Technical Specification Document for the design of the IPhone Mobile Multi-Player Location Based Game named "RedVsBlue"

By

Thinksys.com

Table of Contents

Introduction

Purpose

This document provides the draft of Technical Specifications for the design of the IPhone ***Mobile MultiPlayer Location Based Game named*** "RedVsBlue" development process. The document is based on the Functional Design Specification and corresponding detailed technical analysis done by the team. This document includes all the classes and methods which are used to develop the game.

Scope:

This documentation addresses :

1. RedVsBlue system objective and architecture.
2. RedVsBlue software architecture(API).
3. RedVsBlue Database design(SQlite).
4. RedVsBlue System configuration.

The technical specification is based on expected functional and technical requirements for the development of the "RedVsBlue" IPhone game storage, maintenance and access. This document explores an alternative architecture for handling the display of dynamic content for "RedVsBlue" from a technical point of view. It also presents an analysis of the impact of various frameworks we are going to use to develop the game.

Objectives:

1. To develop a game, in which the main goal is to gain points which will allow you to unlock additional weapons and abilities, if you die you can either wait for your "resurrection" or pay with points.
2. Investigate system, hardware, software issues required to implement it.
3. Identifying the capability and additional functionalities of the game.

Technical specification:

Operating system:

iOS

Software used:

Xcode version 4.5

Frameworks:

Cocoa Frameworks

The Cocoa frameworks consist of libraries, APIs, and run times that form the development layer for all of Mac OS X. Our application will automatically inherit the great behaviors and appearances of Mac OS X and iOS, with full access to the underlying power of the UNIX operating system. Using Cocoa with the Xcode IDE is simply the best way to create native I phone applications.

Design Pattern:(Model -View-Controller)

Cocoa uses the Model-View-Controller (MVC) design pattern throughout. Models encapsulate application data, Views display and edit that data, and Controllers mediate the logic between the two. By separating responsibilities in this manner, you end up with an application that is easier to design, implement, and maintain.

Data Management:

We adopt the popular SQLite library, a lightweight yet powerful relational database engine that is easily embedded into an application. Used in countless applications across many platforms, SQLite is considered a de facto industry standard for lightweight embedded SQL database programming. Unlike the object-oriented Core Data framework, SQLite uses a procedural, SQL-focused API to manipulate the data tables directly .

.

Red Vs Blue Class Description

Main Class:

This is the first class that initiates all the other classes like player, Robot, Weapon. This class contains a main() method which is the first method executed when app starts.

Relationship With Other Classes:

This class has one-to-many relationship with other classes as this class can generate many players in the game by initiating many player class, can also generate many robots in the game.

Login class

This class maintains the registration flow and is also responsible for the player authentication. The class is also responsible for generating unique user id per each new user that sign up to the game. An unique alias will also generated by the system.

Attributes

UID :

Attribute to hold the player unique id.

UserName :

Defines the name of the player, of type String.

Password :

Attribute to hold the password of the player with respect to the user name, of type String.

Login_status:

Attribute to hold the login status of player, of type String.

Methods

addPlayer( ):

This method is used to add the player's data into the database and will generate the random unique uid for the player. This method will return the playerUid of type Alphanumeric.

verifyLogin(username: string, password: string):

This method will authenticate the player by matching player's details into the database. This method will return the login status, of type String.

FBLogin class

User can get into the game by FaceBook account. This class will make a secure connection between the player facebook account and game. This is the child class of Login class

Methods

fbSignIn(FBUsername: string, password: string):

This method will authenticate the player from his/her facebook account and verify the player. The facebook user name and password will be passed as parameter .This method will also return the login status, of type String.

DirectLogin class

If a user want to get into the game directly, this class will check the user authenticity. This class will access the player information from the database to authenticate the player. This class is derived from the Login class.

Methods

signIn(username: string, password: string):

Method will be called to authenticate the player by passing the userName and password as parameters. This class will returns the login status, of type String.

