BattleBoats

About the Application

BattleBoats is an addictive game of luck based upon the well known 'Battleships'. It has both single and multi-player modes where the user guesses where their opponents boats are and tries to destroy them before their own demise. On a successful hit, teams are awarded points.
There are four different difficulty settings which determines how many boats each team has. You can play as either 'red' or 'blue' teams.
At the end of each game, users can submit their score to the high score table which is hosted on my personal web server. The score can be geo-tagged with the devices location so that scores can be displayed on a map.
The multi-player mode is based upon a HTTP protocol with my web server so that any two devices in the world can play each other in real-time.

This tutorial will be broken up into the different views of the application. The first step however, is to set up the global game data and share it across all views we create.

Step 1 – Global Game Data

- Create a new view based iPhone project and call it BattleBoats.
- Add a new class and call it Game (along with header file).
- We are going to add the following *ivars* to Game:

```
NSInteger teamColor
NSInteger difficulty
NSInteger turnNumber
NSInteger teamScore
NSInteger opponentScore
NSInteger teamNumberOfBoatsSunk
NSInteger opponentNumberOfBoatsSunk

NSMutableArray teamMoves
NSMutableArray opponentMoves

NSInteger MP_flag
long MP_uid
long MP_opponent
int MP_gid

NSInteger team_boat1_x
NSInteger team_boat1_y
NSInteger team_boat2_x
NSInteger team_boat2_y
NSInteger team_boat3_x
NSInteger team_boat3_y
NSInteger team_boat4_x
NSInteger team_boat4_y
NSInteger team_boat5_x
NSInteger team_boat5_y

NSInteger opponent_boat1_x
NSInteger opponent_boat1_y
NSInteger opponent_boat2_x;
NSInteger opponent_boat2_y;
NSInteger opponent_boat3_x;
NSInteger opponent_boat3_y;
NSInteger opponent_boat4_x;
NSInteger opponent_boat4_y;
NSInteger opponent_boat5_x;
NSInteger opponent_boat5_y;
```

The first group are general game variables and counters; the two arrays are going to hold each of the teams moves; there are then some multiplayer mode specific variables and then the boat positions are all held as separate *ivars* to make referencing them easier.

– Declare the normal properties and synthesize each in the main file.
– Import the Game.h into the BattleBoatsAppDelegate header file and add a Game ivars called gameData.
– Set the properties and synthesize in the app delegate main file.
– In the first method called didFinishLaunching, the first thing we want to do is initialise the game data.

```
GameData = [[Game alloc] init];
```

In the main view header file (BattleBoatsViewController.h), we are going to add a Game *ivar* called localGameData.

– Set the properties and synthesize in the main file
– In the viewDidLoad method, set up the local game data to be shared with the application delegate class:

```
//INITIALISE GAME DATA AND STORE AS LOCAL
BattleBoatsAppDelegate *appDelegate = [[UIApplication sharedApplication] delegate];
localGameData = [appDelegate gameData];
```

This allows any changes to localGameData to be reflected to the global game data.

Step 2 – The Main Menu

The application launches into the main menu.
– Set the background image (image resource: 'menu.png').

