Duelist!

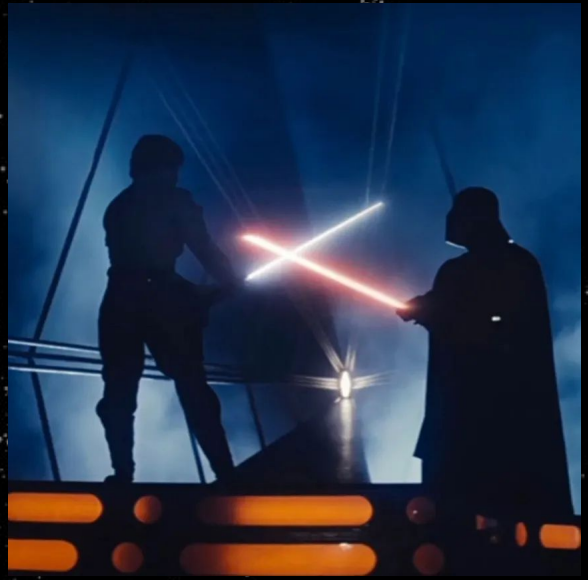
Two Player Sword Fighting Game

Sam
Sebastian
John
Noah

Content

Duelist Ideology

- Fun game to play with friends
- Promoting socializing
- Swords are neat
- The Wii was one of the greatest consoles of all time

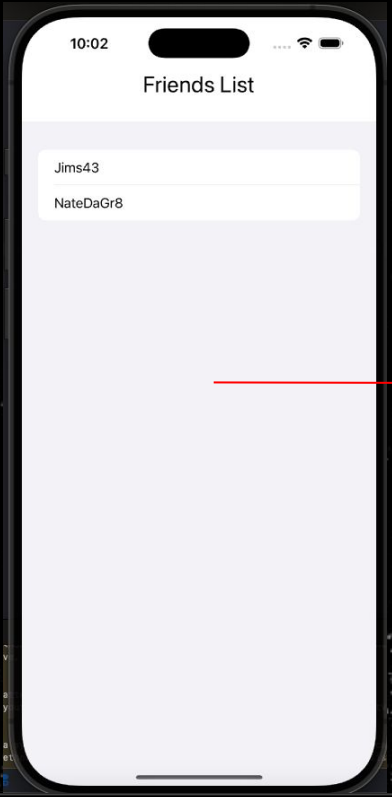


Frameworks

- Primary
  - Core Motion
  - Network
  - Core Audio
  - Multithreading(Combine)
- Secondary
  - Animation
  - Notifications
  - Camera

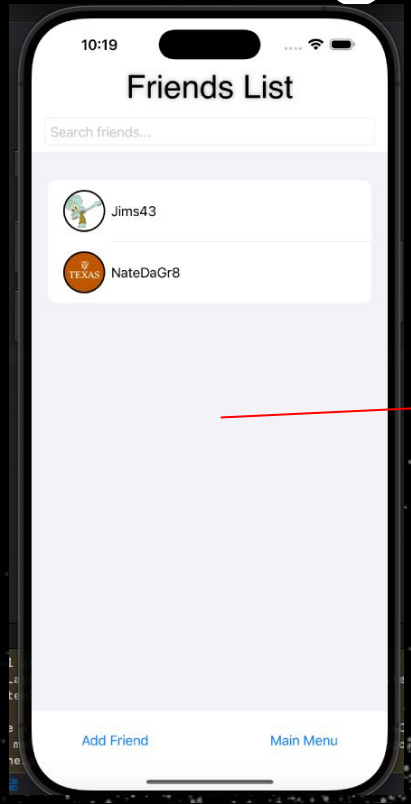
SwiftUI

UI Integration




```
struct FriendsList_V: View {
    @State private var searchText: String = ""
    var body: some View {
        VStack{
            List {
                ForEach(filteredFriends) { friend in
                    Button {
                        print("Friend Selected: \(friend.friendsUserID)")
//                        NavigationHandler.animatePageChange {
//                            nav.currentPage = .otherProfile
//                            //FIXME: Replace with "other profile" and pass the friend selected
//                        }
                    } label: {
                        HStack {
                            Text(String(friend.friendsUserID))
                            Spacer()
                            }
                            .contentShape(Rectangle())
                    }
                    .buttonStyle(PlainButtonStyle())
                }
            }
        }
    }
}

#Preview {
    FriendsList_V()
}
```