Player Class:

Central class of our application, basically all the classes works around this class. This class creates the player on the IPhone screen and is responsible for the various movement and activities of the player. This class contains many methods by the help of which player is able to play the game.

Attributes

PlayerUid :

Attribute to hold the system generated auto increment unique id of the player.

DisplayName:

Attribute to hold the display name of the player in the game, of type string.

PlayerLocation:

Attribute to hold the location coordinates on the map, of type Location

Weapons :

Attribute to describes the weapons owned by the player, of type Weapon.

FatherID:

` Attribute to hold the id of the player who is responsible for introducing this player into the game.

Coins:

Attribute to hold the count of coins with the player, as type Integer.

Methods

setTarget(location: Location) :

This method will be called when the player want to shoot down the opponent or want to defend from any attack. The location parameter in that method tells the longitude and latitude of the target with respect to data and time.

inviteFriend(playerUid: alphanumeric) :

When a player enters into the game he/she is able to invite any friend from his/her I Phone or

facebook contacts, then this method will be called.

fire(location: Location) :

Player will shoot down the opponent by calling this method.

faceToFaceCombat(playerUid: alphanumeric):

Face to Face combat will be available automatically if a player chooses to attack a target who is less than 100 meter from its own location with the help of this method.

getEnergyLevel() :

When player needs energy level or want to get, by calling this method player will get the energy pack.

movePlayer(fromLocation: Location, toLocation: Location) :

This method will be responsible for the movement of the player from one location to another on map and to-location and from-location parameter passes provides the source and destination coordinate.

placeLandmine(location: Location) :

To place the landmine on the battle ground this method will be called and location parameter provides the longitude and latitude coordinate of the location where player wants to place landmine.

chooseWeapon() :

Given a list of weapons, the player can choose a particular weapon by calling this method. This will return an object of type class Weapon.

getCountOfCoins( ) :

The player can get information of amount of coins in his/her account by calling this method.

returnToGame( ) :

Player can return into the game by calling this method.

changeAlias( ) :

By calling this method player can edit his/her display name and create a unique display name.

shareBadge( ) :

When player gets the appropriate badge after wining particular number of game, player can share the badge among his contacts, by calling this method .

getLastLocation( ) :

Player can get the information of his/her last location on the map by calling this method. This method returns a location object of type Location.

Relationship With Other Classes:

Player has one-to-one relationship with Class EnergyLevel, that means a player can have only one energy level at one time and the range of energy level is from 0 to 100 .

Player has one-to-one relationship with class Badge that means a player can earn one badge at a time. This badge represents the skill level of the player.

Player can defends himself by using IronDome class. Player has one-to-many relationship with the IronDome class.

Player has one-to-many relationship with the Coin class, that means a player can earns many coins.

Player has one-to-many relationship with the Weapon class, that means a player can have more than one weapon at one time.

Player has one-to-one relationship with the Radar class .

Ammunition Class

This class provides a way to maintain ammunition in the weapons.

Attributes

AmmunitionID:

This is system generated auto-increment id, of type int.

AmmunitionName:

This Attribute defines the name of the Ammunition, of type string.

AmmunitionType:

Attribute defines type of ammunition for a weapon, of type String.

Weapon class

This class is used to define a weapon used in the game by the player to damage another user. The weapon are of two types

(1) Direct Weapon like Gun, Hand Grenade

(2) Indirect Weapon like landmine.

Attribute

WeaponId :

This is the system generated auto-increment id, of type int.

Ammunition:

Attribute that defines the ammunition used in this weapon, of type Ammunition.

WeaponType :

Attribute defines the type of the weapon, of type String.

WeaponRange:

Each weapon has its own range and this attribute is to hold the range of weapon, of type float.

Methods

getWeaponInfo(weapon:Weapon):

By calling this method player can get the information of any weapon in the game like its range, type its price. This method will return the information about the weapon in string type. Parameter passes into that method will be a object of type class Weapon.

upgradeWeapon(playerplayerUid:alphanumeric, coin: Coin):

To upgrade the weapon, this method will be called by passing two parameters playerUid of type alphanumeric and an object coin of type class Coin. This method will return a new upgraded weapon having more fire power and impact of type class Weapon.