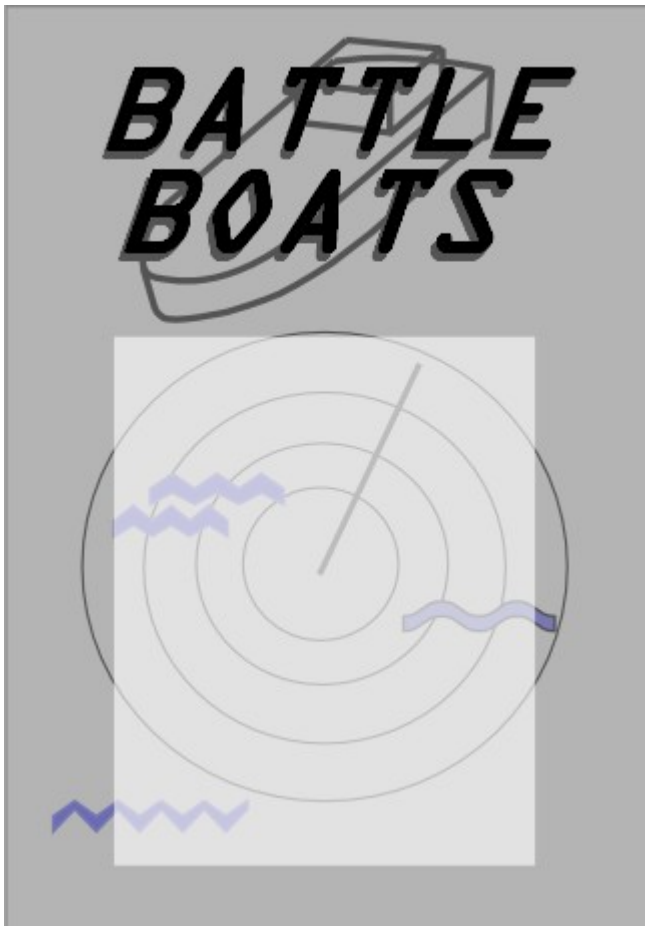

Figure 1: menu.png

– Initialise a rounded button with the title set to 'New Game' and position centrally horizontally. Add a target action to call a method **newGame**.
– Add a 'multiplayer' button underneath and get it to call a method called **multiplayer**.
– Add a button labelled High Scores that calls a method called **highScores**
– Set up a slider that has a minimum of 2 and maximum of 5. Set the default to 4. Add a call to **difficultySliderChanged** method.
– Add a label above the slider with the text set as 'Easy'
– Set up and add a 2 segment control with 'Blue' and 'Red'
– Ensure all of the buttons, the slider, label and segment control have been set up in the header and synthesized in the main file.
– Add the 4 method declarations to the header file. They don't return or take any values.

We are now going to define these 4 methods.

newGame

– Set the game variables teamColor from the segments and difficulty from the slider.
– turnNumber, teamScore, opponentScore, teamNumberOfBoatsSunk and opponentNumberOfBoatsSunk, MP_flag need to all be set to zero. Set each boat position to 0 and using a loop, fill the team and opponent move arrays with 90 zeros.
– Start an instance of y*ourView* (which we will define later) and using an animation (I suggest UIModalTransitionStyleCoverVertical), present that view.

multiplayer

– Set the game variables the same as in newGame but this time the difficulty should be set to 4 and the MP_flag set to 1. The MP_gid and MP_uid also need to be set to zero.
– Start an instance of *multiplayerSetupView* (again, we will define later) and use an animation to present the view.

highScores

– Start an instance of W*ebView* and using a transition, move to that view. We will create WebView later.

difficultySliderChanged

– This method will change the label text to show what difficulty the slider is set to.
– Write *if* statements for integer values 2-5 setting the label text to correspond to the following difficulty settings:
  2: Very Hard
  3: Hard
  4: Easy
  5: Very Easy

– Make sure you have done memory clean up for everything that was manually *alloc'd*
– Ensure that the following header files have been imported (some we haven't made yet) :
  Game.h
  BattleBoatsAppDelegate.h
  YourView.h
  WebView.h
  MultiplayerSetupView.h

If all has gone well, you should have a view that looks similar to figure 2.

