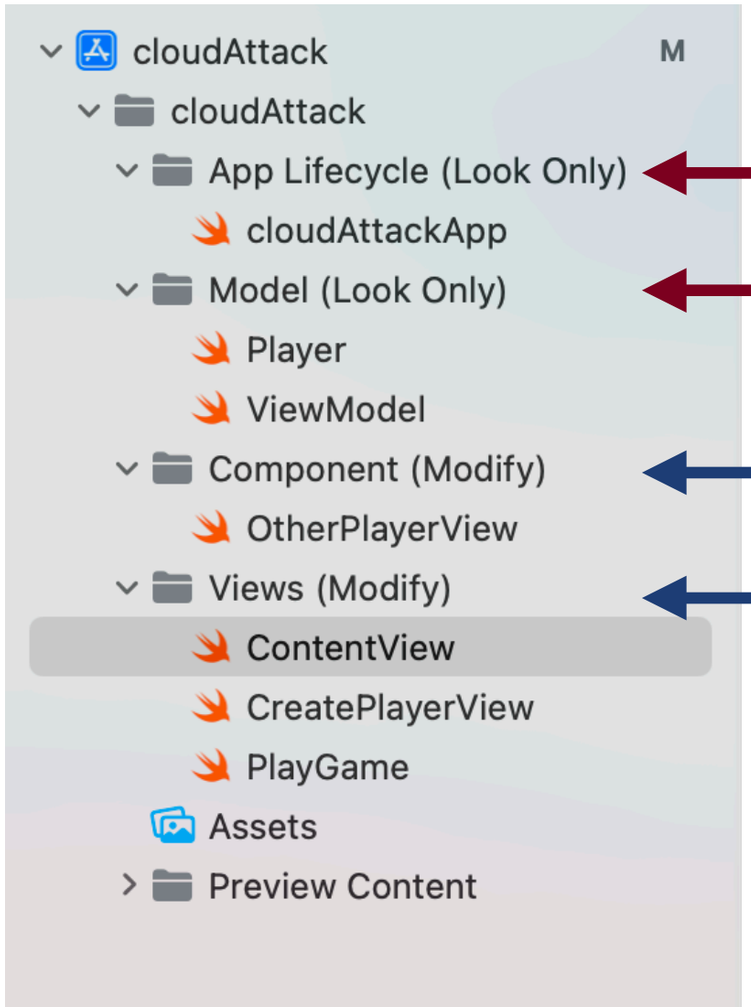
Cloud Attack

Folder Structure



This is the first piece of code that runs. No need to modify it.

The Structure of our data is in Player.swift. The ViewModel lets us communicate with the server to create a user, attack someone, heal our wounds, or get the status of the game.

This component shows the stats for the opponents you want to attack.

This contains three pages that the user sees in our app.
ContentView determines whether we see the login screen or the gameplay screen

Create Player View shows a form to let us join the server.

PlayGame let us see a list of users to attack and gives us an option to heal ourselves

Content View

```
import SwiftUI

struct ContentView: View {
    //This will choose which view to show the user
    //based on if they have joined the server
    //or not. Try changing it from false to true
    //and then back again.
    @State var userIsIn:Bool = false

    //You will modify this to show the proper
    //Views.
    @ViewBuilder
    var body: some View {
        if userIsIn {
            Text("Play Game")
        }else{
            Text("Create a User")
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

userIsIn is a bool which can be true or false. **Change it from false to true** and look at the result. It should show Play Game instead of Create a User. **Change it back to false** before the next step.

This is known as a conditional. When userIsIn is true, it will show text that says “Play Game”. The else runs when userIsIn is false and it will show text that says “Create a User”

Content View

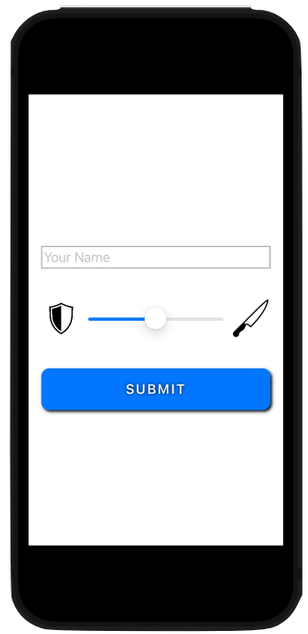
```swift
import SwiftUI

struct ContentView: View {
    //This will choose which view to show the user
    //based on if they have joined the server
    //or not. Try changing it from false to true
    //and then back again.
    @State var userIsIn:Bool = false