Relationship with other class:

Weapon class has many-to-one relationship with the Player Class.

IndirectWeapon class

This class is derived from Weapon class mentioned above. This class helps to calculate the impact of indirect weapon like landmine.

Methods

calculateImpact(weapon: Weapon, precision: float, swipingpower: float):

This method will be called to get the impact of a particular weapon by passing the objects weapon of type class Weapon, precision of type float and swipingpower of type float as parameter. This method will return the impact of weapon in float type.

DirectWeapon class

Like the Indirect weapon class mentioned above ,this class also derived from weapon class to calculate the impact of direct weapon used in the game like gun, grenade.

Methods

calculateImpact(weapon: Weapon, precision: float, swipingPower: float):

To get the impact of direct weapon by passing the objects weapon of type class Weapon, precision of type float and swipingpower of type float as parameter. This method will return the impact of weapon in float type.

Badge:

When a player wins particular number of games, the player will be awarded with the badges. Each badge is associated with a skill level. This class used to implement the badge system in the game.

Attributes

BadgeID:

Attribute to hold the unique id of badge in the game, of type int auto increment.

BadgeName:

Attribute to hold the name of each badge in the game, of type string.

BadgeLevel:

Attribute to hold the level of each badge, of type string.

NumberOfWins:

Attribute to hold the number of wins, of type Integer.

Methods

getBadge(numberOfWin: int) :

When a player wins the enough game to achieve a particular skill level, this method will be called by passing the number of games won as parameter. This method will return an object of type Badge.

EnergyLevel class

The ultimate goal of the the game is save energy and getting coins in the game. Player enters into the game with 100 energy level and on each hit energy level decreases, to implement the energy system in the game, the EnergyLevel class will be used.

Attributes

Range:

Attribute to hole the range of energy between 0 to 100. This attribute is of type Integer.

Methods

increaseEnergyLevel(fak: FirstAidKit):

Whenever a player need to increase his energy level in the game, he/she can get the energy pack from the first aid kit. This will return a float that will represent energy.

decreaseEnergyLevel(weapon: Weapon):

Whenever a player gets hit from the opponent, this method is automatically called.

This will decrease the energy level of the player. The decrease in energy level is propotional on the type of weapon used and its impact. This will return a float that will represent energy.

Relationship with other class:

Player has one to many relationship with the FirstAidKit class, that means a player could get the more than one energy packs .

FirstAidKit class

This class provides the energy pack or First aid kit to the player. These energy packs are distributed on the battle ground and player can get back into the game by restoring energy by the help of these packs. System will automatically spread the packs to random locations.

Attributes

KitID:

This is auto increment id generated by the system.

KitLocation:

Describes the location of kit pack, object of class Location.

Methods

setKITLocation(location: Location):

Method will be called to place the energy pack on location passed in the method, by passing object of type Location as parameter.

Mapview class

To implement the map logic in the game Mapview class will be used. Methods in this class are called to get location on map or to set location which helps to move player from one position on map to other position on map.

Attributes

MapType:

Attribute to hold the Maptype name, of type string.

UIDLocation:

Attribute to hold the coordinate of the player location on map, object of type class Location.

LastLocation:

Attribute to hold the coordinate of the player's last location on map, object of type class Location.

Methods

getLocation( ):

Method will be called to get the location on map and will return an object of type Location.

setLocation(x, y, dataTime):

Method will be called to set the location on map and returns void.

setZoomLevel( ):

Method will be called to set various zoom in and zoom out level on map and returns void.

IronDome class

This class will be used to defend the player from the opponent attack. The catapult and KipatBarzal tools will be used by the player to defend.

Attributes

IronDomeID :

Attribute to hold the unique Id associated with the iron dome, of type Integer.

IronDomeName:

Defines name IronDome, of type string.

IronDomeType:

Attribute to hold the type of particular iron dome, of type string.

