98Point6 Interview Software Design Document

Stan Rosenbaum - Jan 2021

Reviewers: 98Point6 Hiring Committee

# INTRODUCTION

## Document Outline/ Overview of Contents of Document

This document is a blueprint for the architecture for the 9dt mobile game for iOS. This document will begin with a cover a high level description followed by lower level description to provide details for the application system.

## Goals

The purpose of the product to provide simple light diversion, in the form of a game in which the user can play is quick short bursts.

## Scope

This document is over covering the high and low abstract descriptions of the needed designs, data structure, and algorithms to adequately describe the requirements that must be met to completion of functional project. Actual implementation details, are left to the developments team(s) to provide fallibility for day to day tasks and goal.

## System Overview

The target deploying platform is any and all iOS and iPadOS platforms running iOS/iPadOS version 12 or higher

This is make the earliest hardware supported

* iPhone 6s (2015)
* iPhone 6s Plus (2015)
* iPhone SE (2016)
* iPad mini 4 (2015)
* iPad Air 2 (2014)
* iPad 5 (2017)

Any models of hard older then these device are considered obsolete.

For development, each developer will require a MacOS desktop, capable of running XCode 10.0 or greater at a minimum. As development without Apple Branded product is more trouble than it is worth.

## Milestones

# Design Considerations

## Assumptions and Dependencies

We are assuming the game’s REST API server is always available to provide turn responses.[[1]](#footnote-1)

## General Constraints

Time – No More than a month.

Apple Store – Approval

Server – Reliability

## Goals and Guidelines

The game is able to save its state, to allow the player to quit and resume a game session at any time.

The interface should be simple clean, accessible (for Voice Over and other motor impairments).

Localized, so it will be localizable ready, to expand into other language markets.

## Requirments

1. The app must allow the player to choose whether they want to go first, or if they want our service to go first.
2. If there is a win on either side, the app must display who won and let the player play again.
3. If the board is full, the app must tell the user the game is a draw and let the player play again.

## Development Methods

A general Agile Scrum methodology, with a planned technical debt time per sprint.

Although Test Driven Development can be uses to verify the integrity of parts of the system, the bulk of testing must be active user testing due to the interactive nature of the application as a game.

General code reviews should be encouraged. If possible, the 5th grade legibility test should be used toward the end of the project.

## Risks and Alternatives

Time is always a concern.

Data formatting from server is not normal JSON data, this prevents use of Codable data structures to simplify data.

Game state is entirely maintained by client app. Automated tested with the server, although possible, is not guaranteed to return consistent results.

# General Overview and Design Guidelines/Approach

# Assumptions/Constraints/Risks

## Assumptions

We are assuming the game server is operating under the same specification as detailed in the proposal document.

Application should be easiely modifiable and scaleable for future versions. To try size sized grids.

## Constraints

We must ensure that the application has internet connectivy .

We also must ensure the interface is accessible

## Risks

Time is always the number one concern. Although a Month development should be ample time to allow for simple project

Team size. Due to the small size and scope of the project, team size can be a concern. This project should not require a large or even a medium sized development teams. One + one developer, a designer, and a project manager should be enough to cover most cases, where the designer or one of the development may even be able to double as project manager. Concern must be take to make sure the team does nto become too large, as then it would become a text book case of wasted resources, or in classical mythical man month an impediment to the project itself.

# Design Considerations

## Goals and Guidelines

Make efficient use of the UIKit for the presentation.

Use of external assets to minimize use of custom UIViews.

Development of the clean and easily understandable code flow that the project could be used in middle school instruction and understood by students. Like standard English prose, the team should aspire to write code easily understood able at a 5th grade level to achieve maximum readable and understanding.

Clear cut use of the Model View Controller (MVC) Patterns for code reuse if ported to MacOS platforms.

## Development Methods & Contingencies

## Architectural Strategies

Use of Model View Controller (MVC) Patterns

Use of [UIKit Coordinator Pattern](https://www.hackingwithswift.com/articles/71/how-to-use-the-coordinator-pattern-in-ios-apps) to avoid use of Segues (which increases coupling in iOS applications).

## Performance Engineering

Architectural Strategies

# Architectural Design

## Client

This is a client application that communicates with a Stateless Anonymous Server.