    //You will modify this to show the proper
    //Views.
    @ViewBuilder
    var body: some View {
        if userIsIn {
            PlayGame()
        }else{
            CreatePlayerView(playerCreated: $userIsIn)
                .padding()
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Take out the Text elements that were there and put in our **PlayGame** view and our **CreatePlayerView** view.

Note the use of .padding() to give some space around the form.

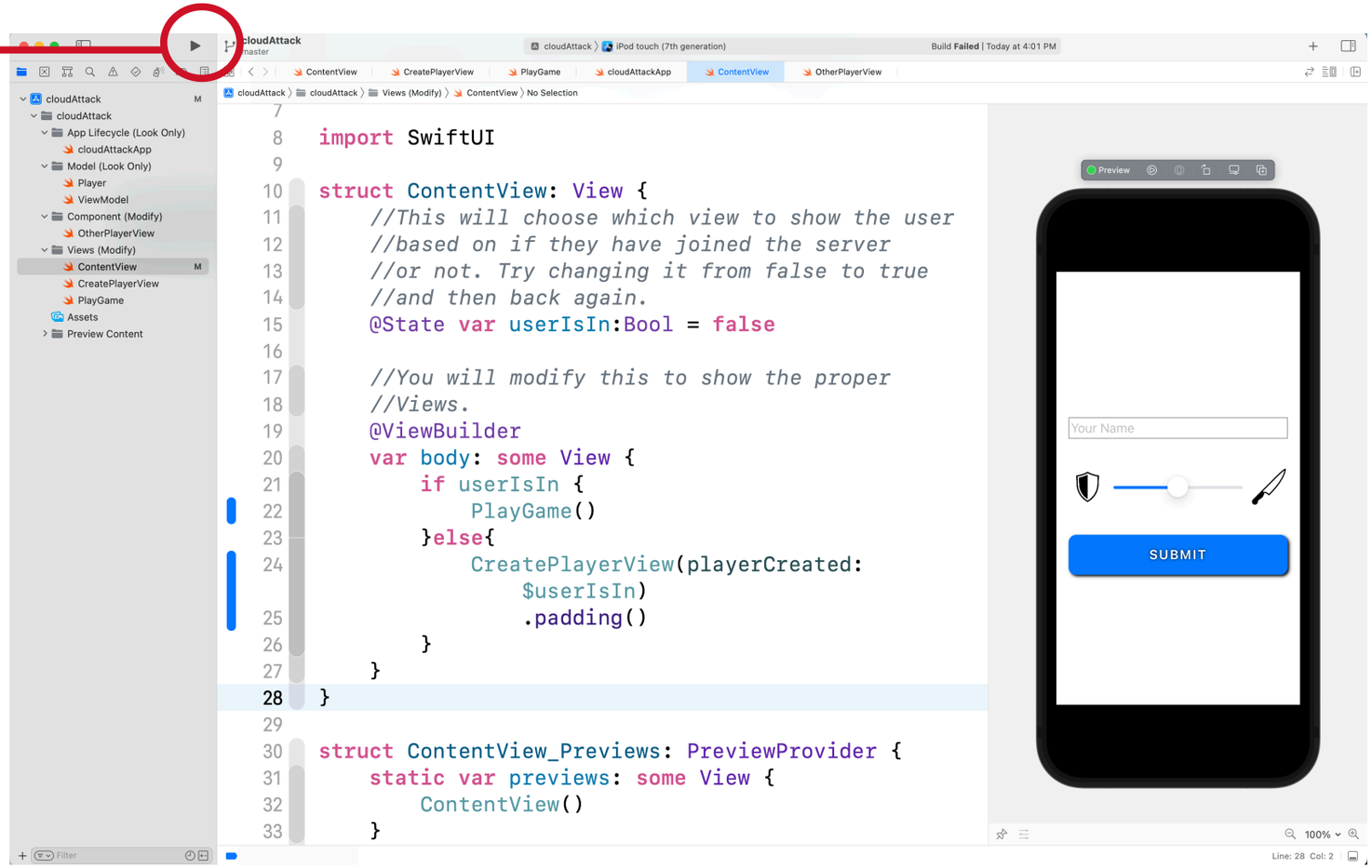
Also note that we are passing userIsIn to the create user view. We can update it there and automatically show the PlayGame view once we are logged in the server.