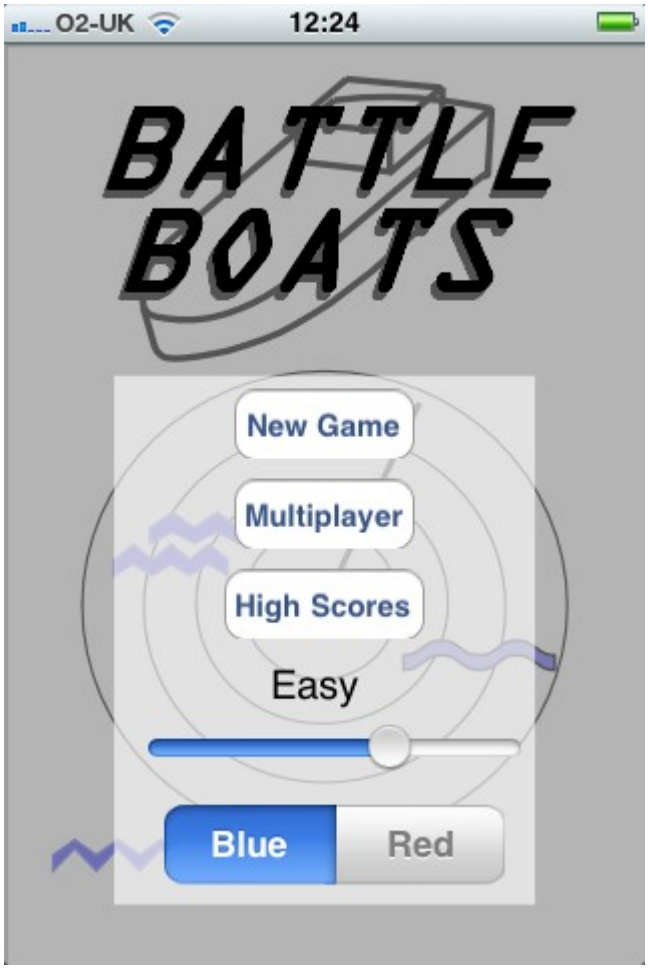

Figure 2: The main menu

Step 3 – YourView

- Create a new view called YourView. This class will look after the users boats.
- Declare localGameData ivar and synthesize it. Add the same code as the main view controller to initialise a shared resource of the global game data.
- Set the background image depending on the selected teamColor. If the teamColor is set to 1 then the background should be red (image resource: 'red.png'), else the teamColor is 0 and the background should be blue (image resource: 'blue.png').
- Add a rounded button with title 'Menu' to the bottom right hand corner of the display (I would suggest 60x30 centred at 285, 440). Have a method named **gotoMenu** called on touch (don't forget to add a button ivar and synthesize it).
- We are now going to add the 'score card' which displays the current team and opponent scores. The background colour of the score cards will depend on the team colours.
- Initialise two temporary numbers of *float* type called red and blue. If the teamColor is set to 1 (red) then set the red float to 50.0 and the blue to 0.0; else if teamColor is set to 0 (blue), set the blue float to 50.0 and red to 0.0.
- Add a label called labelTeamScore, made as with a rectangular frame: 10,425,70,30.
- Set the text to '0', font size to 25 and give it a black border.
- Set the background colour using the temporary floats to define the red and blue:

```
[UIColor colorWithRed:red green:0.0 blue:blue alpha:0.9]
```

- We now need to do the same to make the opponent score card (called labelOpponentScore) with a couple of changes. Set the frame to 80,425,70,30 and swap the red and blue floats around:

```
[UIColor colorWithRed:blue green:0.0 blue:red alpha:0.9]
```

- We now will set up 2 SFX (sound effect) players to play splash and explosion effects. Before we can use the audio players, we need to add a framework to the project. Add `<AVFoundation/AVFoundation.h>` to the import list in the YourView header file.

– Initialise a string called explosionPath and set it to:

```
[[NSBundle mainBundle] pathForResource:@"explosion" ofType:@"wav"];
```

This creates a path to the explosion.wav file.

– Initialise a URL called explosionURL and convert the explosionPath into the URL:

```
[[NSURL alloc] initFileURLWithPath:explosionPath];
```

– We can now create the player, let's called it 'sfxExplosionPlayer'. Add it to the header along with the properties and synthesize in the main file. Initialise it with URL set to explosionURL and call prepareToPlay.
– We can now release the path and URL as they are no longer required.
– Do the same again but for splash. The file is called splash.wav.

The view is now set up ready for interaction from the user. We now will define the methods we shall use. Declare gotoMenu, gotoOpponentView and opponentReady in the YourView header file.

gotoMenu

– This method just returns back to the main menu by dismissing this modal view through animation.

```
[self dismissModalViewControllerAnimated:YES];
```

gotoOpponentView

– This method initialises the opponent view and transitions to the new modal view (suggested transition: UIModalTransitionStyleCrossDissolve

touchesBegan

This method is called whenever a touch event occurs on the device.

```
(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

}
```

The only user interaction with YourView is at the beginning of a new game. The user needs to position their boats. This method checks to see if all the users boats have been placed.

– For each boat, check if the x and y position is 0.
– Get the touch location, convert it into the square co-ordinates (as shown below) and check to see if it is in the playable area.
– If the touch is in the playable area, draw the boat (a blank label) in that square.