Client / Server Software Layers

# Component design

## Application Overview

UIKit Single Window, Single Scene Application

Use of OS.Logger to increase debug reporting

Unit Tests whenever possible to test individual components.

## Human User Interface Design

Being an interaction application we are addressin

### 508 Compliance

## System Architecture

Standard Model View Controller (MVC)

The App should use a standard MVC model where there Controller is the central point for

### Model

NetworkManager(): Application’s contact point with the server via network URL calls.

### Controller

DataManager():

The DataManager job is to negotiate maintain the current game field data in the form of a fixed array of arrays, columns first.

We want to use columns first because that can simply report column state for Accessibility announcements.

By using an array of array, game win and loss then be calculated recursively (similar to the N-Queen problem, see GameLogic for more).

GameLogic()

The primary job of the GameLogic Class is to determine if there is a winner with every added turn. The GameLogic classes check if the most recent piece added results in a winning condition.

### UI

GameView()

The GameView should be collection of gameCell views grouped in by column and then rows.

In terms of Accessibility each column should be interacted with as one whole unit. The Accessibility label for each column unit will need to be set to read the content of the column from the bottom up.

Accessibility Magic Tap for each column triggers the game to respond as if the player selected the column for their turn.

UICell:

The individual cells should be UIImageView components with added UITouchRecognition methods. The Images for each cell should be set programmatically. If possible, use PDF vector files for the default image format as XCode can automatically generated 1x, 2x, 3x… Similarity, all images access should be places in a dedicated asset catalog file.

## Logical and Process Flows

Performance Hardware Architecture

Software Architecture

Security Architecture

The largest security concern could be man in the middle attacks, of interception data between client and server.

Invalid data not applicable for the current game could be simply ignored.

Valid data, but non-authetic data could be used in liu of authentic data.

Performance

The game should be providing good performance in terms of maintaining User engagement. Although,

System Architecture Diagram

System Design

Business Requirements

# Data Design

## Component Design

Data Objects and Resultant Data Structures

Data Conversion

User Machine-Readable Interface

Inputs

Outputs

Detailed System Design

Classification

Definition

Responsibilities

Constraints

Composition

Uses/Interactions

Resources

Processing

Interface/Exports

Detailed Subsystem Design

Glossary

# Requirements

Bibliography

Operational Scenarios

Detailed Design

Hardware Detailed Design

Software Detailed Design

Security Detailed Design

Performance Detailed Design

Internal Communications Detailed Design

System Integrity Controls

External Interfaces

Interface Architecture

Interface Detailed Design

Appendix A: Record of Changes

Appendix B: Acronyms 0

Appendix C: Glossary 1

# Appendix D: Referenced Documents

## Asset Catalog

<https://www.simpleswiftguide.com/how-to-add-image-to-xcode-project-in-swiftui/>

## Localization

<https://medium.com/swlh/app-localization-in-swift-ios-swift-guide-baa2c2e4298e>

<https://medium.com/@Jeehut/localization-in-swift-like-a-pro-48164203afe2>

## Vector Images

[https://useyourloaf.com/blog/xcode-9-vector-images/](https://useyourloaf.com/blog/xcode-9-vector-images/ )

## Image Literals

<https://medium.com/ios-os-x-development/be-literal-76e9b4389eda>

<https://medium.com/@gurdeep060289/color-image-new-literals-in-the-cocoa-town-7ef4f2710194>

Appendix E: Approvals 3

Appendix F: Additional Appendices 4

Appendix G: Notes to the Author/Template Instructions 5

Appendix H: XLC Template Revision History 6

6 PERFORMANCE ANALYSIS 5

7 FEASIBILITY AND RESOURCE ESTIMATES 6

8 SOFTWARE REQUIREMENTS TRACEABILITY MATRIX 7

APPENDIX A: ITMS MODULE MAP 8

APPENDIX B: REQUIREMENTS TRACEABILITY MATRIX 9

Appendix C: Data Flow Diagram 0

APPENDIX D: APPROVALS 2

1. A possible extended goal it for the client app to provide a game service if the REST API is unavailable. This is discussed later. [↑](#footnote-ref-1)