Completed View



Content View

Run the App. Notice how the slider works and you can enter text. Unfortunately tapping the submit button doesn't do anything. We will fix that in the CreatePlayerView File next.



CreatePlayerView

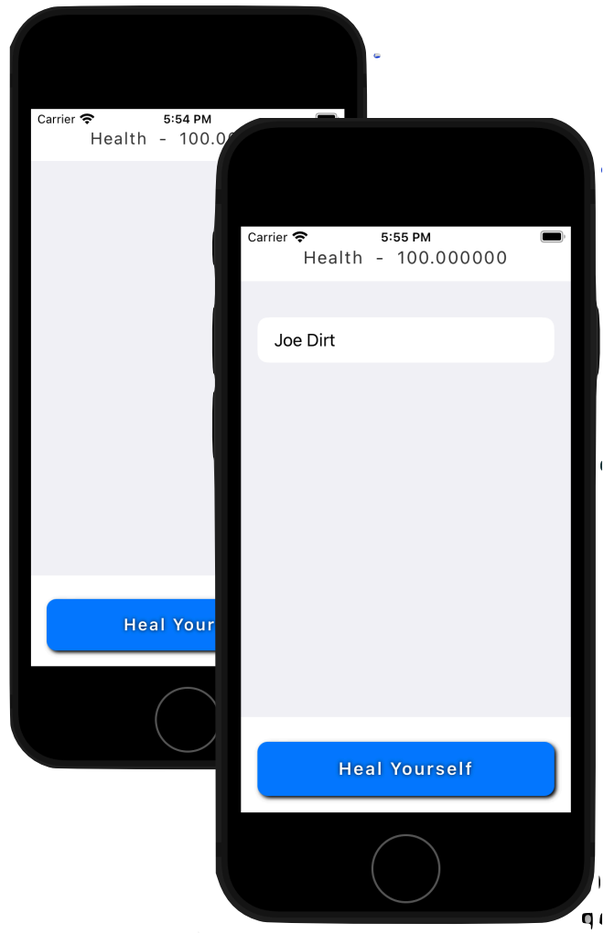
```
    Image("Knife")
        .resizable()
        .frame(width: 50, height: 50, alignment:
            .center)
}
.padding(.bottom, 30)

Button {
    //---------------------------------
    //Add Your Code Here
    ViewModel.shared.addUser(userName: userName,
        attackLevel: Int(attackAmount))
    playerCreated = true


    //End of editing code area
    //---------------------------------
} label: {
    Text("SUBMIT")
        .bold()
        .kerning(2)
        .shadow(color: .black, radius: 2, x: 0, y:
            0)
        .padding()
        .frame(maxWidth: .infinity)
```

Scroll down to the Button code area. Add the following 2 lines of code.

The first line accesses our **ViewModel** and makes a call to the server to add a user with our **userName** which is linked to our textField and the **attackAmount** which is linked to our Slider.

Note that we surround attackAmount with the word Int. This is because our function takes a whole number (Int) and our slider gives us a decimal number (Double)

It will look similar to the images below when you run your code and login.