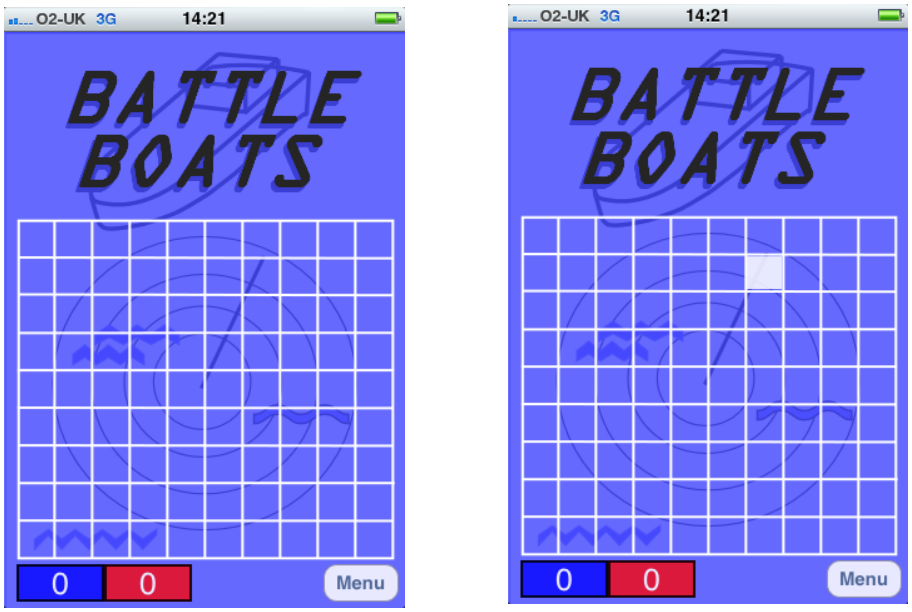

Figure 3: Grid system and placed boat

```
if((localGameData.team_boat1_x == 0) && (localGameData.team_boat1_y == 0)) {
     UITouch *touch = [[event allTouches] anyObject];
     CGPoint location = [touch locationInView:touch.view];
     NSLog(@"Touch Event at: %f, %f",location.x,location.y);
     //check if its in playable area
     NSInteger y_cord = (int) ((location.y - 152) / 30)+1;
     NSInteger x_cord = (int) ((location.x - 13) / 30)+1;
     if((x_cord<=10)&&(x_cord>0)&&(y_cord>0)&&(y_cord<=10)) {
          team_boat1 = [[UILabel alloc] initWithFrame:CGRectMake((location.x)-14,
(location.y)-14,28,27)];
          team_boat1.backgroundColor = [UIColor whiteColor];
          [self.view addSubview:team_boat1];
     }
}
```

Do this for each boat, the number depending on the difficulty level. For example add to the if statement a check to see if the difficulty level is set to greater than 3 to add a forth boat:

```
else if((localGameData.team_boat4_x == 0) && (localGameData.team_boat4_y == 0) &&
(localGameData.difficulty>3)) {
```

– Add ivars for team_boat1 – 4 to the header file as labels and synthesize.

touchesMoved

This method is called when an active touch event moves location.
Going through each boat to check which does not have it's position set yet (ie team_boat1_x == 0), we will move the active label to the new location of the touch event.

```
if((localGameData.team_boat1_x == 0) && (localGameData.team_boat1_y == 0)) {

     UITouch *touch = [[event allTouches] anyObject];
     CGPoint location = [touch locationInView:touch.view];
     //MOVE BOAT
     team_boat1.center = location;
}
```

– Add a check for all team boats (1-4)

touchesEnded

This method is called when a touch event ends (when the user lifts their finger).
For each boat again, we are going to convert the location of the drop to the nearest square co-ordinate and check that the drop was done in the playable area. If it is, save the position to the global game data.

If we are not in multi-player mode (MP_flag==0), we need to create an opponent position and save that to the global game data.

If the drop occurred outside the playable we need to remove the boat from the view.

