

6COSC021W Lab-Based Practical – Specification

Before the assessment starts:	1
Instructions	2
Inputs of the application	2
Outputs of the application	2
Other requirements	2
User Interface(s)	3
The Game screen	3
The Settings screen	4
Appendix A	6
Enums	6
Tint View modifier	6
Color initialization	7
Font Size	7
Disabling	7
Appendix B	8
Randomizing	8

You have 2 hours to complete this assessment. You will be told when you have; 1 hour, 30 minutes, and 15 minutes remaining.

Before the assessment starts:

1. Download the assessment project template from Blackboard.
2. Open the Assets catalog to see the 3 System colors you have to use
3. Run the project and ensure there are no errors or build issues.
4. Make sure you have to hand in any allowed notes you require. See Appendix B.
5. Spend some time inspecting the starter project. The code's main structure will be stubbed out with comments indicating what code needs to be placed in these sections.
6. Familiarise yourself with this brief and ask any questions if you need clarification on any point before you begin coding.
7. Appendix A provides a few hints on how/what you can use to complete this assessment

Instructions

You are expected to create a fun math game for junior students learning math basics. This game will generate random math questions so the students can practice **addition, subtraction, division, and multiplication** leisurely.

Inputs of the application

The application has 3 inputs where only one input is mandatory:

1. User input for the guessed answer [MANDATORY].
2. User input to change the font size
3. User input to change the System color

Outputs of the application

The application has the following outputs based on whether the guessed answer is correct or wrong once the “Submit” button is pressed.

1. If the answer is **correct**.
 - a. The app must increment points in the middle of the screen by 1.
 - b. A text should congratulate the user for their guess if the answer is correct.
2. If the answer is **incorrect**.
 - a. Points in the middle of the screen should be decreased by 1 (Points > 0).
 - b. A text should show the user the correct answer.

Other requirements

The application has several other expectations. They are

1. Operators should only include numbers between 0-9 ($0 \leq x < 10$)
2. For generating the Operators (+, -, /, *), you have to use an `enum` (See Appendix A)
3. Once the answer is submitted, the “Submit” button should be disabled to prevent the user from submitting an answer for a single question more than once
4. A question should be generated:
 - a. Each time the user changes/reopens the game screen (use `.onAppear {}`)
 - b. When the “NEXT” button is pressed
5. When the “NEXT” button is pressed:
 - a. The user's previous input has to be cleared.
 - b. At the same time, the app must show a new question to the user.
 - c. The app should clear the text congratulating/correcting the user's guess
6. The user should only be able to enter ASCII-enabled characters into the app (No Emojis)
7. The app should not increment the points lower than 0
8. The app should persist the points that the user has gained. For this, you can leverage `@AppStorage`.
9. All inputs, calculations, and outputs (Except font size) should be done in Integers
10. See Appendix B for more requirements/hints

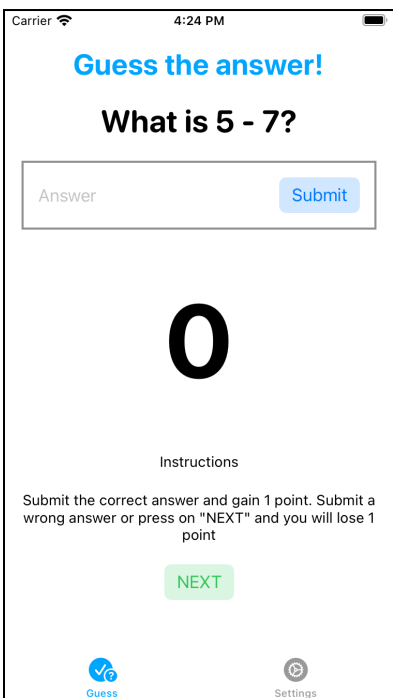
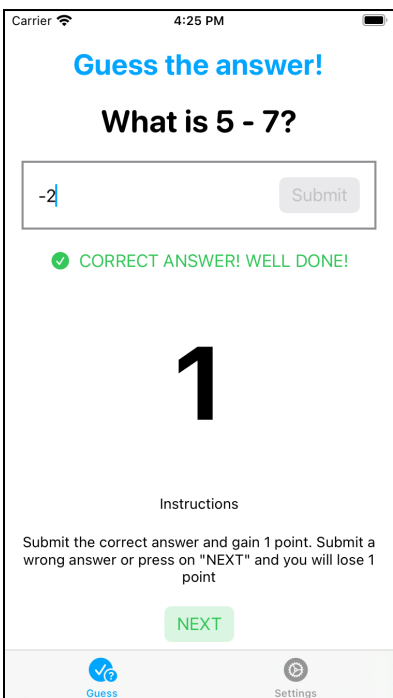
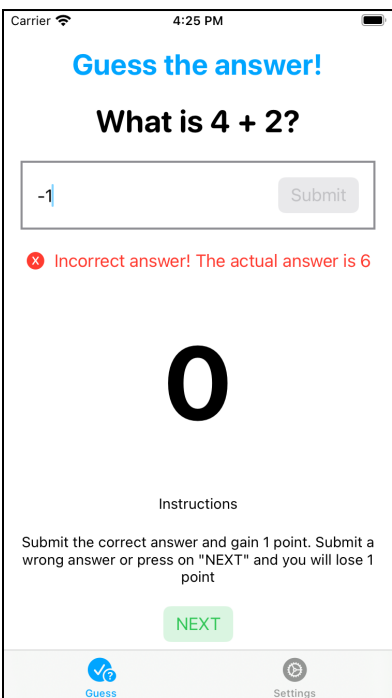
User Interface(s)

