ASSIGNMENT 2:
iOS Development

Name: Truong Hong Van (s3957034)
Instructor: Mr. Tom Huynh
COSC2659|COSC2813 iOS Development
RMIT University Vietnam, Semester 2024B

Table of Contents

Introduction

I. Project background

From Candy Crush to Gardenscapes, the match-three game genre has long captivated audiences worldwide, regardless of gender and age gaps (Anatasia 2024). This popularity could be attributed to the fact that this type of game typically has simple rules and intuitive user interfaces. Such simple rules can still challenge the players' minds to find appropriate strategies to achieve the goals. Therefore, match-three games balance strategy, creativity, and relaxation. Another interesting finding is that even with similar mechanics, every three-match game provides a unique world and theme, such as a candy world and garden-building journey, making them distinguishable and attractive. These characteristics have inspired me to create *Emotis*, a simple three-match game that guides players through the journey of emotions with our food friends.

II. Project overview

1. **Goals and objectives**

This project aims to offer fully functional and engaging match-three board games for users. The objectives include:

- Provide a well-functioned game view with precise logic for players to interact with the elements.
- Offer game progressions with adjustable difficulty modes to suit the user's preferences.
- Enhance user experience with an intuitive interface with sound effects, language translation, and customizable themes.
- Provide guidance for novice users and statistics illustrating the players' achievements.

2. **Target audience**

The target audience for this game includes individuals who enjoy match-three games and seek a relaxing yet mentally stimulating experience, from children to adults. This report demonstrates the app's functionality, which can serve as a user manual or a reference to developers and stakeholders interested in the game implementation.

III.Game rules

In Emotis, the primary objective is to create element chains that match the defined targets for each level. A detailed explanation of the game's concepts will be outlined below.

1. **Basic concepts**

- Element chain: An element chain consists of three or more elements of the same type in a row or column. To create these chains, the player interacts with the elements by

swiping vertically or horizontally. The element chains will then disappear, making room for new ones, and any relevant goals will be updated automatically.
- Valid swap: A swap is valid when it results in an element chain.
- Invalid Swap**:** An invalid swap is when a player attempts to swipe elements that create no chain.

Each gameplay consists of:

- Goal: The player needs to collect a specified number of a particular element type to achieve the target.
- Moves: The player has a limited number of moves to accomplish the goal. Each valid swap decreases the move count.
- Score: The player earns points for creating chains, with a base score of 60 points for a three-element chain.
- Timer: Introduced in hard mode, it requires the player to complete the goals within the time limit.

Overall, a level is accomplished if the player fulfills all the goals within the move and time limits. The game fails if the player runs out of moves or time before meeting the targets.

2. Difficulty and Level

There are three difficulty modes, which can be adjusted in the Game Settings.

- Easy mode: The player needs to fulfill one target with limited moves only.
- Medium mode: The player needs to complete two target elements with limited moves.
- Hard mode: The player needs to fulfill two targets, with a specified number of moves and limited amount of time.

As players progress to higher levels, the difficulty increases by requiring more elements to be collected and reducing the number of available moves, stimulating the users to find the best strategies to win.

Implementation details

I. In-scope

This project development could be divided into two phases: the game logic and the user interface. On the one hand, SpriteKit, a Swift framework for 2D games is employed (Apple Developer 2024a) primarily for organizing the game scene elements and detecting interactions between user and elements. On the other hand, the SwiftUI framework is responsible for handling the application views and logic. One of the challenges is the integration of SwiftUI with SpriteKit because SpriteKit follows UIKit's principles. Nevertheless, SwiftUI currently provides a way to display SpriteKit's

game scene using SpriteView (Hudson 2022). The detailed implementation will be explained below.

1. **Game (logic and view)**

a. Game logic

Each gameplay is a level that the player needs to beat; therefore, the level covers all core mechanics of the game operations, including managing element and tile grid, detecting chains of elements, handling swaps, calculating scores, and updating the target quantity.