Each if statement should look similar to:

```
if((localGameData.team_boat1_x == 0) && (localGameData.team_boat1_y == 0)) {
     CGPoint location = team_boat1.center;
     NSInteger y_cord = (int) ((location.y - 152) / 30)+1;
     NSInteger x_cord = (int) ((location.x - 13) / 30)+1;
     if((x_cord<=10)&&(x_cord>0)&&(y_cord>0)&&(y_cord<=9)) {
          team_boat1.center = CGPointMake((((x_cord-1)*30)+25),(((y_cord-1)*30)+165));
          localGameData.team_boat1_x = x_cord;
          localGameData.team_boat1_y = y_cord;
          if(!localGameData.MP_flag) {
               localGameData.opponent_boat1_x = (arc4random() % 10) + 1;
               localGameData.opponent_boat1_y = (arc4random() % 9) + 1;
          }
     }
     else
          [team_boat1 removeFromSuperview];
}
```

This line is to align the centre of the boat into the middle of the square.

```
team_boat1.center = CGPointMake((((x_cord-1)*30)+25),(((y_cord-1)*30)+165));
```

On the forth boat, if in multi-player mode, we need to upload the boat positions to the server and wait for the opponent positions to be set. The boat positions are given as the square number rather than co-ordinates. This conversion is simply:

```
boat1 = ((localGameData.team_boat1_y*10)-10)+localGameData.team_boat1_x
```

– Create a string to form the URL with the game variables and boat positions in the format below:

```
http://api.dylanjones.info/battleboats/myboats?
gid=MP_gid&uid=MP_uid&boats=boat1,boat2,boat3,boat4
```

– Execute the HTTP request
– Add a place-holder label called *labelWaiting.* Add it to the header and synthesize it in the main file. Make it 200x120 and place it in the centre of the display.
– Set the place-holder text to “Waiting for Opponent”.
– Call the method **opponentReady** (we'll define later).

– For each boat placement, check if the game difficulty level is that boat number. For example, boat 3, check if difficulty==3.
– If true, call **gotoOpponentView** (defined earlier).

Summary: We have placed the users boats and stored their positions in the global game data. If we're in multi-player mode, we have submitted the users boats to the server.

opponentReady

This method makes a HTTP request checking for the opponent's boat positions. If none are returned it calls itself after a short time delay.

– Start of by checking if the opponents boats have been set by testing opponent_boat1_x. If it's 0, call opponentReady after a 12 second delay:

```
[self performSelector:@selector(opponentReady) withObject:nil afterDelay:12];
```

– Generate the URL string in the format:

```
http://api.dylanjones.info/battleboats/ready?gid=MP_gid&uid=MP_uid
```

– If the response string has a length greater than 5 then the response contains the boat positions.
– Create a temporary array called opponentBoats using the componentsSeperatedByString method. The boat positions are separated by a comma.
– For each position in the array, convert the square number to the co-oridnate system and store the x and y position in the global game data. For example, the first boat position is extrapolated by:

```
int i = [[opponentBoats objectAtIndex:0] intValue];
if(i<=10) {
      localGameData.opponent_boat1_x = i;
      localGameData.opponent_boat1_y = 1;
}
else if(i<=20) {
      localGameData.opponent_boat1_x = i-10;
      localGameData.opponent_boat1_y = 2;
}
else if(i<=30) {
      localGameData.opponent_boat1_x = i-20;
      localGameData.opponent_boat1_y = 3;
}
else if(i<=40) {
      localGameData.opponent_boat1_x = i-30;
```

```
            localGameData.opponent_boat1_y = 4;
      }
      else if(i<=50) {
            localGameData.opponent_boat1_x = i-40;
            localGameData.opponent_boat1_y = 5;
      }
      else if(i<=60) {
            localGameData.opponent_boat1_x = i-50;
            localGameData.opponent_boat1_y = 6;
      }
      else if(i<=70) {
            localGameData.opponent_boat1_x = i-60;
            localGameData.opponent_boat1_y = 7;
      }
      else if(i<=80) {
            localGameData.opponent_boat1_x = i-70;
            localGameData.opponent_boat1_y = 8;
      }
      else if(i<=90) {
            localGameData.opponent_boat1_x = i-80;
            localGameData.opponent_boat1_y = 9;
      }
```

- Repeat this same method for each of the opponent boats (1-4).
- Once all the opponents boats positions have been stored in the global game data, the waiting place-holder can be removed and the **gotoOpponentView** method called.

viewDidAppear

This method is called each time the 'YourView' view comes into view. As the game progresses, the active view switches between the Opponent and Your Views.