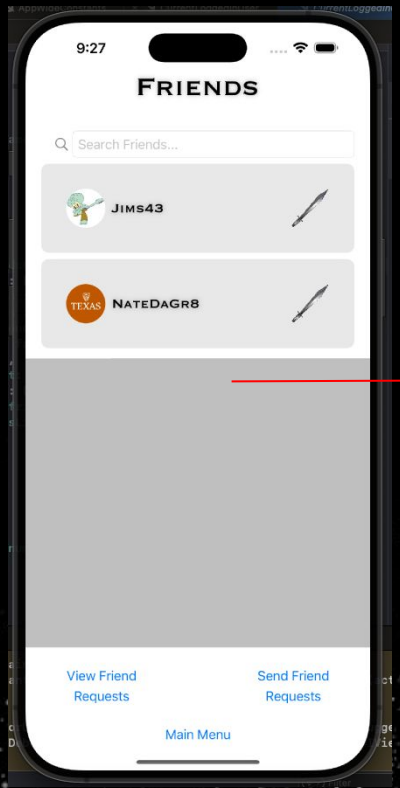
UI Integration




```swift
struct FriendsList_V: View {
    @EnvironmentObject var nav: NavigationHandler
    @EnvironmentObject var userManager: CurrentUserManager

    @State private var searchText: String = ""

    var filteredFriends: [Friend] {
        if searchText.isEmpty {
            return userManager.currentUser.friendsList
        } else {
            return userManager.currentUser.friendsList.filter {
                $0.friendsUserID.lowercased().contains(searchText.lowercased())
            }
        }
    }
    var body: some View {
        VStack{
            D_Label(title: "Friends List", fontSize: Globals.LargeTitleFontSize)
            TextField("Search friends...", text: $searchText)
                        .padding(.horizontal)
                        .textFieldStyle(RoundedBorderTextFieldStyle())

            List {
                ForEach(filteredFriends) { friend in
                    Button {
                        print("Friend Selected: \(friend.friendsUserID)")
                        NavigationHandler.animatePageChange {
                            nav.currentPage = .otherProfile(friend: friend)
                        }
                    } label: {
                        HStack {
                            Image(friend.image)
                                .resizable()
                                .aspectRatio(contentMode: .fit)
                                .frame(width: 50, height: 50)
                                .clipShape(Circle())
                                .overlay(Circle().stroke(style: StrokeStyle(lineWidth: 2)))
                            Text(String(friend.friendsUserID))
                            Spacer()
                            }
                            .contentShape(Rectangle())
                    }
                    .buttonStyle(PlainButtonStyle())
                }
            }

            HStack(spacing: Globals.LargeHSpacing){
                Button("Add Friend"){
//                      NavigationHandler.animatePageChange {
//                          nav.currentPage = .addFriends
//                      }
                }
                Button("Main Menu"){
                    NavigationHandler.animatePageChange {
                        nav.currentPage = .mainMenu
                    }
                }

            }
            .padding(Globals.SmallHPadding)
        }
    }
```

```swift
import SwiftUI

struct D_Label: View {
    var title: String
    var fontSize: CGFloat

    var body: some View {
        Text(title)
            .foregroundStyle(.black)
            .font(.custom("Default", size: fontSize))
            .foregroundColor(.yellow)
            .shadow(radius: 3)
    }
}

#Preview {
    D_Label(title: "Title", fontSize: 24)
        .font(.title)
}
```