Element and tiles management

Initially, the JSON file stores a 2D array that serves as the basis for the game board's initialization. The 0s in the array indicate no tile placement. The game grid is initialized with elements randomly placed, ensuring that no matching chains are present on the board. To maintain the principle, a repeat-while loop regenerates an element at a specified position until its adjacent elements are of different types. This ensures that new tiles are created without forming immediate matches.

```
if tiles[col][row] != nil {
        var type: ElementType
        // Ensure that no element chain exists on the initial board
        repeat {
            // Randomize based on chosen character set
            if self.characters == "food" {
                type = ElementType.random1()
            }
            else {
                type = ElementType.random2()
            }
        }
        while (col >= 2 &&
               // Check the adjacent elements in a row
               elements[col - 1][row]?.type == type &&
               elements[col - 2][row]?.type == type) ||
                // Check the adjacent elements in a column
                (row >= 2 &&
                 elements[col][row - 1]?.type == type &&
                 elements[col][row - 2]?.type == type)

                // Add new tile to the array
                let newTile = Element(column: col, row: row, type: type)
                elements[col][row] = newTile
                set.insert(newTile)
    }
```

Figure 1. Gameboard initialization

```
for r in 0..<rows {
    for col in 0..<columns {
        if let e = elements[col][r] {
            // Try swapping the element horizontally
            if col < columns - 1, let e2 = elements[col + 1][r] {
                elements[col][r] = e2
                elements[col + 1][r] = e
                // See if either element is a part of a chain
                if hasChain(atColumn: col + 1, row: r) || hasChain(atColumn: col, row: r) {
                    // Swap either way
                    set.insert(Swap(elementA: e, elementB: e2))
                }
                // Swap back
                elements[col][r] = e
                elements[col + 1][r] = e2
            }
```

Figure 2. Swapping elements

Swapping elements and chain detection

Players interact with the game by swapping elements to form chains of three or more matching elements. If the interaction is considered a valid swap, the swap() function handles it by changing the positions of the two involved elements in the game board array. In other words, element A moves to element B's position, and vice versa.

Next, the hasChain() function checks whether an element is part of a valid chain, both horizontally and vertically. The function examines the adjacent elements in all four directions and counts the

number of consecutive elements of the same type. A chain exists if there are three or more matching elements.

At the beginning of the game, the system determines if there are any valid swaps to ensure the game is playable with at least one possible move. The detection is applied to each element on the game board by temporarily swapping elements and checking if any chains are formed. If a valid chain is detected, a swap is added to the set of possible swaps, and the involved elements are swapped to their original positions. This process facilitates the valid swap check during gameplay, as the system only needs to check if the player's swap exists in the predefined set.

Shuffling, removing chains and topping up elements

Regarding the shuffle, the game will shuffle the elements until a move is available in case no possible swaps exist on the board. This approach ensures that the player always has options and keeps the game progressing.

In terms of chain detection, any time the player performs a swap, the game will identify and add the chain to a set for removal. For example, when finding a horizontal chain, the detection starts from a specified element and counts the number of adjacent elements of the same type until no more matches are found. Once identified, the chains are removed from the board by setting the corresponding elements to nil.

After a chain is formed and disappears from the board, it leaves empty spaces. The topUpElements() function fills these empty spaces by randomly assigning new elements. However, to ensure variety, the newly created element cannot have the same type as the last newly added element.

Targets, scores and combos

The game rewards the player with points for each chain formed, with a base of 60 points. This reward gets higher if the player successfully creates successive chains, or combos in a move. The score is then calculated based on the length of chain and the current combo multiplier. Besides, the game will track the player’s progress towards the goals by updating reducing the target quantity during the game play. This ensures that the player is aware of how close he is to achieving the level goals.

b. Game View

Main game board

The visual representation of the game board is organized in three layers using SKScene, an object that controls the SpriteKit content (Apple Developer 2024b). To briefly explain, the game layer acts as the container of the entire game board, ensuring that the board is positioned relative to

the center of the screen (using an anchor point). Next, the tiles layer displays the background tiles, where the elements can be placed. Finally, the elements layer contains all interactive game elements or sprite nodes. In this case, a sprite is a 2D image of an element, and we can specify the size and position of each sprite (Apple Developer 2024c). By that, the elements layer is rendered on top of the tiles, and this layer also manages their swiping animations and user interactions.