It is expected that you try to replicate the given UI as much as possible.

There are two tab screens in the expected math calculator application. They are:

1. The game screen - Consolidates components relevant to the game
2. The settings screen - Settings to change the game's appearance

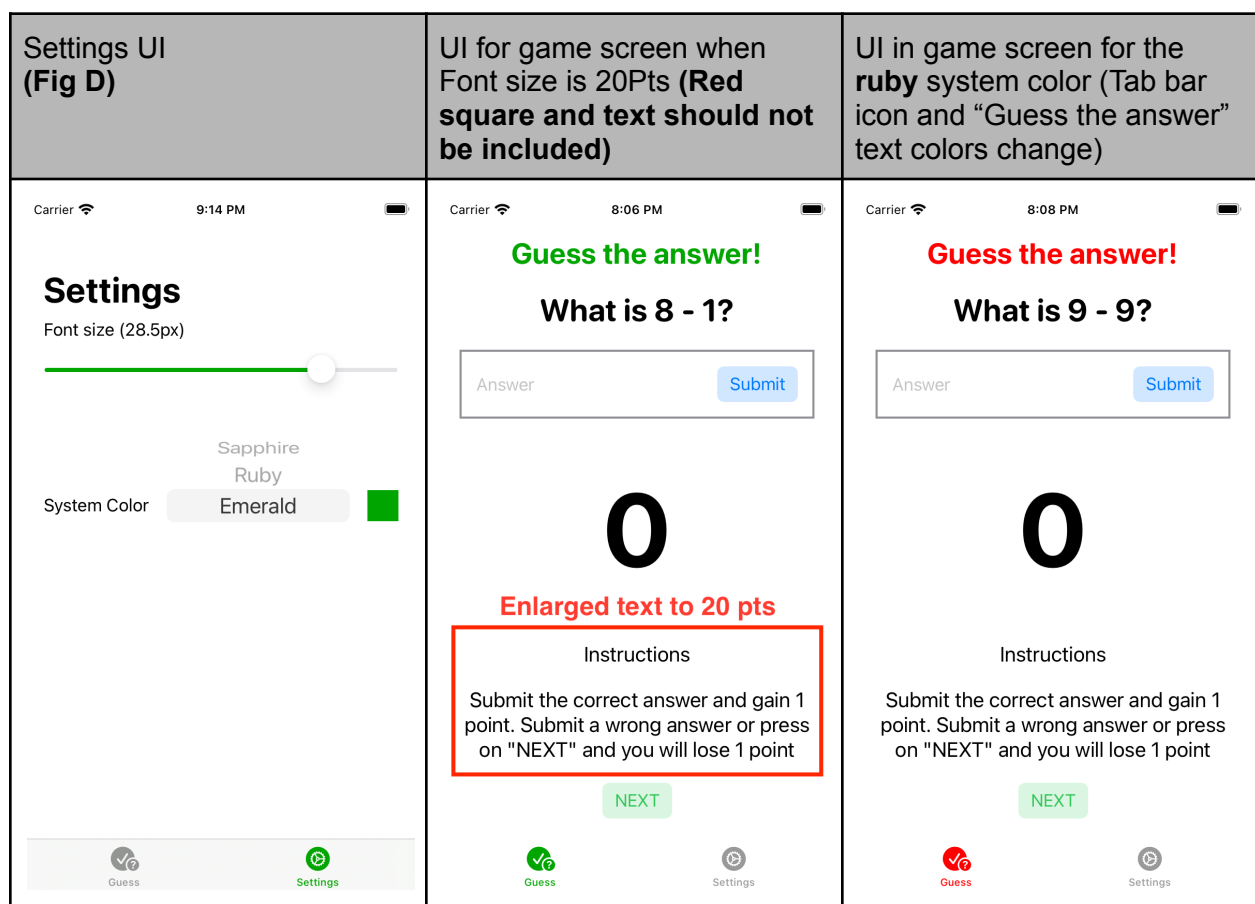
The Game screen

Unanswered UI (Fig A)	UI for a correct answer (Fig B)	UI for an incorrect answer (Fig C)
		