UI Integration



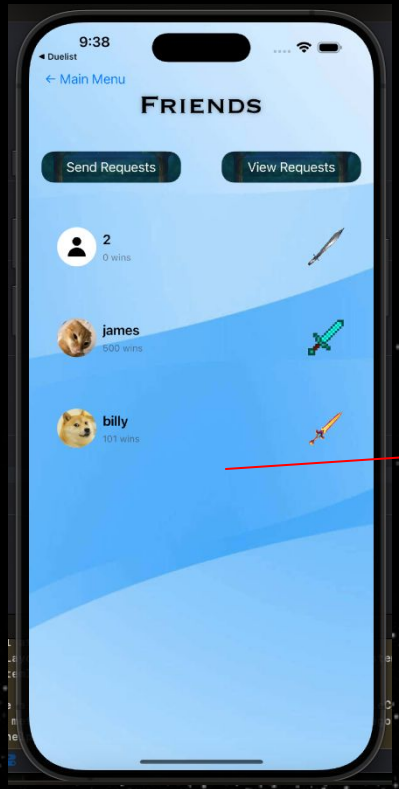

```swift
struct FriendsList_V: View {
    @EnvironmentObject var nav: NavigationHandler
    @EnvironmentObject var usesManager: CurrentUserManager

    @State private var searchText: String = ""

    var filteredFriends: [Friend] {
        if searchText.isEmpty {
            return usesManager.currentUser.friendsList
        } else {
            return usesManager.currentUser.friendsList.filter {
                $0.friendsUserID.lowercased().contains(searchText.lowercased())
            }
        }
    }
    var body: some View {
        VStack{
            D_Label(title: "Friends", fontSize: Globals.LargeTitleFontSize)

            D_List {
                VStack{
                    D_TextField(text: $searchText, type: .search, keyword: "Friends")

                    ForEach(filteredFriends, id: \.id) { friend in
                        D_ListRow {
                            Button {
                                NavigationHandler.animatePageChange {
                                    nav.currentPage = .otherProfile(friend: friend)
                                }
                            } label: {
                                HStack {
                                    ProfilePhotoTemplate(size: .small, image: friend.image)
                                    D_Label(title: friend.friendsUserID, fontSize: Globals.HeadingFontSize)
                                        .foregroundColor(.black)

                                    Spacer()
                                    SwordPhotoTemplate(image: friend.sword)
                                }
                                .contentShape(Rectangle())
                            }
                            .buttonStyle(BorderlessButtonStyle())
                            .padding()

                        }
                    }
                }
            }
            HStack(spacing: Globals.LargeHSpacing){
                Button("View Friend Requests"){
                    NavigationHandler.animatePageChange {
                        nav.currentPage = .viewFriendRequests
                    }
                }
                Button("Send Friend Requests"){
                    NavigationHandler.animatePageChange {
                        nav.currentPage = .sendFriendRequests
                    }
                }

            }
            .padding(Globals.SmallHPadding)
            Button("Main Menu"){
                NavigationHandler.animatePageChange {
                    nav.currentPage = .mainMenu
                }
            }
        }
    }
}
```

```swift
import SwiftUI

struct D_Label: View {
    var title: String
    var fontSize: CGFloat

    var body: some View {
        Text(title)
            .foregroundStyle(.black)
            .font(.custom("Copperplate", size: fontSize))
            .foregroundColor(.yellow)
            .shadow(radius: 3)
    }
}
```

```swift
struct D_List<Content: View>: View {
    let content: Content

    init(@ViewBuilder content: () -> Content) {
        self.content = content()
    }

    var body: some View {
        List {
            content
        }
        .listStyle(.plain) // or whatever style you prefer
        .listRowInsets(EdgeInsets(top: 10, leading: 20, bottom: 10, trailing: 20))
        //.scrollContentBackground(.hidden) // iOS 16+
        .background(Color.black.opacity(0.25))
    }
}
```

```swift
enum TextFieldType {
    case normal
    case secure
    case search
}

struct D_TextField: View {
    @Binding var text: String
    var type: TextFieldType
    var keyword: String

    var body: some View {
        switch type{
        case .normal:
            TextField(
                "",
                text: $text,
                prompt: Text(keyword)
            )
            .textFieldStyle(RoundedBorderTextFieldStyle())
            .padding(Globals.SmallHPadding)

        case .secure:
            SecureField(
                "",
                text: $text,
                prompt: Text(keyword)
            )
            .textFieldStyle(RoundedBorderTextFieldStyle())
            .padding(Globals.SmallHPadding)

        case .search:
            TextField(
                "",
                text: $text,
                prompt: Text("Search \(keyword)...")
            )
            .textFieldStyle(RoundedBorderTextFieldStyle())
            .padding(.leading, 40)
            .overlay(
                HStack {
                    Image(systemName: "magnifyingglass")
                        .foregroundColor(.gray)
                        .padding(.leading)
                    Spacer()
                }
            )
        }

    }
}
```