User interaction detection

The user interactions with the elements are detected by the method provided by SpriteKit, touchesBegan(), touchesMoved(), and touchesEnded() (Murugan 2015). The interaction flow begins when the first player's touch on the screen is identified. Then, the touch location is covered into relative columns and rows within the game grid, from bottom to top and from left to right. If the touch falls within the bounds of the game grid and there is an element in the location, an animation highlights the selected element. As the player moves their finger, the game will continually detect the current touch location, to identify the swipe direction, either horizontal or vertical. If it is a valid direction, the game will try swapping the touched element with the adjacent element in the swipe direction. Finally, the interaction ends when the player moves his finger off the screen.

```swift
// Detect the swipe direction
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
    // Convert the location into valid row and column number
    guard let touch = touches.first else { return }
    let location = touch.location(in: elementsLayer)
    let (success, column, row) = convertPoint(location)
    if success {
        var horizontalDiff = 0
        var verticalDiff = 0
        guard let fromColumn = swipeFromColumn, let fromRow = swipeFromRow else { return }
        // Swipe left
        if column < fromColumn {
            horizontalDiff = -1
            print("Swipe left")
        }
        // Swipe right
        else if column > fromColumn {
            horizontalDiff = 1
        }
        // Swipe down
        else if row < fromRow {
            verticalDiff = -1
        }
        // Swipe up
        else if row > fromRow {
            verticalDiff = 1
        }
        if horizontalDiff != 0 || verticalDiff != 0 {
            trySwap(hDiff: horizontalDiff, vDiff: verticalDiff)
            hideSelectionEffect()
            // Ignore the rest of the swipe motion
            swipeFromColumn = nil
        }
```

Figure 3. Identify touches moved

```swift
/* Handle Removing Chains */
// Implement animation for removing chains
func animateRemoveChains(for chains: Set<Chain>, completion: @escaping () -> Void) {
    for chain in chains {
        animateScore(for: chain)
        for element in chain.elements {
            if let sprite = element.sprite {
                // Trigger animation for each sprite
                if sprite.action(forKey: "removing") == nil {
                    let scaleEffect = SKAction.scale(to: 0.1, duration: 0.3)
                    scaleEffect.timingMode = .easeOut
                    sprite.run(SKAction.group([
                        SKAction.sequence([scaleEffect, SKAction.removeFromParent()]),
                        disappearSound
                    ]), withKey: "removing")
                }
            }
        }
    }
    // The game will continue after the animation finish
    run(SKAction.wait(forDuration: 0.3), completion: completion)
}
```

Figure 4. Remove chains

Scene animations and sound effects

The animations, such as swiping gestures and removing elements, are handled using SKAction, which allows us to animate a sprite node (Jacobson 2017). To perform an animation effect, for example, in the case of removing chains, we can define animation properties such as scaling and fading, and then run them in a sequence of actions. In this case, the sprite is scaled down to 10%

of its original size over 0.3 seconds before disappearing from the board. For sound effects, the built-in playSoundFileNamed() function is used, allowing sound effects to play in sync with the animations.

Logic – View connection, Data flow and Save and Resume

On the one hand, the game manager acts as the bridge between the game logic and the game view. The flow of game properties such as goals and scores are handled using ObservableObject protocol and Publish property wrapper, enabling the affected user interface to automatically update when the data changes (Apple Developer 2024d). The game manager interacts directly with the game scene and level manager to update the data, handle swipe gestures and determine whether the game is over.

On the other hand, the game manager handles the save and resume feature by controlling the GameState structure, which stores necessary game data, including the time remaining and the current elements' position on the game board. When a new game starts, a game state is initialized, and when the user exits the app, the current game state is saved. Upon opening the app again, if a saved game exists, the game manager will load the saved game and restore all relevant details encoded in the UserDefaults.

Main game view

The main game view is organized in a VStack, with the level number, score, and target box above the game board, ensuring that the targets are visible as the player progresses. When the game manager indicates that the level is accomplished, a modal showing the level score, combos, and any achievements the player gains is displayed on top of the game. The same approach is applied when the player fails, but no score or achievement is considered.

Figure 5. Game view

Figure 6. Leaderboard

Figure 7. Statistics

2. Leaderboard view

Functionality	Description
Leaderboard	The leaderboard data, including the player list, is stored in UserDefaults, ensuring the data persists across app launches. When a user registers his username in the Welcome view, it is added to the list, provided it is unique. To facilitate this process, a custom UserDefaults handler is implemented to store and manage an array of Player objects. This custom handler decodes and encodes the array of Player objects when loading and saving data during registration. Finally, the top ten players are displayed in rows on the leaderboard, showing their ranks, usernames, total scores, and highest achievements.
Top 5 players bar chart	The bar chart visualizes the comparison of total scores among the top five players on the leaderboard. Implemented using the Charts framework in SwiftUI, it leverages built-in properties such as marks, annotations, and axes to construct the chart (Apple Developer 2024e). For interactivity, players can tap on a bar to view more details,

