Article

Exchanging data between players in real-time games

Send data between players in a real-time multiplayer game.

Overview

In a multiplayer game, you can exchange game data between players to synchronize the state of the game or to facilitate communication between players. You can begin exchanging data when two or more players join or after all the player slots fill when you start the game. For example, if your game provides text messaging, players can send messages to each other while they wait for other players to join the game. GameKit handles the low-level communication of generic data; you design the structure of your game data, and implement the interface for presenting or entering that data.

Start exchanging data

When enough players join a multiplayer game, you can begin exchanging data. For example, present an interface for players to send text messages.

If you use the `GKMatchmakerViewController` class to find players, you can start the game and begin exchanging data when GameKit calls the `matchmakerViewController(_:didFind:)` delegate method.

To begin exchanging data before the game starts, implement the `match(_:player:didChange:)` delegate method. Use the match `expectedPlayerCount` property in the method to determine when you can begin exchanging data. If two or more players join the game, you can begin exchanging data, or if the expected player count is `0`, you can start the game and then exchange data. GameKit also invokes this method when players disconnect or leave the game.

```
// Handle an unknown, connected, or disconnected player state.
func match(_ match: GKMatch,
             player: GKPlayer,
             didChange state: GKPlayerConnectionState) {

    print("Available player slots: \(String(match.expectedPlayerCount))")

    switch state {
        case .connected:
            // Handle connected state.
        case .disconnected:
            // Handle disconnected state.
        default:
            // Handle unknown state.
    }
}
```

To reinvite players whose match state becomes disconnected in a two-player match, implement the `match(_:shouldReinviteDisconnectedPlayer:)` delegate method to return `true`.

Send data to players

GameKit exchanges generic data between players to give you flexibility in designing your game data. You just need to convert your model objects to a `Data` object before you send it. Then, you choose the type of transmission and size of the data depending on the characteristics of your game. You can send data to a subset of players or all players in the game.

Use the `send(_:to:dataMode:)` or `sendData(toAllPlayers:with:)` method to send data to one or more players. To guarantee that players receive the data in the order you send it, pass `GKMatch.SendDataMode.reliable` as the data mode. If the data is small and a delay in transmission invalidates its contents (such as position and velocity in a real-time game), pass `GKMatch.SendDataMode.unreliable` as the data mode.

The following example shows how to convert a string to a `Data` object and send it as a message from one player to another player.

```
// Send a text message from one player to another.
func sendMessage(content: String){
    do {
        // Send the game data to the other player.
        let data: Data? = content.data(using: .utf8)
        try singlesMatch?.sendData(toAllPlayers: data!, with: GKMatch.SendDataMode.u
```

```
    } catch {
        return
    }
}
```

For large data or many players, you can layer a network topology that performs better over the peer-to-peer topology that GameKit provides. For example, reduce the network traffic by implementing a client-server or ring topology. If you send data only to and from one player, you can reduce the network connections by half.

If you choose a client-server topology, use the `chooseBestHostingPlayer(completionHandler:)` delegate method to find the player with the best network connection to act as the server. Invoke this method simultaneously in each game instance running on each player's device.

Receive data from players

Finally, you process the data that players receive from other players. For the recipients of the data, implement the `match(_:didReceive:fromRemotePlayer:)` delegate method to unpack the data sent by other players. For example, if the data contains a text message from another player, convert the `Data` object to a string. To display which player sent the data, use the player parameter.

```
// Receive a message sent from one player to another.
func match(_ match: GKMatch, didReceive data: Data, fromRemotePlayer player: GKPlaye
    let content = String(decoding: data, as: UTF8.self)
    // Update the interface from the game data.
}
```

See Also

Real-time games

- Creating real-time games
  Develop games where multiple players interact in real time.
- Finding multiple players for a game
  Discover and invite other players to participate in a real-time game.
- Adding voice chat to multiplayer games
  Enable players to voice chat with all, or groups of, players in a multiplayer game.

Matchmaking rules

Game Center applies different type of rules you create in a particular order to find the best matches.

`class` **GKMatchRequest**

An object that encapsulates the parameters to create a real-time or turn-based match.

`class` **GKMatchmaker**

An object that creates matches with other players without presenting an interface to the players.

`class` **GKMatchmakerViewController**

An interface that allows a player to invite other players to a real-time game and automatch to fill any empty slots.

`protocol` **GKInviteEventListener**

A protocol that handles invite events from Game Center.

`class` **GKInvite**

An invitation to join a match sent to the local player from another player.

`class` **GKMatch**

A peer-to-peer network between a group of players that sign into Game Center.