UI Integration




```swift
struct FriendsList_V: View {
    @EnvironmentObject var nav: NavigationHandler
    @EnvironmentObject var authManager: AuthManager

    @State private var friends: [User] = []
    @State private var isLoading = false
    @State private var showError = false
    @State private var errorMessage = ""

    var body: some View {
        D_Background {
            BackButton(label: "Main Menu", destination: .mainMenu) {
                VStack {
                    D_Label(title: "Friends", fontSize: Globals.LargeTitleFontSize)

                    // Navigation buttons to friend requests
                    HStack(spacing: 20) {
                        D_Button(action: {
                            nav.currentPage = .sendFriendRequests
                        }) {
                            Text("Send Requests")
                        }

                        D_Button(action: {
                            nav.currentPage = .friendRequests
                        }) {
                            Text("View Requests")
                        }
                    }
                    .padding(.vertical)

                    if isLoading {
                        ProgressView("Loading friends...")
                            .frame(maxWidth: .infinity, maxHeight: .infinity)
                    } else if friends.isEmpty {
                        VStack {
                            Text("No friends yet")
                                .font(.title2)
                                .foregroundColor(.secondary)
                            Text("Send friend requests to connect with other players!")
                                .font(.body)
                                .foregroundColor(.secondary)
                                .multilineTextAlignment(.center)
                        }
                        .frame(maxWidth: .infinity, maxHeight: .infinity)
                    } else {
                        D_List {
                            ForEach(friends, id: \.id) { friend in
                                D_ListRow {
                                    FriendListRow(friend: friend)
                                }
                            }
                        }
                    }

                    Spacer()
                }
            }
            .padding(.top, 10)
        }
        .task {
            await loadFriends()
        }
        .alert("Error", isPresented: $showError) {
            Button("OK") { }
        } message: {
            Text(errorMessage)
        }
    }
```

```swift
struct D_Background<Content: View>: View {
    @EnvironmentObject var nav: NavigationHandler
    let content: () -> Content

    var body: some View {
        GeometryReader { geo in
            ZStack {
                Image("background_default")
                    .resizable()
                    .scaledToFill()
                    .frame(width: geo.size.width, height: geo.size.height)
                    .clipped()
                    .ignoresSafeArea() // ensures it fills under notch/safe
                content()
                    .frame(width: geo.size.width, height: geo.size.height)
            }

        }
        .ignoresSafeArea()
    }
}
```

```swift
struct D_Butt
    @GestureS
    var actio
    var label

    var body:
        Butto
            Z
```

```swift
struct D_List<Content: View>: View {
    let content: Content

    init(@ViewBuilder content: () -> Content) {
        self.content = content()
    }

    var body: some View {
        List {
            content
        }
        .listStyle(.plain) // or whatever style you prefer
        .listRowInsets(EdgeInsets(top: 10, leading: 20, bottom: 10, trailing: 0))
        .scrollContentBackground(.hidden) // iOS 16+
        .background(Color.clear)
    }
}
------,
    .foregroundStyle(.white)
    .padding(.horizontal, Globals.SmallH
    .padding(.vertical, Globals.SmallVP
    .background(
        RoundedRectangle(cornerRadius: G
            .fill(isPressed ? ThemeColor
            .padding(.vertical, Globals.
        )
    }
    .contentShape(Rectangle())  //Defines the ta
    .overlay(     // This gives a visual feedbac
        RoundedRectangle(cornerRadius: Globals.C
            .stroke(isPressed ? ThemeColors.acce
    )
    .simultaneousGesture(
        DragGesture(minimumDistance: 0)
            .updating($isPressed) { _, gestureSt
                gestureState = true
            }
    )
}
.buttonStyle(.plain)
.padding(.horizontal)
}
}
```

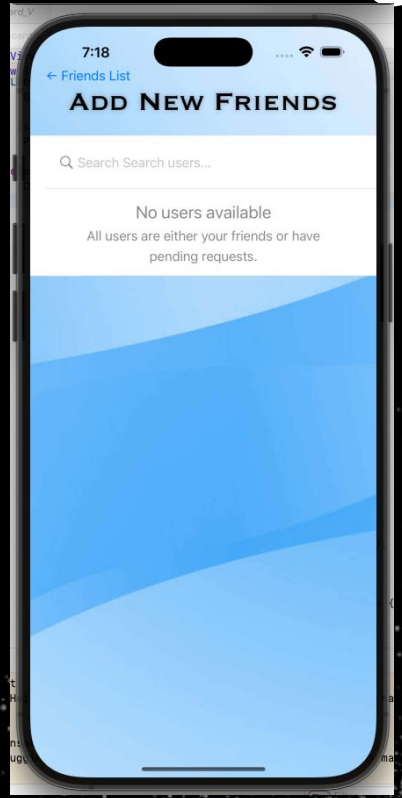
```swift
struct D_Label: View {
    var title: String
    var fontSize: CGFloat

    var body: some View {
        Text(title)
            .foregroundStyle(.black)
            .font(.custom("Copperplate", size: fontSize))
            .foregroundColor(.yellow)
            .shadow(radius: 3)
    }
}
```

