TabletopKit / TabletopGame

Class

TabletopGame

An object that manages the setup and gameplay of a tabletop game.

visionOS 2.0+

```
class TabletopGame
```

Overview

First, create a `TableSetup` object that represents your game layout and equipment. Add seats for players to occupy, conforming to the `TableSeat` protocol, and equipment for them to manipulate, conforming to the `Equipment` protocol.

Pass an object that conforms to the `Tabletop` or `EntityTabletop` protocol to the `TableSetup` initializer.

```
let table = Table()
root = createRootEntity(table: table.entity)
var setup = TableSetup(tabletop: table)
```

Implement your structure to initialize the protocol properties, such as `shape`, `entity`, and `id` properties for the `EntityTabletop` protocol.

```
struct Table: EntityTabletop {
    var shape: TabletopShape
    var entity: Entity
    var id: EquipmentIdentifier

    init() {
        self.entity = try! Entity.load(named: "table/table", in: contentBundle)
        self.shape = .round(entity: entity)
        self.id = .table
    }
}
```

Then, create the TabletopGame object that represents your game instance by passing the TableSetup object to the init(tableSetup:version:) initializer.

```
game = TabletopGame(tableSetup: setup)
```

Place the localPlayer in a seat at the table using claimSeat(_:) or a similar method.

```
game.claimAnySeat()
```

Next, implement an object that renders your game layout and equipment. Set the game's renderer, conforming to the TabletopGame.RenderDelegate protocol, using the addRenderDelegate(_:) method. Implement the onUpdate(timeInterval:snapshot:visualState:) method to render the current state of the game. Alternatively, conform to the EntityRenderDelegate protocol.

```
game.addRenderDelegate(self)
```

If needed, you can draw a debug representation of selected items in the game using the debugDraw(options:) method.

```
game.debugDraw(options: [.drawTable, .drawSeats, .drawEquipment])
```

Then, add actions to the equipment that controls gameplay using the addAction(_:) and addActions(_:) methods.

Finally, pass an object to the addObserver(_:) method that conforms to the TabletopGame.Observer protocol. Implement the Observer protocol methods to progress gameplay when players interact with the equipment.

Topics

Creating a tabletop game

`init(tableSetup: TableSetup, version: Int)`

Creates a tabletop game with the specified table configuration and version of rules.

`var rootPose: Pose3D`

Update the root pose for the current player

`func update(deltaTime: Double)`

Update the game manually. Call this function if `automaticUpdate` was not set when registering the Tabletop instance.

`func withCurrentSnapshot((TableSnapshot) -> Void)`

Adding equipment to the game

`var equipment: [any Equipment]`

`var equipmentIDs: [EquipmentIdentifier]`

`func equipment(matching: EquipmentIdentifier) -> (any Equipment)?`

`func equipment<E>(of: E.Type) -> [E]`

`func equipment<E>(of: E.Type, forEntity: Entity) -> E?`

Retrieves the specified equipment type associated with an entity if it exists.

`func equipment<E>(of: E.Type, matching: EquipmentIdentifier) -> E?`

Managing seats

`func claimAnySeat()`

Claims any free seat. Has no effect if the player is already seated or if there are no free seats.

`func claimSeat(some TableSeat)`

Claims the given seat. If provided Seat is not part of the table, it has no effect

`func claimSeat(matching: TableSeatIdentifier)`

Claims the given seat. If provided ID does not exist, it has no effect

`func releaseSeat()`

Releases the seat for this player. If the player is not seated it has no effect

Getting the player

`var localPlayer: Player`

The player who runs this tabletop game instance on their device.

Adding actions

`func addAction(some TabletopAction)`

`func addAction(some CustomAction)`

`func addActions(some Sequence<any TabletopAction>)`

Observing actions

`protocol Observer`

A protocol for objects that progress gameplay when players take actions.

`func addObserver(some TabletopGame.Observer)`

`func removeObserver(some TabletopGame.Observer)`

`enum ActionCancellationReason`

The possible reasons for cancelling an action or an interaction.

Jumping to bookmarks

`func jumpToBookmark(StateBookmark)`

Restores game to the given bookmark

`func jumpToBookmark(matching: StateBookmarkIdentifier)`

Restores game to the given bookmark

`var bookmarks: [StateBookmarkIdentifier]`

Starting interactions

`func startInteraction(onEquipmentID: EquipmentIdentifier) -> TabletopInteraction.Identifier?`

Starts a local interaction. It will return `nil` if too many interactions are already happening at the same time.

Canceling interactions

`func cancelAllInteractions()`

Cancels all local and remote interactions. This releases control of all the equipment and rolls back all the actions added to the canceled interaction.

`func cancelInteraction(matching: TabletopInteraction.Identifier)`

Cancel the local or remote interaction matching the given identifier. This causes any actions added to it to be rolled back, and releases the controlled equipment and any tossed equipment.

Rendering the table

`func addRenderDelegate(some TabletopGame.RenderDelegate)`

`func removeRenderDelegate(some TabletopGame.RenderDelegate)`

`protocol RenderDelegate`

A protocol for the object that renders your entire game.

`protocol EntityRenderDelegate`

A protocol for the object that renders your entire game using RealityKit.

Supporting multiple players

`func attachNetworkCoordinator(some TabletopNetworkSessionCoordinator)`

`func detachNetworkCoordinator()`

`var multiplayerDelegate: (any TabletopGame.MultiplayerDelegate)?`

`protocol MultiplayerDelegate`

An object that handles players joining multiplayer games.

Enabling group activities

`func coordinateWithSession(GroupSession<some GroupActivity>)`

Begins coordination of the game with a group session

Drawing debug representations

`func debugDraw(options: DebugDrawOptions)`

Enable or disable debug visualizations

See Also

Essentials

{} Creating tabletop games

Develop a spatial board game where multiple players interact with pieces on a table.

{} Synchronizing group gameplay with TabletopKit

Maintain game state across multiple players in a race to capture all the coins.

`struct TableSetup`

An object that represents the arrangement of seats, equipment, and counters around the game table.

`protocol Tabletop`

A protocol for the table surface in your game.

`protocol EntityTabletop`

A protocol for the table surface in your game when you render it using RealityKit.

`struct TabletopShape`

An object that represents the physical properties of the table.