	thanks to the annotation modifier. The annotation box displays the selected player's total score accordingly.
Player's score by level line chart	The line chart allows users to choose a specific player from a picker and view their score trend by level. Each scoreline is displayed using LineMark, with individual scores represented by PointMark. Since the line chart does not natively support the onTapGesture function, which would enable tapping on a point, a hidden circle button is applied to cover each point. The position of these buttons is calculated using GeometryReader, allowing users to interact with the points and view more details conveniently.

Figure 8. Leaderboard view features

3. Setting view

The game setting allows users to change the application theme, difficulty mode, language, and character sets during the gameplay. These changes are all stored in UserDefaults and AppStorage to ensure they remain consistent across the app launches. This view is displayed in a sheet view, with the presentation height from medium to large.

Functionality	Description
Light / Dark theme	The player can switch between light and dark modes using a picker. The theme is controlled by the AppStorage property wrapper, ensuring that when the user selects a theme, all views in the app immediately update their color palettes to match the chosen theme.
Mute / Unmute background music	The game manages sound effects and background music separately, allowing them to play simultaneously. Specifically, the background music is managed by the BackgroundMusicManager class, which follows a Singleton pattern, ensuring that only one instance of this class exists in the application (GeeksforGeeks 2024). Moreover, to make the music loop, the number of loops is set to -1 (Hudson 2019). However, the user can mute the background music by tapping the speaker icon. This background music is consistently applied in every view of the app.
Language translation	The application supports two languages: English and Vietnamese. To enable translation, a string catalog is used to translate text into Vietnamese and handle pluralization (Apple Developer 2024f). After manually translating the text in the catalog, a LanguageManager conforming to ObservableObject is implemented, allowing affected views to react immediately to language changes. The player can select a language using a language picker, and the chosen language is stored

	in UserDefaults, ensuring that the application updates the language immediately upon selection.
Character set change	The player can change the character set on the game board from food to animals. When selected, the character set name is saved to UserDefaults. This data is later retrieved by the GameManager and Level views to render the sprite nodes according to the chosen set.
Difficulty mode	The application provides three difficulty modes, with different goals and properties. The mode chosen is handled by UserDefaults to be later taken from the Level manager. Depending on the mode, the level will retrieve the related level pack loaded from the JSON file. Additionally, the hard mode has the time limit for each level, and it is controlled by the Timer object. When the game starts, the timer starts counting down, and it will stop if the user accomplishes all the goals before the time is out.

Figure 9. Setting view features

4. How to play view, Welcome view and Responsiveness

Functionality	Description
Welcome view	The Welcome view is the first screen the player encounters upon opening the application. Using the NavigationStack and navigationDestination modifier, the player can tap buttons to navigate to other main views of the app. If the player chooses the “Play” button, they must first register a username before being able to start the game.
Interactive how to play view	The How to Play view provides step-by-step instructions on how to play the game. A current step State controls the steps, allowing an appropriate view to be rendered according to the current step. Instructions are displayed at the top of the screen, and the player can move to the next or previous instruction by tapping the arrows. At each step, an animated bouncing arrow highlights a part of the game that the player needs to focus on for a comprehensive learning experience. In the final step, one of the possible swaps on the game board is highlighted to guide the player. To exit this view, the player must try the gameplay until the game is over.
Device responsiveness	The application is designed to be compatible with multiple devices, from iPhones to iPads. This is achieved by implementing frame and text sizes relative to the UIScreen, rather than using fixed size values. For example, the title text size is defined using the

	UIScreen.main.bounds.width * 0.05, ensuring the text scales appropriately according to the screen size.

Figure 10. How to play, welcome views and responsiveness

II. Out-of-scope / Known problems

Although Emotis is designed to be fully functional, there are some limitations and unexpected issues encountered during development that significantly impact the application's performance and functionality.

1. **Limitations**

- Insufficient data: Emotis currently offers only five levels for each difficulty mode, which may limit the game's replayability.
- Limited sound effects: The game only supports sound effects during gameplay, with no additional audio effects outside the core gaming experience.
- Limited languages: The application is currently available in only two languages, English and Vietnamese, which restricts its accessibility to a broader audience.
- Unlocked levels: All levels are accessible immediately upon registration, which may reduce the sense of progression or achievement typically expected in similar games.