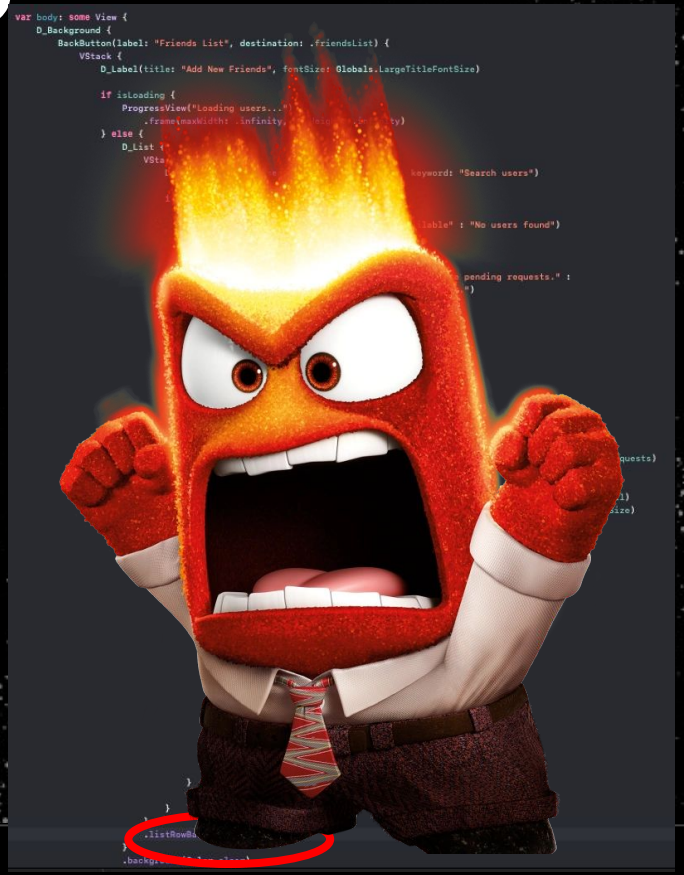
```swift
// Helper view for displaying individual friends
struct FriendListRow: View {
```

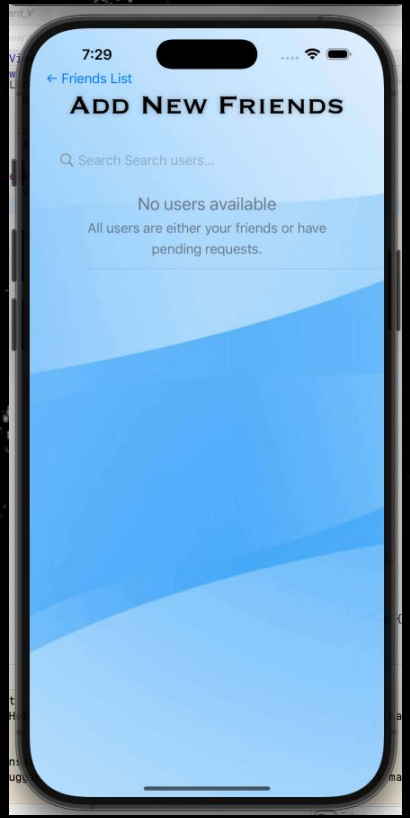
```
vigationHandler
erProfile(user: friend, source: .friendsList)
.getProfileImageView(for: friend, size: .small)
.leading) {
ername)
dline)
dColor(.primary)
```

```swift
struct D_ListRow<Content: View>: View {
    let content: Content

    init(@ViewBuilder content: () -> Content) {
        self.content = content()
    }

    var body: some View {
        content
            .font(.headline)
            .padding()
            .frame(maxWidth: .infinity, alignment: .leading)
            .scrollContentBackground(.hidden)
            .background(Color.clear)
            .cornerRadius(10)
            .listRowSeparator(.hidden)
            .listRowBackground(Color.clear)
    }
}
```