- First check that we are in game play by verifying that the turnNumber is greater than zero.
- Iterate through the opponentMoves array, converting the index to square co-ordinates and testing if the square has been hit or splashed, drawing an image accordingly.

```
int count = [localGameData.opponentMoves count];
for(int a=0; a<count; a++) {
      int i = a;
      int x_cord;
      int y_cord;
      if(i<=10) {
            x_cord = i;
            y_cord = 1;
      }
      else if(i<=20) {
            x_cord = i-10;
            y_cord = 2;
      }
      else if(i<=30) {
            x_cord = i-20;
            y_cord = 3;
      }
      else if(i<=40) {
            x_cord = i-30;
            y_cord = 4;
      }
      else if(i<=50) {
            x_cord = i-40;
            y_cord = 5;
      }
      else if(i<=60) {
            x_cord = i-50;
            y_cord = 6;
      }
      else if(i<=70) {
            x_cord = i-60;
```

```
            y_cord = 7;
      }
      else if(i<=80) {
            x_cord = i-70;
            y_cord = 8;
      }
      else if(i<=90) {
            x_cord = i-80;
            y_cord = 9;
      }
      else{
            NSLog(@"The maths has gone wrong %i",i);
      }
      int squareStatus = [[localGameData.opponentMoves objectAtIndex:a] intValue];
      if(squareStatus==1) {

            //DRAW EXPLOSION
            UIColor *explosionImage = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"explosion.png"]];
            explosion = [[UILabel alloc] initWithFrame:CGRectMake((((x_cord-1)*30)+12),
(((y_cord-1)*30)+152),28,27)];
            explosion.backgroundColor = explosionImage;
            [self.view addSubview:explosion];
            [explosionImage release];

      }
      else if(squareStatus==2) {

            //DRAW SPLASH
            UIColor *splashImage = [UIColor blackColor];
            splash = [[UILabel alloc] initWithFrame:CGRectMake((((x_cord-1)*30)+12),
(((y_cord-1)*30)+152),28,27)];
            splash.backgroundColor = splashImage;
            [self.view addSubview:splash];
            [splashImage release];

      }
      else {
            //DO NOTHING
      }

}
```

- To see if we need to play a sound effect, we test the current opponent score card against the global data. If the global game data says that the opponent score is greater than the current score card value then the opponent must have hit a boat so play the explosion sound effect.
- If the global data matches the current score card then the opponent did not hit a boat so play the splash effect.

```
int stringTest = [labelOpponentScore.text intValue];
if(stringTest<localGameData.opponentScore) {
      //PLAY EXPLOSION
      [sfxExplosionPlayer play];
}
else {
      //PLAY SPLASH
      [sfxSplashPlayer play];
}
```

- Update the score cards with the current scores
- Check to see if we need to end the game. Test to see if either opponentNumberOfBoatsSunk or teamNumberOfBoatsSunk equal the game difficulty level
- If true, the game can end with an alert view, else go to opponent view for the next turn.
- Add an alertView to the header file and synthesize in the main file.
- If the teamScore > opponentScore then the user has won and they see a “Congratulations” title in the alert view else if they lost a “You Lost”.
- Create a message string that asks the user if they want to submit their score to the high

scores table.
- Add the UIAlertViewDelegate to the header file and initiate an alertView with the approriate title and message string.
- Set the cancel button to have the title “No Thanks” and another button to “Submit”.
- Add a UITextField to the header, synthesize it, give it a place-holder “Nickname” and add it to the the alert view.

alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex

This method will be called when the user presses either of the buttons in the alert view.

- If the buttonIndex is 0 then the user has pressed “No Thanks” and does we do not need to do anything else. The game can be closed by calling the dismissModalViewController method to return to the main menu.
- If buttonIndex is 1 then the user would like to submit their score.
- Add the CLLocationManagerDelegate to the header file as well as ivars for the location manager and user location. Add the properties and synthesize them in the main file.
- Initialise an instance of the location manager, set the distance filter to 2000 and start updating the user location.

locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation

This method is then called when the location has been retrieved from the device

- Stop updating the users location.
- Create a string with the URL to submit the high score in the format:

```
http://api.dylanjones.info/battleboats/highscore/?
nickname=NICKNAME&score=SCORE&latitude=LATITUDE&longitude=LONGITUDE
```

- Make the HTTP request with the variables set.
- Dismiss the modal view to return to the main menu

Summary: We have set up all the methods that update YourView with the opponent moves, drawing the explosions, splashes, playing the sound effects and updating the score cards. We have declared the method to get the opponent boat positions in multi-player mode. We alert the user to the end of the game and allow them to submit their score along with their location to the server.