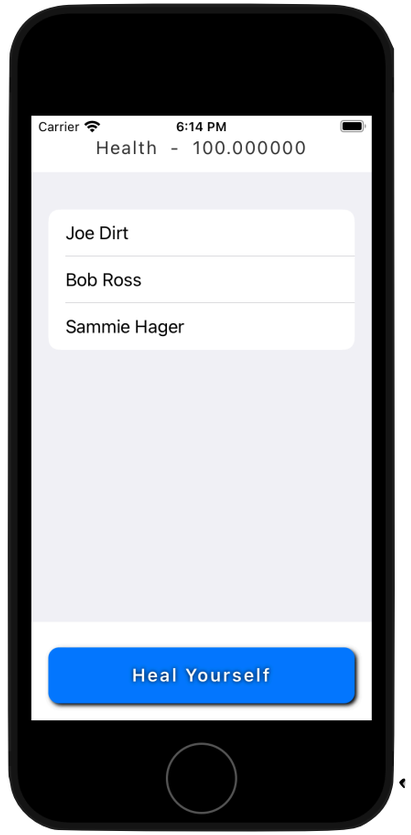
PlayGame

```
    if(player.id != model.us?.id){
     OtherPlayerView(otherPlayer:
         player, ourID: id)
    }
}

Button {
    //----------------------------
    //Add Your Code Here

    model.heal(yourID: model.us!.id)

    //End of editing code area
    //----------------------------
} label: {
    Text("Heal Yourself")
        .bold()
        .kerning(2)
        .shadow(color: .black, radius:
            2, x: 0, y: 0)
```

Scroll down to the Button code section and add the line of code that tells our **model** to contact the server and **heal** this id.

Our id is stored in the **us** variable that is in the **model** variable.

model -> us -> id



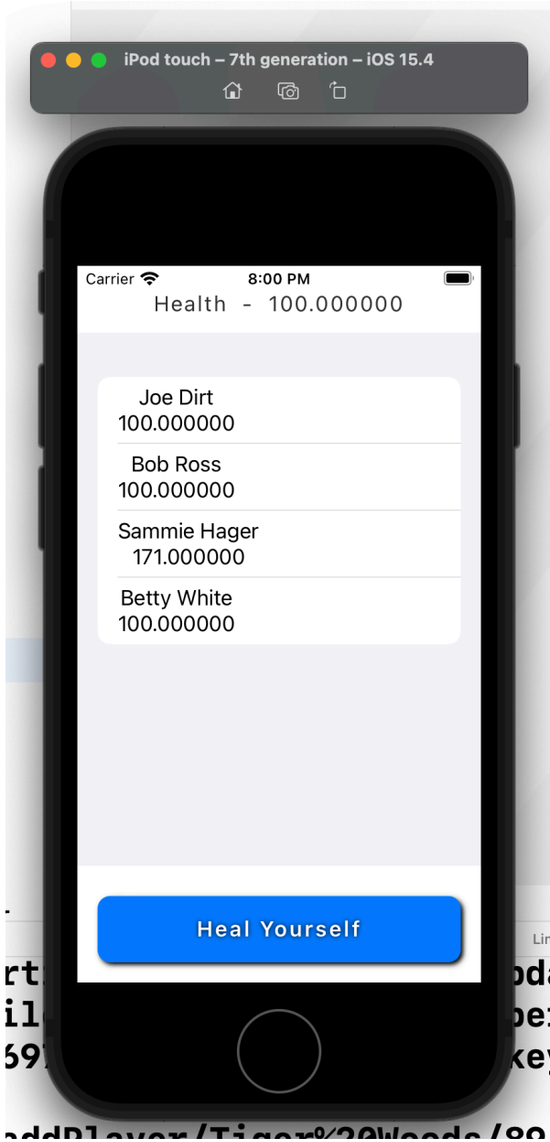
Run the app and press the Heal Yourself button. Notice how you gain health at the top each time you press the button.

OtherPlayerView

```swift
import SwiftUI

struct OtherPlayerView: View {
    //Variable area, do not modify.
    //This stores all the data of the player
    var otherPlayer:Player
    //This stores our ID, it will get set when
    //the list creates this view.
    var ourID:Int = -1


    var body: some View {
        VStack {
            Text(otherPlayer.name)
            Text("\(otherPlayer.health)")
        }

    }
}
```