UI Integration Challenges





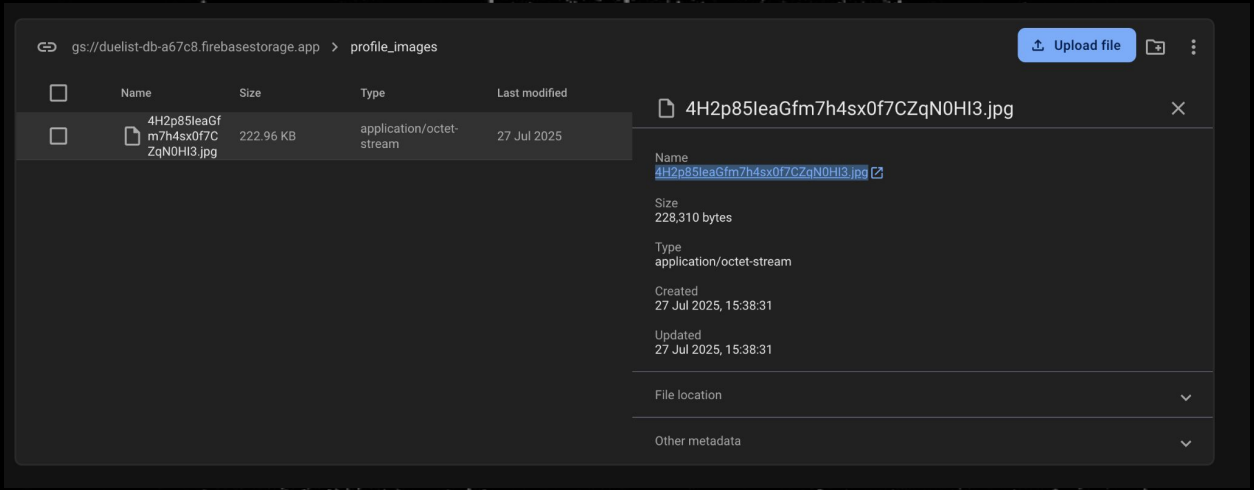


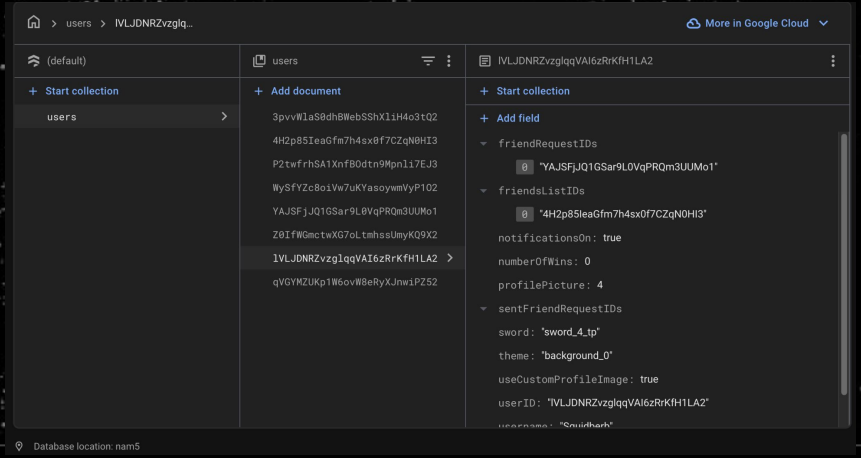
Theming

- Lots of hidden work
- Adding background in views was involved
- Every theme added adds aesthetic inconsistencies
- IOS Device UI mode complications
- Custom components were retooled to support custom themes
- Current implementation needs work for extensibility + compatibility

