Distributed Rock Paper Scissors game using the Swift Programming Language

Renan Greca

February 26, 2019

Abstract

1 Introduction

Glossary:

Item: either rock, paper, or scissors

Player: a participant in the game

Play: a tuple containing a player's identifier and an item

Round:

Match

2 Distributed applications in Swift

Swift was introduced at Apple's WorldWide Developers Conference in June of 2014, with the goal of replacing Objective-C as the primary language used to develop apps for Apple's macOS and iOS operating systems. Despite the clear objective, Swift was designed as a general-purpose programming language and can be used to develop many different kinds of applications and follows modern language design principles. Its three guiding principles is to be safe, fast and expressive [2]. As of 2015, Swift is open-source and its development is community-driven, with its fifth major release planned for 2019 [3].

However, it is a new language that, as of today, is still maturing. Many of the language's most optimized and documented features are those which provide useful functionality to macOS and iOS developers, since they form most of Swift's current userbase. Today, Swift's concurrent features are primarily used for performing background tasks, such as receiving data from a remote server, without interrupting the user's interaction with the application.

The Swift Concurrency Manifesto [1] provides the guidelines that should be followed in future enhancements to the language related to parallel, concurrent and distributed computation features, but many of these features are still lacking today. Thus, developing a distributed application in Swift provided a certain degree of challenge, because little documentation exists on Swift's distributed capabilities.

In order to develop a distributed program in Swift, a few options for inter-process communication were considered [4]. Swift provides some low-level APIs, inherited from Objective-C, that would allow communication between processes using sockets, but these APIs are poorly documented and generally not designed for application developers; they form the foundation for friendlier APIs. Ultimately, distributed notifications were chosen for passing messages between processes.

Distributed notifications are a feature of macOS that allows for processes to send data to other processes by sending and observing notifications [5]. Each notification has a string identifier; when one process sends a notification with a certain identifier and another process is observing notifications with the same identifier, information is transmitted. The advantage of this approach is its simplicity to implement; the disadvantage is that it relies on an operating system feature and therefore restricts usage to devices running macOS.

3 Program Design

The main goal of this implementation of Rock Paper Scissors was the distributed approach. That is, each player of the game is represented by a process that possesses its own memory and knows very little about its peers.

Generally speaking, each player must perform the following steps for a match of Rock Paper Scissors to occur:

1. Announce its presence to other players and discover its opponents;

2. Send a play to the other players and receive plays from the others;

3. Check each play to count the number of points each player should receive in the current round;

4. Repeat steps 2 and 3 until enough rounds have been played.

The `Player.swift` file contains the code used for each player process. It features three command-line arguments: `noPlayers`, which determines how many players are expected to be in a match; `noRounds`, to set the number of rounds to be played; and `shouldPrintLogs`, which determines whether or not that particular process will display its log messages.

When a player process is launched, the first thing it does is to announce its existence by broadcasting a "hello" message. This is done using a notification with the `player.begin` identifier, which also carries the process identifier, which is used to identify the different players in the match. Keep in mind that, if the player is the first one to join a match, no one will receive this notification.

Upon receiving a "hello" message, a player replies with another message. This allows all players to identify each other: they receive either a "hello" or a reply from each other player. Since the notifications are broadcasted, one process may receive several replies from the same player; in this case, redundant messages are ignored. From this point onward, each player stores an independent data structure that contains every player's identifier and score.

Once the number of players identified is equal to the `noPlayers` argument, the match begins.

4 Evaluation and Results

5 Conclusion

References

[1] Chris Lattner (2017), "Swift Concurrency Manifesto," `https://gist.github.com/lattner/31ed37682ef1576b16bca1432ea9f782#part-4-improving-system-architecture`.

[2] Apple (2019), Swift," `https://swift.org/`.

[3] Apple (2019), Swift Evolution," `https://github.com/apple/swift-evolution`.

[4] Mattt (2015), Inter-Process Communication," https://nshipster.com/inter-process-communication/.

[5] Apple (2019), DistributedNotificationCenter," https://developer.apple.com/documentation/foundation/distributednotificationcenter.