We decided to do a turn by turn battleship game. The goal is to defeat all the enemy ships before your ships are defeated. The following describes how we met the project requirements.

Base Requirements

The application must have a fully functional and well-designed foreground user interface.

The app opens up to a beautiful UI. The main screen utilizes a underwater gif of waves that goes with the game's theme. The text is different font, that flows well with the purpose. The title is cursive, and the buttons to click on are are regular text that is positioned appropriately. The app constrains everything (text, buttons, etc) no matter which phone screen size it is playing on.

The application must persist data between user sessions via local storage on the device, or via web services.

The app is integrated with parse, which persist data through user sessions. We store our users in a User table that keeps track of each player's win-loss record. This data is displayed in a leaderboard table view controller, where players can select a cell and play against any user. We also have a Game table in Parse, which keeps track of all of the game data, such as whose turn it is and the coordinates of the ship to be moved. Each row in the game table corresponds to a specific game instance between two users. When a user makes a move, the game table is updated with the data for that move. The opponent is always listening for a change in the record. When it sees that the turn variable in the record has been flipped, it will read the record and apply the changes to its own game view.

The application must be stable and not crash when used. Your final project should include automated test cases that utilize the iOS testing frameworks.

The menu system of our application should be stable, and has been tested by hand. We struggled to figure how to write xctests for parse loads in a short amount of time. The gameview is mostly stable, but it is also unpredictable because of the way we exchange player moves by reading and writing to parse so quickly. It would also be a large task to learn how to write test cases for a view with so many conditions. So, in short, our test cases are lacking.

Networking Integration Requirements

Directly integrate with a data store in the network via web services. This can be an existing data store from another service, or something you have implemented/deployed yourself.

The app utilizes Parse in order to integrate with a data store over the network. We use parse to send data of the last players turn to the current player's phone. Whether the last player attacked or moved a ship, their exact action was recorded via Parse. The user's phone then updates their local data with the information saved on Parse. The enemies ships will move if the other player moved the ship on their turn. If your ship is sunk, the information is obtained over Parse, and you see an animation of your ship sinking.

Network Integration Requirements

Integrate with a popular social media platform using the Social Framework.

We can log in to the app by also logging into facebook. We give the option to log in via parse, or sign up with parse as well. But if the user logs in with facebook, they get the unique username that facebook gives the player through parse (be3zlds0…. instead of wegenerg or ethan1).

App Integration Requirements

Have your app implement an app extension of its own

Our app extension is a version of the Today extension that was implemented in class. It runs a parse query to search for all of the available games to play and tells the user how many games are available to play, as well as tells the user where to go within the app to

start a game. The today extension also has a button that triggers a URL scheme to open the app directly from the extension.

Advanced Platform Integration/Deployment Requirement

Directly integrate an interactive map

We allow the users to move the World around while the camera stays in place. This was the suggested method on multiple blogs, and post. The user can pan the world until and see the entire map.

Non-trivial integration of 3D graphics via OpenGL ES

We use 3D models obtained from the internet, and modify and manipulate how they look to suit our game. We did not use any OpenGL ES to make any of our models. Any editing of our models were made in Blender, and SceneKit. We had to piece the models together to make them move properly, which had to be done with a combination of Blender and Scenekit.

Beyond The Call Of Duty

We added animations, and motions to the game to make the game seem more realistic. Before we only had ships move to the spots they were supposed to. Now the ships rotate, and animate movement when they move. When a ship dies we also have it pivot into a sinking position, and is sinks down below the map. The hardest parts about the strictly the game itself outside of the specified requirements was camera movement and ship movement.

In order to move the camera it could have been as simple as setting the scenes allowsCameraControl to true. Although doing this allowed the player to rotate the view under the map, which is not good functionality. There was literally no documentation on how to get custom camera movement. The only help was suggestions to move the world instead of the camera. We found code that allowed the movement of objects through a UIPanGestureRecognizer. We tried the UIRotateGestureRecognizer movement to restrict rotation only, although we also found that once the allowsCameraControl is set to true the attributes of the camera can not be changed. We had to implement our camera manually. We

tried the UISwipeGestureRecognizer although this did not get us the constant change in transition position when the player moved their finger across the screen. This means that from the point the player touch and moved their finger to a new position, we did not get the local position change from where the finger was the last transition. If the player only moved their finger a distance of 2.34 on x and 4.21 on z, we have to update the world's current position to move by that much also. This was done by using UIPanGestureRecognizer and getting the sender objects translationInView which is the new position position the user's finger is at in comparison to the old position. We had to group everything in the world except the camera into a separate node, and move this node according the movement of the user. This took a lot of trial and error, but gives us the functionality that we want. It is a little buggy, although this has to do with the fact that the phone has a lot to render (texture, lighting, object movements). On every movement of the world, the app needs to re-render all the materials, and with realistic directional lighting, this puts more strain on the phone.