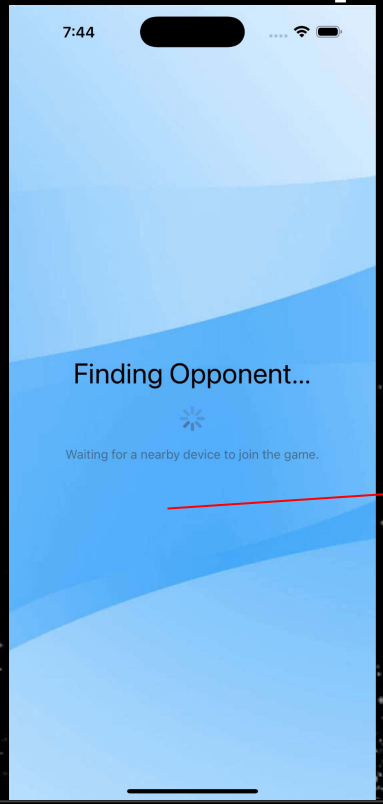
Database

- Learning experience
- Firebase Auth + Firestore + Firestorage for custom profile pictures
- Swift UI complicates it
- Swapping out our stopgap tooling can sometimes involve nearly rewriting a view
- "Singleton" pattern + @published vars
- Making changes to schemas can involve headache





Gameplay




```swift
VStack(spacing: 24) {
    Text("Finding Opponent...")
        .font(.largeTitle)
        .padding(.top)

    ProgressView()
        .scaleEffect(1.5)

    Text("Waiting for a nearby device to join the game.")
        .font(.subheadline)
        .multilineTextAlignment(.center)
        .foregroundColor(.secondary)
}
.padding()
}
.onReceive(viewModel.$isConnected) { connected in
    if connected {
        showGame = true
    }
}
.fullScreenCover(isPresented: $showGame) {
    gameplayContent
}
```

```swift
func send(gameState: GameState) {
    guard !session.connectedPeers.isEmpty,
          let data = try? JSONEncoder().encode(gameState) else { return }

    try? session.send(data, toPeers: session.connectedPeers, with: .reliable)
}
```

```swift
enum MultipeerRole {
    case advertiser
    case browser
}

static func assignRole(from id: String) -> MultipeerRole {
    return id.hash % 2 == 0 ? .advertiser : .browser
}
```

```swift
private var browser: MCNearbyServiceBrowser?

private let serviceType = "duelist"
private let myPeerID = MCPeerID(displayName: UIDevice.current.name)

private lazy var session = MCSession(peer: myPeerID, securityIdentity: nil, encryptionPreference: .required)
private lazy var advertiser = MCNearbyServiceAdvertiser(peer: myPeerID, discoveryInfo: nil, serviceType: serviceType)
```

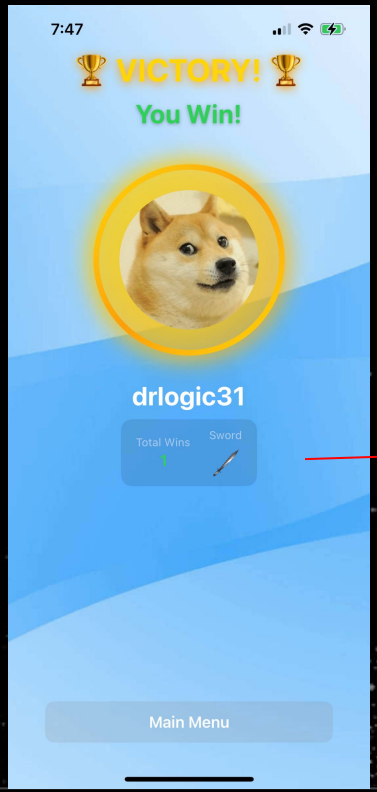
Gameplay


```swift
let currentAction = classifyMotion(acceleration: accelerationMagnitude, jerk: jerk, yawDelta: yawDelta)
//print("xAccel: \(xAccel), yAccel: \(yAccel), zAccel: \(zAccel)")
actualAction.append(currentAction)

if(actualAction.count == 3){
    let newAction = classifyAction(actualAction)
    deviceMotionData.action = newAction
    actualAction.removeAll()

    if newAction != previousAction {
        print("Delagating properly? : \(String(describing: delegate))")
        print("New Actino: \(newAction)")
        previousAction = newAction
        delegate?.handleLocalAction(newAction)
    }
}
```

```swift
func classifyMotion(acceleration: Double, jerk: Double, yawDelta: Double) -> Action {
    if acceleration > 0.7 && jerk < 1.5 {
        return .attack
    }
    if isShakeMotion(yawHistory) && jerk > 2.0 {
        return .block
    }
    return .idle
}

func classifyAction(_ actions: [Action]) -> Action {
    var counts: [Action: Int] = [:]
    for action in actions {
        counts[action, default: 0] += 1
    }
    return counts.max(by: { $0.value < $1.value })!.key
}

func isShakeMotion(_ history: [Double]) -> Bool {
    guard history.count >= 6 else { return false }

    var changes = 0
    for i in 1..<history.count - 1 {
        let delta1 = history[i] - history[i - 1]
        let delta2 = history[i + 1] - history[i]
        if delta1 * delta2 < 0 { // direction changed
            changes += 1
        }
    }
    return changes >= 3
}
```

```swift
let currentTime = CACurrentMediaTime()
let deltaTime = currentTime - self.lastUpdateTime
self.lastUpdateTime = currentTime

//Acceleration and magnitude in G's
let acc = deviceMotion.userAcceleration
  let xAccel = acc.x
  let yAccel = acc.y
  let zAccel = acc.z
let accelerationMagnitude = sqrt(acc.x * acc.x + acc.y * acc.y + acc.z * acc.z)

let jerk = abs(accelerationMagnitude - self.lastAccelleration) / max(deltaTime, 0.001)
self.lastAccelleration = accelerationMagnitude

//let yaw = deviceMotion.attitude.yaw
let yaw = deviceMotion.attitude.yaw
let yawDelta = abs(yaw - self.lastyaw)
self.lastyaw = yaw

//shake dog
yawHistory.append(degrees(radians: yaw))
if yawHistory.count > maxYawHistory {
    yawHistory.removeFirst()
}

let rawAngle = degrees(radians: yaw)
let normalizedAngle = normalizeAngle(rawAngle)
let smoothedAngle = smoothAngle(normalizedAngle)
```