Step 4 – Opponent View

This view gets and processes the users move. In single player mode, generates and processes an opponent move.

- Create a new view called OpponentView
- Add the BattleBoatsAppDelegate and Game header files to the OpponentView.m and header file.
- In the viewDidLoad method, initialise an instance of localGameData, just like in the YourView.m adding the ivar to the header file.
- Add the background image like before but with the images reversed ie if YourView was red, set OpponentView to blue.
- Add a rounded button to the bottom right hand corner with the text 'Back' and get it to call a method named **goBack**.
- Initialise the score card, the same as in YourView.
- Add the sound effect players.
- Increment the turnNumber.
- Iterate through the teamMoves array, converting the index into the x and y co-ordinates and drawing explosions and splashes as appropriate (use the code from YourView).

goBack

- Create a method called goBack, declare it in the header file.
- Add a call to dismiss the modal view to return back to the 'YourView'.

opponentReady

- Create a method named opponentReady. This method will be only used in multi-player mode to check if it's the user's turn.
- Initialise a string in the following format to make a HTTP request to the server:

```
http://api.dylanjones.info/battleboats/turn?gid=MP_gid&uid=MP_uid
```

- Check the response string if it is '0'. If it is then call opponentReady again after a delay of 5 seconds.
- If the response is not 0 then create an array (called response) with components of the response string separated by a comma.
- If the first object in the array is “HIT”, then add points to the opponents score based on the turn number:

```
localGameData.opponentScore =
localGameData.opponentScore + ((90 - localGameData.turnNumber) * 10);
```

- Increment the team number of boats sunk.
- Add the move into the opponent move array by replacing the existing zero value with a 1 for a hit, ie:

```
[localGameData.opponentMoves replaceObjectAtIndex:[[response objectAtIndex:1] intValue]
withObject:[NSNumber numberWithInt:1]];
```

If the response is not “HIT” then all we need to do is store the move as a miss:

```
[localGameData.opponentMoves replaceObjectAtIndex:[[response objectAtIndex:1] intValue]
withObject:[NSNumber numberWithInt:2]];
```

- Call the **goBack** method after a 0.5 second delay to go to YourView

touchesBegan

This method will deal with touch events on the opponent view where the user will be targetting the opponent's boats.

- Get the location of the touch event and convert the position into square co-ordinates using the same conversion done in YourView:

```
NSInteger y_cord = (int) ((location.y - 152) / 30) +1;
NSInteger x_cord = (int) ((location.x - 13) / 30) +1;
```

- If the touch is in the playable area, check to see if the user has made that move already by checking the team moves array at the square number index:

```
int here = ((y_cord*10)-10)+x_cord;
if([[localGameData.teamMoves objectAtIndex:here] intValue]==0) {
}
```

- If the move has not been done before, test to see if the move is a hit by checking if any of the co-ordinates match up:

```
if(
((x_cord==localGameData.opponent_boat1_x)&&(y_cord==localGameData.opponent_boat1_y)) ||
((x_cord==localGameData.opponent_boat2_x)&&(y_cord==localGameData.opponent_boat2_y)) ||
((x_cord==localGameData.opponent_boat3_x)&&(y_cord==localGameData.opponent_boat3_y)) ||
((x_cord==localGameData.opponent_boat4_x)&&(y_cord==localGameData.opponent_boat4_y)) ||
((x_cord==localGameData.opponent_boat5_x)&&(y_cord==localGameData.opponent_boat5_y))
) {
}
```

- If there is a match, draw an explosion in that square.
- Play the explosion sound effect.
- Add points to the team score, using the same calculation as in the **opponentReady** method.
- Record the move as a hit in the team moves array.
- Increment the opponent number of boats sunk count.
- If there is not a match, draw a splash and play the splash sound effect.
- Record the move as a miss in the team moves array.