- Note that in **Fig B** and **C**, when the user submitted their answers, the “Submit” button is disabled till “Next” is pressed
- The “Guess the answer!” text is a standard SwiftUI Text component with a certain styling

- The following items should also be configurable from the settings page. You can use a common `@AppStorage` **key (String param)** to access these settings on any/all of your screens.
 - The “Guess the answer!” tint color should change when the System Color in the settings screen changes (See Fig D).
 - The font size of the “Instructions” and “Submit the correct...” text in the game screen should change based on the font size selected in the settings screen using the slider (See Fig D).
 - The TabBar tint should also change according to the System Color selected in the settings screen.

The Settings screen



- The “Settings” title is a title added to the navigation bar in the settings screen.
- The **System Color** section has a 30x30 square beside the picker to show the selected color to the user.
- The Font Size and System Color should be persistent, i.e., the settings chosen should be visible to the user once they quit and reopen the app.
- The selected Font size **should** be reflected in the “Font Size (Y px)” text and should be formatted to **one** decimal place

- The pickers should be styled as what is provided in the screenshots.

Appendix A

Enums

Assume you want to classify if some random person in the university is a student or a teacher. An enum can be used instead of hard-coded strings for “student” and “teacher” to run functionality based on the Person type. An enum allows you to create your own “Types” or datatypes. See how you can create and use an enum below

```
// Used to create your own "types"
// Here a person can be a student or a teacher and never any
other type
// Enum Definition
enum Person: String {
    case student = "Student"
    case teacher = "Teacher"
}

// This variable can be a @State variable
// the person constant will hold one of the cases defined in the
Person enum. Can be student or teacher
let person: Person = .student
// Running functionality based on the type of "person" variable

// Switch cases should always be inside a code block. e.g., a
function.
// Enum usage based on the Person type in person
var someString: String = ""
switch person {
case .student:
    someString = "This is a \"(person.rawValue)\" //.rawValue is
the value after the Enum case definition (after the "=" in the
enum )
case .teacher:
    print("This is a teacher")
}
```

Tint View modifier

There are two types of modifiers that can be used to change the foreground color within a SwiftUI app. They are the `.foregroundColor(_:)` and `.tint(_:)` view modifiers. The

`foregroundColor(_:)` is usually used for text-based components. Whereas `.tint(_:)` is primarily used for controls/control surfaces that the user **directly touches and interacts** with. Remember that view modifiers can be added to topmost views to give a common style to all.

Color initialization

A custom color in the assets catalog can be initialized using a similar approach to how an image from the assets catalog is shown in a SwiftUI app, i.e., by passing the name of the assets into the respective Views (Image/Color) constructor.

Font Size

In SwiftUI, sizing related to various components primarily use `CGFloat`. `CGFloat` is just another fancy way of saying `Double`.

Disabling

SwiftUI allows you to disable control surfaces using a view modifier named `.disable(someBool)`

Appendix B

Randomizing

1. For generating random operands you can use Swift Ranges and `.randomElement()`

E.g:

```
let range = 1..<100 // Generates a range between 1 to 99
let randomNumber: Int? = range.randomElement() // Generates
a random number from the range
```

2. For generating random operators, you can implement `CaseIterable` to your enum. Let's consider the previous enum example in Appendix A. `CaseIterable` allows you to get an array of all the cases in the enum so you can run array-based functions on it

E.g:

```
// Enum conforming to CaseIterable
// CaseIterable allows you to get an array of all the cases
// String is the type of the enums value. Can be used as
student.rawValue

enum PersonType: String, CaseIterable {
    case student = "Student"
    case teacher = "Teacher"
}

let studentType: PersonType = .student
let personTypeArr: [PersonType] = PersonType.allCases //
Getting an array of PersonType
let randomPersonType: PersonType? =
personTypeArr.randomElement() // Getting a random element
print(randomPersonType.rawValue) // Prints the respective
string for the enum case
```