Gameplay

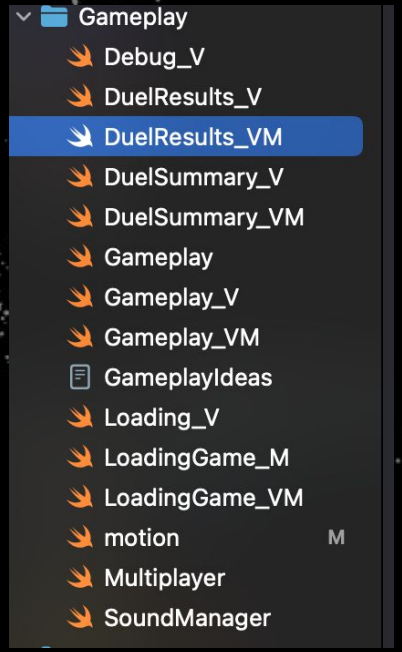



```swift
init(winnerName: String, currentUser: User?, opponentUser: User?, authManager: AuthManager) {
    self.winnerName = winnerName
    self.authManager = authManager
    self.currentUserWon = (winnerName == "You")

    if currentUserWon {
        self.winnerUser = currentUser
        self.loserUser = opponentUser
    } else {
        self.winnerUser = opponentUser
        self.loserUser = currentUser
    }
}

func updateWinCount() {
    guard let winner = winnerUser else { return }

    Task {
        do {
            try await FirebaseService.shared.updateUserWins(winner.userID, winner.numberOfWins + 1)

            if currentUserWon {
                await MainActor.run {
                    authManager.user?.numberOfWins += 1
                }
            }
        } catch {
            print("Error updating win count: \(error)")
        }
    }
}
```

Gameplay Skill Issues (Challenges)

- A lot of moving parts
- Keeping track of states and properly passing them through
- Smoothing
- Detection
- Integrating with the V-VM-M system
- Connecting devices
- Allowing for connections in Info.plist

```swift
private var lastEvaluatedPair: (Action, Action) = (.idle, .idle)
@Published var winner: String? = nil
@Published var localHealth = 30
@Published var opponentHealth = 30
@Published var isBlocking = false
@Published var swordAngle: Double = 0
@Published var opponentAction: Action = .idle {
    didSet {
        evaluateDamage() // Recalculate when opponent acts
    }
}
@Published var myAction: Action = .idle {
    didSet {
        isBlocking = (myAction == .block)
        evaluateDamage() // Recalculate when I act
    }
}
@Published var opponent: String = "" // Add opponent data
```



Demo!

Thanks!

Questions?