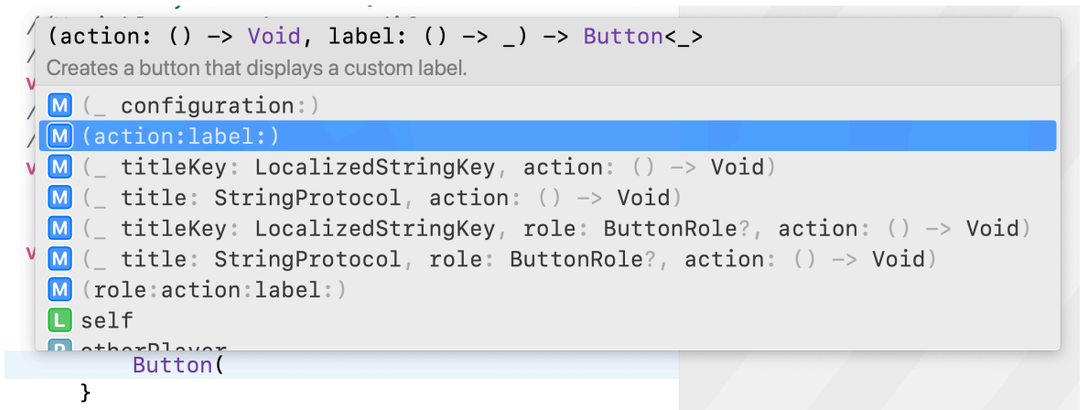
Add another Text label. Note that it takes a string but it has a \( ) in it. This is an area to put the value of a variable. In this case we are putting a **Double** (Decimal Number) in this spot.

Pause after you type **otherPlayer.** You will see a popup with all the items you can access from the other player.



OtherPlayerView

1. Add a Button below the Text. Choose the one highlighted in blue.



2. With the area shown below still highlighted, press return.

```
    Text("\(otherPlayer.health)")
    Button(action: () -> Void, label: () ->
        _)
}
```

```
Button {
    code
} label: {
    code
}
```

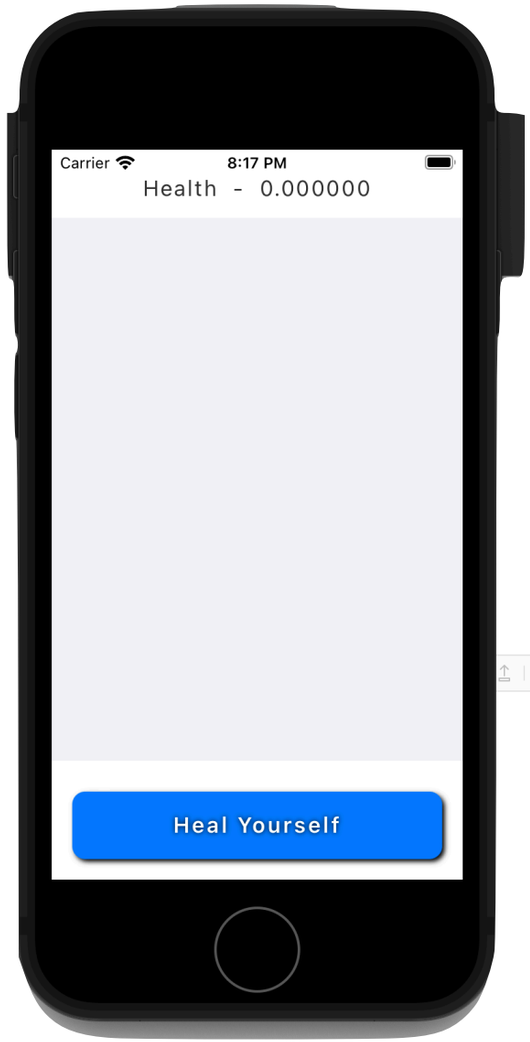
Code to run when the button is clicked.

The Design of the clickable area. It can be images or text or any other view.