Ship movement was difficult because we actually couldn't find help on how to import an external 3d model into a scenekit view. Through trial and error we found that scenekit can import 3d models as either .obj files or .dea files. The .obj files was the first 3d file type I tried and was good because it kept our 3d models "glued together" so ships could move as one whole object. The problem was .scn files (which is the type of files swift uses to run a 3d game) cannot convert/import a .obj file object into the game. So after some more trial and error I found that .dea files can also be imported into a scene file as an object. The problem with .dea files for us is that if the models had separate pieces, then they would be independent of each other when the entire ship was supposed to move. So the ship would move with the turret still in place. This was fixed by naming each individual piece of the 3d model the same thing inside the .dea file before converting over to the scene view. Then in the scene file each part of the 3d ship would also have to be named the same thing, although named differently from the .dea file itself. This is because when we are using the same file for different ships, so when a user tapped on a ship, we got the texture meshes from the position where the user tapped. These mesh names come from what we named the different parts ship in the .dea file. So if the scene view and the .dea file had the same name for the ships parts, the game thought we were trying to move the mesh inside of the .dea file, not the game object inside the scene view. With our ships originating from the same file, there was no way the game could know which ship we were moving.

Yet another problem with the ships is adding them to a scene view would also rotate them in a weird way. We never found the reason for this. Although it was adding a ship from the same file would also position in a different angle. Therefor their local axis's

were different, and we could not move/rotate them with the same code we used for the movement of other ships. To fix this we gave each ship a unique name, and unique movement where it was needed.

Then the issue of moving the ships on the other person's screen as well. It took a long time to figure out how we wanted to move the ships on the opponents phone as well. We tried experimenting with 2 cameras inside the scene. This way the other user could automatically see what the other player did as he did it. We hoped this would save us a huge amount of time, instead of implementing some kind of update. Unfortunately, we realized that there was no way to connect both players in the same exact game session (it was worth a try). Instead we utilized Parse, not only to persist our data for the app extension like we originally wanted, but also to control the entire functionality of our game.

Parse created its own nightmare although. We tried to use Parse's PFPush() method in order to push notifications to the phone. We wanted to stay away from this as we found out that actual push notifications (that alert the user) can't be achieved unless we had a full developer account. We still tried to get push notifications to the phone which would trigger the game to update, but couldn't find enough information on how to get the other phone to react to the data being pushed to it. Instead we utilized a NSTimer object that constantly queries the data from Parse, and gets the current turn. Originally we had an observer on the player's turn as well to monitor when it switched, but realized that this was redundant as we could just run a comparison on queried turn status.

To sum up the movement aspect, adding motion to objects in 3d space was difficult. There was very little documentation on how to do so. Most documentation about movement in a game was SpritKit, which is 2D and nearly all the code was unique to 2D space alone. In order to get movement to work in the way it does took trial and error.

Missed functionality

Real Time
The observer was going to come in handy if we could databind our local variables to Parse. Because we have to run a new query to update our gameObject variable, there was no way to Databind, and therefore no feasible way to get instant feedback from

parse when a value changes. This stopped us from going real time with our game. We could have had the NSTimer run alot faster, to try to simulate real time. This would have to be done a different thread so the NSTimer is not constantly slowing down or stopping the user from playing the game. On IPhone 5s and above this could be possible as they implement dual-cores, although we wanted the game to be playable on previous versions, and ran out of time to implement functionality if we decided to do so.

chat room
We created an app that used Peer-to-Peer chat. A method to directly connect two users in close proximity to each other. We wanted to use this functionality to be able to find and matchmake people near each other which was our original idea for matchmaking. We found this would be bad as if people walked away, the session would end. After that we never went back to implementing a chat room. When we continue working on the app after the semester, we will implement a chat screen from data obtained on parse.

more than 2 people
This was the last thing on our to do list. As time ran out we scrapped the idea. We almost scrapped the idea to even make it multiplayer with how little time we had left. We were going to make it single player with the game state getting persisted over parse, and simple AI that move, and attack when the player was in a certain distance. It would have been a great boost to the game if we could have friends on the same team. At the very least make the game more strategic.

more map materials
We wanted more materials in the game, although more materials also slows down the game. We will explore multithreading, creating our own low poly 3D models, and look into using light more efficiently than we currently are.

different class ships (different 3D models)
Because of all the problems we had with finding good 3D models for free, we only have 1 class ship. We wanted submarines, carriers, and other ships, although none were free. The battleship was the only military like 3D model we could find.

Music and Sound Effects
We would have liked to add music and sound effects. Although there were other things that were more important. We spent 5 minutes trying to get background music, and decided to save it for later if we get time.

Settings Menu

Contains no functionality. It was designed to handle turning notifications on and off, music on and off toggle, and a volume slider. Without those functions, the menu became useless.

Ally Ship NPC's

A big part of the game we couldn't get implemented in time was allied NPC's. These allies would either follow the player when commanded, or guard an outpost (if we could have more 3D materials without the game lagging). These NPC's could also be generated to attack the Enemy base, somewhat like League of Legends. This will be something we come back to later as well.

Leveling/Currency System

Going into the game we wanted to make a good leveling system, where the user could level up and have certain perks and abilities. The user could also generate currency from their main base and outpost. This currency could be used to buy perks, buff allies, or increase defenses of outpost and main bases. Although this was a very important feature to have for us, it was not a basic necessity to create the game. And with all the drawbacks we had from even getting 3D models into our game, and get them moving, this got pushed back.

There were other things we considered such as Leaderboards, Sea Monsters, and day/night. These are more cosmetics that we mentioned, but we have high ambitions for the game. If we release to the app store, we're going to either buy really good 3D models, or create good low poly models of our own.