Hearts Card Game in Swift

Swift is most commonly used to create apps. In order to show off many of the cool features of Swift, I have coded most of the Hearts Card Game. It is run via command line, and can contain any combination of Users and CPUs. The assignment is to implement certain functions of this game described below.

**Step 0**
Install swift on your machine, or run your code on a linux box.

**Step 1**
Get a hold of Hearts.zip
Unzip file

You can now go into the Sources folder and view the code.

**Step 2**

In general, every card of the deck is dealt out to the players. The two of clubs begins the first “trick”. The next player then must match the suit if they have an available card. If not, they can play any card. After each player has added a card to the trick, the card with the highest rank of the initial suit is the winning card, and the player that played that card receives the trick.

You cannot play a heart unless: 1. A heart has already been played OR 2. You do not have any cards of the initial suit. When a player receives a trick, they will gain a point for every heart card contained in it, and 13 points for the queen of spades. The player with the least points when someone hits the max score wins.

There is a UML diagram of the finished program included in the Hearts directory. You might view this to get a sense of how the program should function.

**Step 3**
Implement the appropriate functions to fulfill the following goals:

- Give the Array class functionality to look up the index of an element it contains

- Determine if a Player is currently holding a given suit.

- Ability for the game to find the player with the two of clubs (or nil if no one has it anymore)
- Ability for the game to add up the points that each player has gained at the end of the hand given the tricks they have accumulated. Remember to check to see if a player is in control.

**Step 4**

Note some of the cool features of Swift!

For example, in Extensions.swift we create two functions that we need because the built in libraries won't satisfy our needs.

Another example is the if let syntax. This insures that we do not get a nil value. Another example is passing functions around through some other functions (namely getCard: () -> Card).

**Step 5**

Build and run the program. If the program works, it should do this:

```
$ ls
Hearts.zip
$ unzip Hearts.zip
Archive: Hearts.zip
  creating: Hearts/
  inflating: Hearts/.DS_Store
  creating: _MACOSX/
  creating: _MACOSX/Hearts/
  inflating: _MACOSX/Hearts/._DS_Store
  inflating: Hearts/Packages.swift
  creating: Hearts/Sources/
  inflating: Hearts/Sources/.gitignore
  inflating: Hearts/Sources/Card.swift
  inflating: _MACOSX/Hearts/Sources/
  inflating: _MACOSX/Hearts/Sources/_Card.swift
  inflating: Hearts/Sources/CPUs.swift
  inflating: _MACOSX/Hearts/Sources/_CPU.swift
  inflating: Hearts/Sources/Extensions.swift
  inflating: _MACOSX/Hearts/Sources/_Extensions.swift
  inflating: Hearts/Sources/Game.swift
  inflating: _MACOSX/Hearts/Sources/_Game.swift
  inflating: Hearts/Sources/Gameplay.swift
  inflating: _MACOSX/Hearts/Sources/_Gameplay.swift
  inflating: Hearts/Sources/main.swift
  inflating: _MACOSX/Hearts/Sources/_main.swift
  inflating: Hearts/Sources/Player.swift
  inflating: _MACOSX/Hearts/Sources/_Player.swift
  inflating: Hearts/Sources/Trick.swift
  inflating: _MACOSX/Hearts/Sources/_Trick.swift
  inflating: Hearts/UML.jpg
  inflating: _MACOSX/Hearts/.UML.jpg

$ cd Hearts
$ ./Hearts
$ ./Hearts
compiled Swift module 'Hearts' (8 sources)
linking /build/Debug/Hearts
Playing Hearts in Swift

Shuffling and dealing cards...

Each player must pass three cards to the next player

CPU: [Four of Spades ₵, Five of Spades ₵, Seven of Spades ₵, Three of He]