Your completed code

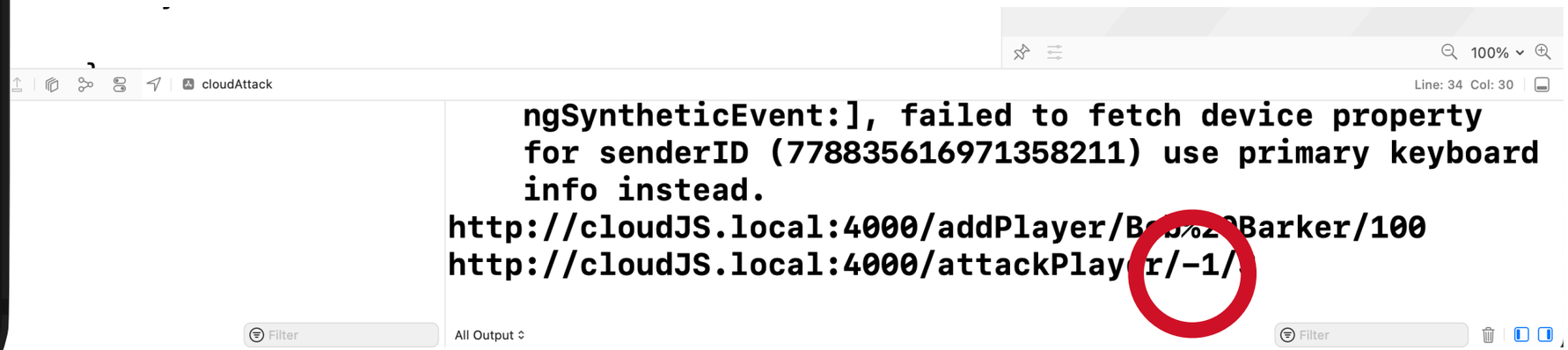
```
VStack {
    Text(otherPlayer.name)
    Text("\(otherPlayer.health)")
    Button {
        ViewModel.shared.attack(yourID:
            ourID, theirID: otherPlayer.id)
    } label: {
        Text("Attack")
    }

}
```

Error!!



When you run the app and try to attack someone, the entire group runs away! I guess that's a win.

The solution is in the console below. Notice that our ID was never set so our server got confused.



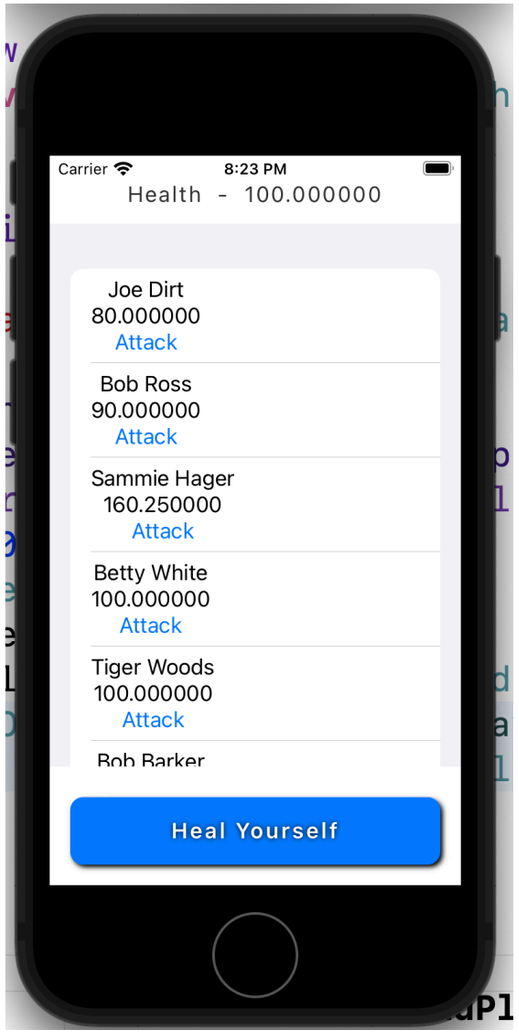
PlayGame

```
struct PlayGame: View {
    @State var id:Int = -1
    @ObservedObject var model = ViewModel.shared


    var body: some View {
        VStack {
            Text("Health  -  \(model.us?.health ??
                0)")
                .kerning(1.5)
                .foregroundColor(Color(.displayP3,
                    red: 0.2, green: 0.2, blue:
                    0.2, opacity: 1))
            List(model.players){
                player in
                if(player.id != model.us?.id){
                 OtherPlayerView(otherPlayer:
                     player, ourID: id)
                }
            }

            Button {
```

This was from a previous build and is no longer used. Remove this line of code.

Replace this id with **model.us!.id**

Ready to Play



You can attack people and people can attack you.

All is good!