input and within the sight range of this player.

From the results of evaluation we can find some drawbacks of the current hat detection algorithm as it uses fixed threshold value to detect colors under different lighting conditions and doesn't take account of the other features of the hat (i.e. the shape of the hat). A further study could exploit the way of using optimal threshold combines with shape features to detect the hat. Or even exploit the possibility of using human tracking or face recognition technology for enemy detection. Another problem we found is the accuracy of the location coordinates affects the game performance a lot. But this is mainly a hardware problem that we cannot solve from the software side.

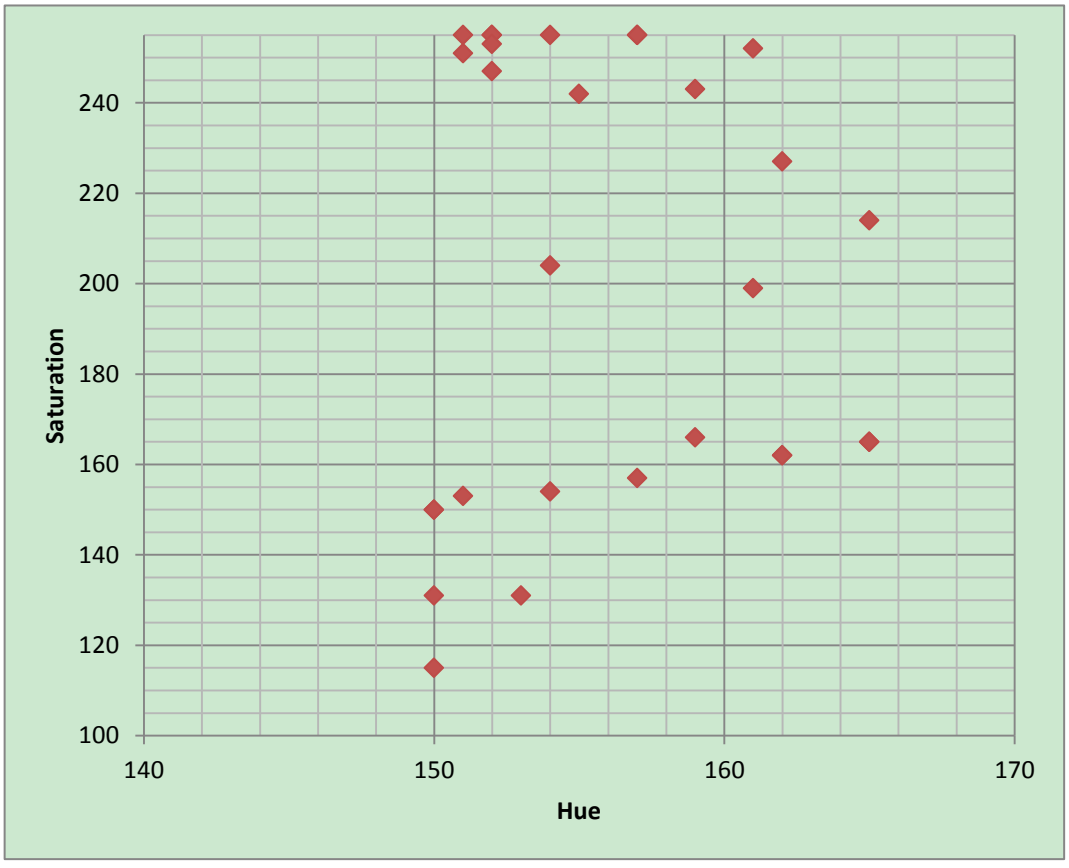
Although the Mobile Phone Paintball game implemented in this dissertation does not have

enough playability due to the limitation of current mobile phone’s hardware level, we believe the game concept we proposed will become more and more practical and popular with the development of the mobile phone hardware technology.