- If the multi-player flag is raised (MP_flag==1) then submit the move to the server by the following HTTP request:

```
http://api.dylanjones.info/battleboats/move?gid=MP_gid&uid=MP_uid&move=here
```

- Add a place-holder label, like the one added earlier.
- Fire the **opponentReady** method.

- If the multi-player flag is off then the opponents move has to be generated randomly.
- Use the `arc4random` function to generate a position in a do...while loop checking if that randomly generated move has been done before in the opponent moves array:

```
//AI TURN
int here = 0;
int squareStatus = 0;
do{
      x_cord = (arc4random() % 10) + 1;
      y_cord = (arc4random() % 9) + 1;
      here = ((y_cord*10)-10)+x_cord;
      squareStatus = [[localGameData.opponentMoves objectAtIndex:here] intValue];
} while(squareStatus!=0);
```

- Check to see if the opponent move matches a location of one of the teams boats using the long if statement similar to the one used above (just change opponent to team in the variable names).
- If it is a hit, add the points, record the hit in the opponent moves array and increment the number of team boats sunk count.
- If it is a miss then just add the move to the opponent moves array.
- Fire the **goBack** method to return to YourView.
- Ensure all the memory management has been done for this class.

Summary: The opponent view is set up to handle the users go and generate a random go for the opponent in single-player mode, else wait for the opponent to take their turn in multi-player mode. This along with YourView handles all the gameplay. The next stage is to create the high score view.

Step 5 – High Scores

The high scores table is a web page generated by my web server and delivered to the device in a simple web view.

- Create a new view and call it WebView
- Add an ivar called webView of type UIWebView, add the properties and synthesize it in the main file.
- Start an instance of webView that covers the whole screen.
- Create a string "http://dylanjones.info/battleboats/highscores" and convert it to a NSURL.
- Load the URL into the webView.
- Add two rounded buttons (including ivars, properties and synthesis) to the lower right hand side of the view called "Back" and "Map". Call **goBack** and **gotoMap** for touch events on the buttons, respectively.

goBack

This method will return to the main menu by dismissing the modal view.

- Add the self dismissing call to the method.

gotoMap

This method initialises an instance of MapView and transitions to the new view.

- Create an instance of MapView and set a transition style.
- Present the new modal view.

- Create a new view called MapView, add the header file to the main file of WebView.
- In the header file of the new MapView, add the MapKit framework.
- Add the delegate: `MKMapViewDelegate`
- Create ivars for the MapView (MKMapView), a button and NSMutableData called responseData. Add the properties and synthesize in the main file.

- Create a new class called Annotation, add the MapKit framework.
- Add the `MKAnnotation` delegate to the header file and the following ivars:

```
CLLocationCoordinate2D coordinate;
NSString *title;
NSString *subtitle;
```

- Add the properties and synthesize them in the main Annotation file.

- Back in the MapView main file, import the Annotation header file.
- Initialise an instance of MapView that is the size of the screen with a standard map type, showing the users location.
- Make a URL request to:

```
http://api.dylanjones.info/battleboats/map
```

The response should be similar to:

```
Fred,3490,53.809680,-1.554218,129.11.76.215,1291126506;
Dylan,3470,53.809681,-1.554218,129.11.76.216,1291141439;
```

- Convert the response string into an array with the components separated with a semi-colon. This splits the response into an array with the separate scores as separate objects.
- For each object, separate the string again but this time separated with a comma, spliting the string into the different data values: nickname, score, latitude, longitude, IP Address and timestamp.
- For each high score, create an instance of an annotation.
- Set the title to the nickname, the subtitle to the score and the coordinate to a *CLLocationCoordinate2D* created from the latitude and longitude.
- Add the annotation to the map.

- Add a button to the MapView with title “Back”, position it in the lower right hand corner and add the **goBack** method to be called on a touch event.

goBack

This method returns to the WebView high score table

- Add a call to dismiss the modal view to return up a view.

The WebView and MapView should then look similar to those shown overleaf:

Figure 4: The High Scores WebView

Figure 5: The MapView with annotations

That is the whole of the application now built. Add a suitable icon to the .plist file so the application can be identified on the device and give it a test!