2. **Known problems**

- Directional swiping issues: Players can reliably swipe only from bottom to top and from left to right when performing a swap. Swipes in other directions are sometimes possible but are inconsistent and unreliable.
- Performance-related freezes While some levels render smoothly, certain gameplay scenarios cause the application to freeze due to performance issues during loading.
- Game State loading bug: Although the game state correctly saves the position of elements, it fails to load the saved element array upon resuming. If the game attempts to display the saved game board, the application freezes, even though it continues to receive the player's touch input.
- Device compatibility issues: The application does not fully support device compatibility, as some elements are not positioned correctly when switching to larger screens, leading to an unsatisfactory user experience on devices with varying screen sizes.

Design elements and User experience

I. Visual appeal

Emotis is expected to bring a playful, vibrant and amiable atmosphere to the users, since children are target audiences of the game. Hence, the visual elements are designed to fit the theme, in both dark mode and light mode.

1. **Fonts**

Raleway is used to deliver all the text content of the game, since it gives a neat and organized look. The primary aim of choosing this font is to ensure a high level of readability of important information, thereby enhancing user experience. Secondly, as other visual elements in the game tend to have bright colors and lively vibes, this clean font style brings a balance to the screen without overpowering other elements.

2. Color schemes

a. Light theme

The light theme scheme consists of warm and bright tones inspired by the colors of the sunrise. The soft pink serves as the primary color of the theme, giving a lighthearted, warm, and cheerful feel. Hence, this color is applied to primary buttons and important content. Other light shades of pink also act as secondary and tertiary colors to balance the soft pink and other visual elements. Meanwhile, the muted teal shades act as accent colors, adding an element of freshness and energy without being overly intense. Therefore, this type of color is applied to highlighted buttons and text. Finally, the light, neutral background brings a gentle tone to make other elements outstanding. Overall, the light theme design aims to give a fresh, energetic, and warm look, like the beginning of the morning.

b. Dark theme

In marked contrast, the dark theme is inspired by the night. As expected, it delivers a cool, calm, and mysterious atmosphere. The teal color is the primary color, making vital elements noticeable in a deep, dark blue background. However, with the soft, pale purple as the accent color, the views become vibrant and attractive to users.

3. Layouts

Since the colors used in Emotis and the character images are colorful and bright, the app's layout is expected to follow a minimalist style so that the players are not overwhelmed with complex and strong-colored designs. With that principle in mind, important sections in the game are covered with rectangular shapes, with titles and indicators to guide the player through the experience. The buttons are highlighted with shadow, and the general layout is organized mainly in vertical pattern so that the whole design looks neat and organized. For the game board, the elements are organized in a grid, with light-colored tiles displayed in the background to make these elements stand out. Overall, the layout is believed to enhance the visual appeal in combination with colors and font styles.

II. Consistency

1. Fonts and Colors

Regarding the font, a single font style is applied throughout the application, ensuring that the players are not confused with different styles. With the same font style, the text still follows a hierarchical order as the titles and body text are distinguishable by font size, colors, and boldness.

About color schemes, the chosen colors are applied uniformly across the app. If the app is switched to a dark theme, all the views, including sheets and modals, are displayed accordingly. Moreover, the application consistently applies primary colors to primary buttons, secondary colors for less important elements, and accent colors to highlight important parts.

2. Imagery

In terms of imagery, all the images have 1x, 2x, and 3x scale displays, making sure that the elements are visible in larger screen sizes. The images of elements in the grid should maintain similar width and height, so as the achievement badges. Additionally, all the images are of high quality to maintain an engaging look.

Regarding icons and buttons, they should follow similar styles and be large enough for users to notice and interact with.

III.User-friendly interface and user-centered design

Le Tonneau is expected to be intuitively used even by novice users. That concern is addressed by analyzing two aspects, usability considerations and prioritizing user needs.

1. Prioritizing user needs

The design elements in Emotis align with the Human Interface Guidelines for game design (Apple Developer 2024g). The analysis of how the application prioritizes user needs is divided into two aspects: Design and Functionality.

a. Design

Navigation: The app's layout is designed with simplicity and clarity where the navigation elements are outstanding and straightforward, ensuring that users can easily navigate through different sections. Additionally, the bright, distinct colors help to visually differentiate the interactive elements from others and guide the users through the interface without overwhelming them

Responsiveness: The app's design adapts seamlessly to different screen sizes. This approach ensures that users have an optimal experience regardless of device, which is a primary principle of Apple's guidelines.