Appendix A

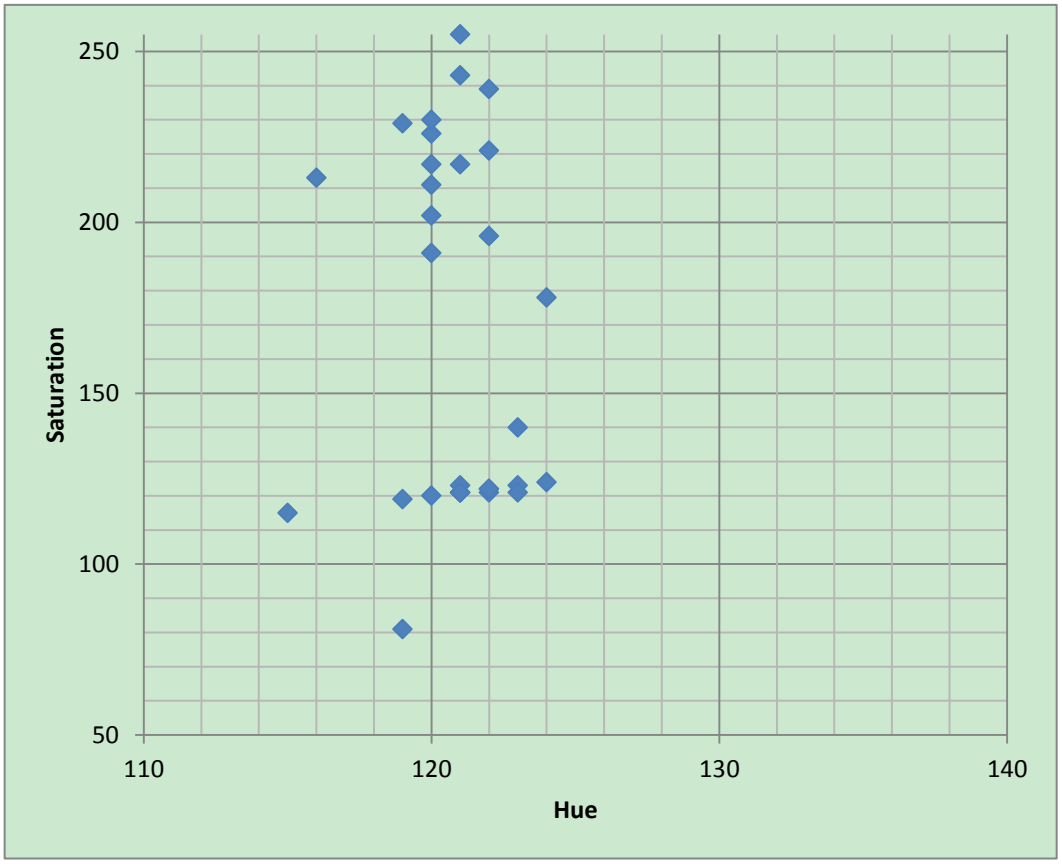
Hue and Saturation Distribution Charts

Choose hue and saturation threshold for red ( ):