Relationship with other class:

This class has many-to-one type relationship with Player class.

Catapult class

This is a child class of IronDome class. It represents a Catapult that is used to defend from the enemy attack.

Methods

shootFromCatapult(toLocation: Location):

By calling this method player can shoot the target from catapult, by passing the object toLocation of type Location as parameter.

This method returns void.

KipatBarzel class

Player defends and shoot the target by using this class, which is the child class of IronDome class mentioned above.

Methods

shootFromKB(toLocation: Location):

By calling this method player can defend and shoot the target from Kipat Barzel technique, by passing the toLocation object of type class Location as parameter into the method which returns void.

Coin class

To maintain the economics in the game this class is used. The methods of this class helps in keeping the track of each player's account and also to locate the coin stashes spreads over the battle ground.

Attributes

CoinID:

Attribute to hold the system generated auto increment id for the coin, of type Integer.

PackName:

Attribute to hold the name of a specific coin pack in the game, of type string.

Methods

CoinStashLocation:

Attribute to hold the coordinate of location of coin stashes on the battle ground, object of type class Location.

buyCoin(playerUid: alphanumeric, packname: string):

By calling this method player will get the coins to be in the game or to survive into the game, by passing unique id of the player who needs coins to acquire new weapon,skills,energy packs. The packname will also be passes as parameter of type string.

getAccountInfo(playerUid: alphanumeric):

At any stage of the game player can get the information about the account by calling this method, by passing the palyerUid as parameter into the method. The method will return the account information in string format.

getCoinStashLocation(location: Location):

To get the exact location of the coin stash spreads over the battle ground this method will be called by passing location object of type class Location into the method.

setCoinStashLocation(location: Location):

This method will place the coin stash to the location passed into the method. Parameter is the object location of type class Location.

Radar class

This class will be used to defend the player by implementing the radar system. The methods of this class will be called when a player wants to choose a radar options.

Attributes

RadarID:

Attribute to hold the system generated auto increment unique ID.

RadarName :

Attribute to hold the radar name, of type string.

Methods

isUnderRadar(weapon: Weapon):

To check weapon is under radar or not, this method will be called by passing the object weapon of type class Weapon as parameter. This method will return status in String type.

getUnderRadarLocation(weapon :weapon):

To get the location of the weapon which is under radar, this method will be called by passing the object weapon of type class Weapon as parameter. This method will return the object of class Location.

Robot class

This class will implement the robots logic in the game. The idea to create “Like-Humans” participants that a regular user couldn't differentiate between them and a real user. Each robot will be associated with unique robot id and they will move a particular path.

Attributes

RobotID :

Attribute to hold the system generated auto increment id, of type Integer.

Interaction_frequency :

Attribute to hold the movement frequency of the robot, of type Integer.

RobotPath :

Attribute to hold the array of locations, of type Location.

Methods

***moveRobot(fromLocation*: Loaction, toLocation: Location):**

To move the robot on a particular path on map, this method will be called by passing the source and destination coordinate on map. The objects fromLocation and toLocation of type class Location, will be passed as parameters.

fireRobot(location: Location):

The system generated robots will fire by calling this method and passes the location ,where they want to shoot. The object location of type class Location as parameter. This method will return void .

Location class

At different stages of the game, player and system both requires locations on map. The location will contain the longitude and latitude coordinates on map. And also require to get and set the locations at different stages. This class will define x and y coordinate of object with respect to date and time.

Attributes

X_location :

Attribute to hold the x coordinate of the object, of type float.

Y_location :

Attribute to hold y coordinate of the object, of type float.

Methods

getLocation():

At any stage of the game player and system can get the location of any object on map by calling this method. This method will return x and y coordinate of type float.

setLocation(x: int, y: int, date):

System and player will be able to place any object (like energy packs by the system and to shoot by the player) on battle ground by passing x and y coordinate with respect to date and time.

getDate&Time():

By calling this method anyone can get the date and time. This method will return an object of type Date.

RedVsBlue Class Diagram