Iconography and Typography: Icons used in the app are taken from SF Symbols. The icons are large enough and straightforward to understand.

Typography: For typography, the use of Raleway font brings a balance between readability and aesthetic appeal. The text sizing is also scalable based on UIScreen dimensions, ensuring that the text content remains legible across all devices. Moreover, the text size is calculated carefully to ensure they are above 11pts for readability on iPhone and iPad. Finally, the text follows a hierarchical order with different font sizes so that the users can distinguish between body text and headings.

b. Functionality

Customization: The app allows users to customize their experience with different themes, skins, language, and play modes. Hence, the app can meet diverse user preferences, enhancing user satisfaction and engagement.

Performance optimization: Despite some known performance issues, the app is designed to provide smooth gameplay where possible.

2. Usability considerations

The usability of Emotis will be analyzed following Usability Heuristics for game design (NN/g 2024).

Aesthetic and minimalist design: Emotis follows a simple layout for different views. All the components placed in the app have purposes, with minimal unnecessary elements.

User control and freedom: The application offers a customized experience for users in various aspects, such as themes, languages, and character sets. Additionally, the players can always exit a view and resume the previous game upon opening the app.

Consistency and standards: Emotis maintains consistent use of fonts, colors, and icons throughout the app, regardless of themes and devices. This consistency should help players feel comfortable during the gameplay and reduce the learning curve.

According to this analysis, Emotis is believed to be a relatively user-friendly and user-centric design app.

Conclusion

The development of Emotis has been a rewarding journey for me as a software developer. First and foremost, I got firsthand experience to dive into the game logic, and it surprised me that I could re-build my favorite game given my learning experience. This process motivated me to sharpen my skills and apply what I learned to real-world projects to be ready for my future career. Regarding technical development, I recognize the importance of application functionality and performance. Despite that, there would be some cases where we need to sacrifice one aspect to provide a more comprehensive experience for the user. This notion has developed my decision-making skills in future projects. Finally, I learned that the primary goal of creating an application,

either a list app or a game, is to provide a user-friendly interface that meets diverse user preferences. Hence, usability aspects should always be re-evaluated throughout development, not just in the final phase.

In future development, there are several aspects that Emotis can be refined to provide a more satisfying gaming experience for players. Nevertheless, my primary focus is to enhance the application performance, which is one of the known problems of the game. This process can be divided into three phases:

- Phase 1: Conduct a comprehensive performance audit to identify the issues in the code.
- Phase 2: Optimize the loading process and UserDefaults management to reduce game freezes.
- Phase 3: Implement a more flexible game state management to resolve this issue, ensuring smooth game operations.

References

Anastasia (2024) *The Evolution of Match-3 Games: Genres of Classic Formula Fresh*, *EJAW Game Development Studio*, accessed 3 September 2024, https://ejaw.net/match-3-games-why-should-you-develop-them/

Apple Developer (2024a) *SpriteKit*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/spritekit/

Apple Developer (2024b) *SKScene*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/spritekit/skscene

Apple Developer (2024c) *Getting Started with Sprite Nodes*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/spritekit/skspritenode/getting_started_with_sprite_node s

Apple Developer (2024d) *ObservableObject*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/combine/observableobject

Apple Developer (2024e) *Swift Charts*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/charts

Apple Developer (2024f) *Localizing and varying text with a string catalog*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/documentation/xcode/localizing-and-varying-text-with-a-string-catalog

Apple Developer (2024g) *Designing for games*, *Apple Developer Documentation*, accessed 3 September 2024, https://developer.apple.com/design/human-interface-guidelines/designing-for-games

GeeksforGeeks (2016) *Singleton Design Pattern*, *GeeksforGeeks*, accessed 3 September 2024, https://www.geeksforgeeks.org/singleton-design-pattern/

Hudson P (2019) *How to loop audio using AVAudioPlayer and numberOfLoops*, *Hacking with Swift*, accessed 3 September 2024, https://www.hackingwithswift.com/example-code/media/how-to-loop-audio-using-avaudioplayer-and-numberofloops