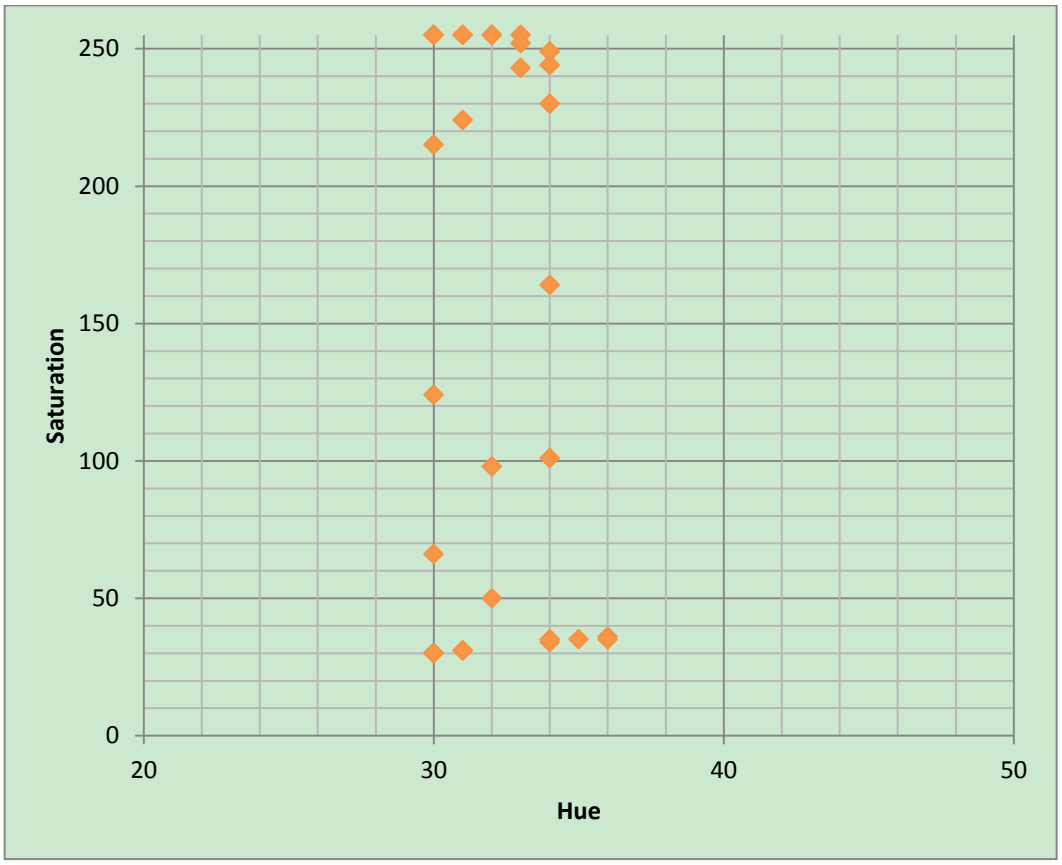
Hue: 150 – 165 Saturation: > 150

Choose hue and saturation threshold for blue ( ):



Hue: 115 – 125 Saturation: > 118

Choose hue and saturation threshold for yellow ( ):



Hue: 28 – 38 Saturation: 0 – 255

Bibliography

[1] Navneet Dalal and Bill Triggs. 2005. Histograms of Oriented Gradients for Human Detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01 (CVPR '05), Vol. 1. IEEE Computer Society, Washington, DC, USA, 886-893.

[2] J. Hu, C. Juan and J. Wang. "A spatial-color mean-shift object tracking algorithm with scale and orientation estimation," Pattern Recognition Letters, vol. 29, n. 16, pp. 2165-2173, December, 2008.

[3] S. Kar, Swati Hiremath, D.G. Joshi and V.K.Chadda, Face Verification Using Template Matching, 12th International Conference on "Advanced Computing & Communication (ADCOM-2004), Ahmedabad, pp.207-213, Dec 15-18, 2004.

[4] Ehsan Fazl-Ersi and John K. Tsotsos. 2009. Local feature analysis for robust face recognition. In Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications (CISDA'09). IEEE Press, Piscataway, NJ, USA, 333-338.

[5] Jian Yang, David Zhang, Alejandro F. Frangi, and Jing-yu Yang. 2004. Two-Dimensional PCA: A New Approach to Appearance-Based Face Representation and Recognition. IEEE Trans. Pattern Anal. Mach. Intell. 26, 1 (January 2004), 131-137.

[6] Alexander C. Berg, Tamara L. Berg, and Jitendra Malik. 2005. Shape Matching and Object Recognition Using Low Distortion Correspondences. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 – Volume 01 (CVPR '05), Vol. 1. IEEE Computer Society, Washington, DC, USA, 26-33.

[7] Koen van de Sande, Theo Gevers, and Cees Snoek. 2010. Evaluating Color Descriptors for Object and Scene Recognition. IEEE Trans. Pattern Anal. Mach. Intell. 32, 9 (September 2010), 1582-1596.

[8] Theo Gevers and Arnold W. M. Smeulders. 1997. Color Based Object Recognition. In Proceedings of the 9th International Conference on Image Analysis and Processing-Volume I - VolumeI(ICIAP '97), Alberto Del Bimbo (Ed.), Vol. I. Springer-Verlag, London, UK, 319-326.

[9] S. Belongie, J. Malik, and J. Puzicha. 2002. Shape Matching and Object Recognition Using Shape Contexts. IEEE Trans. Pattern Anal. Mach. Intell. 24, 4 (April 2002), 509-522.

[10] Peng Chang, John Krumm, "Object Recognition with Color Cooccurrence Histograms," cvpr, vol.2, pp.2498, 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR'99) - Volume 2, 1999

[11] John D. Bossler, John R. Jensen, Chris Rizos, R.B. McMaster (Eds.), Manual of Geospatial Science and Technology, Taylor & Francis, London, 2002.

[12] R. Bajaj, S. L. Ranaweera, and D. P. Agrawal. Gps: Location-tracking technology. Computer, 35(4):92-94, March 2002.

[13] Omer Rashid, Ian Mullins, Paul Coulton, and Reuben Edwards. 2006. Extending cyberspace: location based games using cellular phones. Comput. Entertain. 4, 1, Article 4 (January 2006).

[14] Goran M. Djuknic and Robert E. Richton. 2001. Geolocation and Assisted GPS. Computer 34, 2 (February 2001), 123-125.

[15] M. Vossiek , L. Wiebking , P. Gulden , J. Wieghardt , C. Hoffmann , P. Heide , S. Technol and G. Munich "Wireless local positioning", IEEE Microw. Mag., vol. 4, p.77, 2003.

[16] Ching, W.; Rue Jing Teh; Binghao Li; Rizos, C.; , "Uniwide WiFi based positioning system," *Technology and Society (ISTAS), 2010 IEEE International Symposium on* , vol., no., pp.180-189, 7-9 June 2010

[17] Timo Salminen and Simo Hosio. 2006. Region-based positioning method for cellular networks. In Proceedings of the 3rd international conference on Mobile technology, applications \& systems (Mobility '06). ACM, New York, NY, USA, Article 41.

[18] H. Liu, H. Darabi, P. Banerjee and J. Liu "Survey of wireless indoor positioning techniques and systems", IEEE Trans. Syst., Man. Cybern. C, Appl. Rev., vol. 37, p.1067, 2007.

[19] Zandbergen, Paul A. Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning. Research Article; Transactions in GIS. University of New Mexico : Blackwell Publishing Ltd, 2009.

[20] M. Bell, M. Chalmers, L. Barkhuus, M. Hall, S. Sherwood, P. Tennent, B. Brown, D. Rowland, S. Benford, M. Capra, and A. Hampshire, “Interweaving mobile games

with everyday life," in CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems. New York, NY, USA: ACM, 2006, pp. 417–426.

[21] LaMarca, A. et al. (2005) Place Lab: Device Positioning Using Radio Beacons in the Wild. Proceedings of third international conference on Pervasive computing, Munich, Germany, May 8-13.

[22] Laasonen, K., Raento, M. and Toivonen, H. (2004) Adaptive On-Device Location Recognition, Proceedings of Second International Conference on Pervasive Computing, April 18-23, Linz/Vienna, Austria, 2004, pp. 287-304

[23] E. Choi, W. Bang, S. Cho, J. Yang, D. Kim and S. Kim, "Beatbox music phone: gesture-based interactive mobile phone using a tri-axis accelerometer," IEEE Int'l Conf. Industrial Technology (ICIT '05), Hong Kong, pp. 97-102, December 2005.

[24] A.Y. Benbasat and J.A. Paradiso, "An Inertial Measurement Framework for Gesture Recognition and Applications," LNCS: Gesture and Sign Language in Human-Computer Interaction, vol. 2298, no.2002, pp. 77-90, 2002.

[25] Lopes, I.L., Vaidya, B., Rodrigues, J.R.: Sensorfall an accelerometer based mobile application. In: 2nd International Conference on Computational Science and Its Applications, pp. 749–754, 2009.

[26] Hoseinitabatabaei, S.A., Gluhak, A., Tafazolli, R., uDirect: A Novel Approach for Pervasive Observation of User Direction with Mobile Phones, Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on, pp. 74 – 83.

[27] Honkamaa, P., Siltanen, S., Jäppinen, J., Woodward, C., & Korkalo, O. (2007). Interactive outdoor mobile augmentation using markerless tracking and GPS. Proceedings of the Virtual Reality International Conference VRIC Laval France, 285-288. VTT Technical Research Centre of Finland.

[28] Volker Paelke, Christian Reimann, and Dirk Stichling. 2004. Foot-based mobile interaction with games. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology* (ACE '04). ACM, New York, NY, USA, 321-324.

[29] Forum Nokia: "Evolution of Mobile Gaming". Available at http://www.forum.nokia.com/info/sw.nokia.com/id/c52ab94e-e29d-498a-a36a-e80296e4184a/EvolutionOfMobileGaming10en.pdf.html, retrieved 25/06/2011.

[30] Sukhwant Kaur and H. K. Kaura. 2009. Design and Implementation of a Multiplayer Bluetooth Game. In Proceedings of the 2009 Second International Conference on Emerging Trends in Engineering \& Technology (ICETET '09). IEEE Computer Society, Washington, DC, USA, 1080-1085.

[31] Chan, M.M.H., Tam, V., King-Shan Lui. 2006. BlueGame - a bluetooth enabled multi-player and multi-platform game: an experience report. In proceeding of the

Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE, 258 – 261.

[32] Zhang, T. Hayes, J.R. Bell Commun. Res., AT&T Network Syst., Morristown, NJ. Client/Server Architectures Over Wide Area Networks. In Proceedings of Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on 1997.

[33] iOS Developer Library. http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html, retrieved 23/08/2011

[34] iOS Developer Library. http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html, retrieved 23/08/2011

[35] Twisted Matrix Labs. http://twistedmatrix.com/trac/, retrieved 10/08/2011

[36] SQLite. http://www.sqlite.org/, retrieved 10/08/2011

[37] Python v2.7 documentation. http://docs.python.org/library/ sqlite3.html, retrieved 10/08/2011