Hudson P (2022) *How to integrate SpriteKit using SpriteView*, *Hacking with Swift*, accessed 3 September 2024, https://www.hackingwithswift.com/quick-start/swiftui/how-to-integrate-spritekit-using-spriteview

Jacobson J (2017) *A Guide to SpriteKit Actions*, *Medium*, accessed 3 September 2024, https://medium.com/hackernoon/a-guide-to-spritekit-actions-c20b079f5398

Murugan K (2015) *How to use touchesBegan, touchesMoved, touchedEnd using swift in iOS*, *Wordpress*, accessed 3 September 2024, https://ktrkathir.wordpress.com/2015/08/06/how-to-use-touchesbegantouchesmovedtouchedend-using-swift-in-ios/

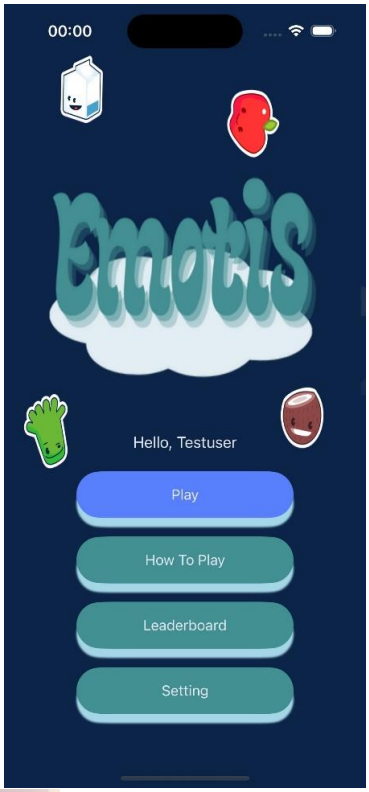
Nielsen Norman Group (NN/g) (2024) *Ten usability heuristics*, NN/g website, accessed 3 September 2024. https://www.nngroup.com/articles/ten-usability-heuristics/

Appendix

Figure 11. Welcome view

Figure 12. Dark theme

Figure 13. How to play



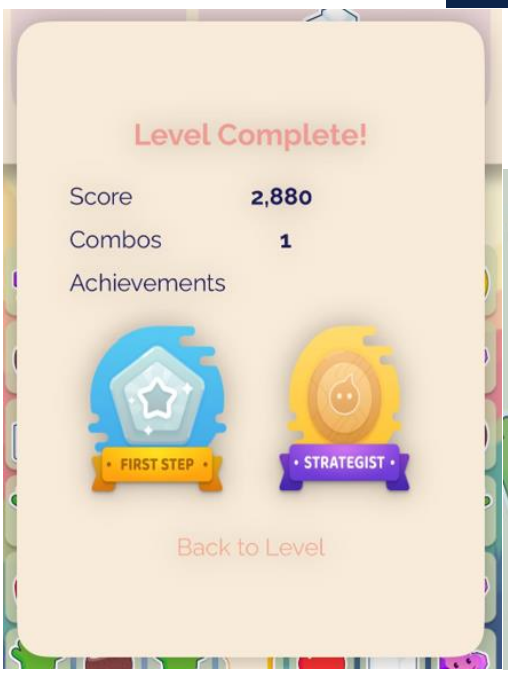


Figure 14. Level complete modal

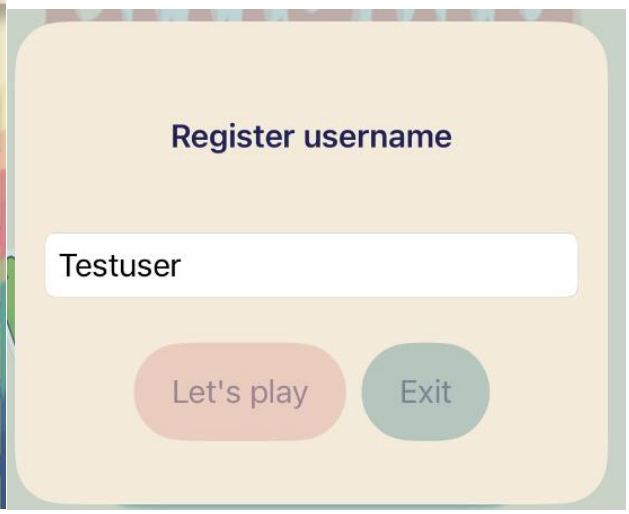


Figure 15. Register username

Figure 16. Level view

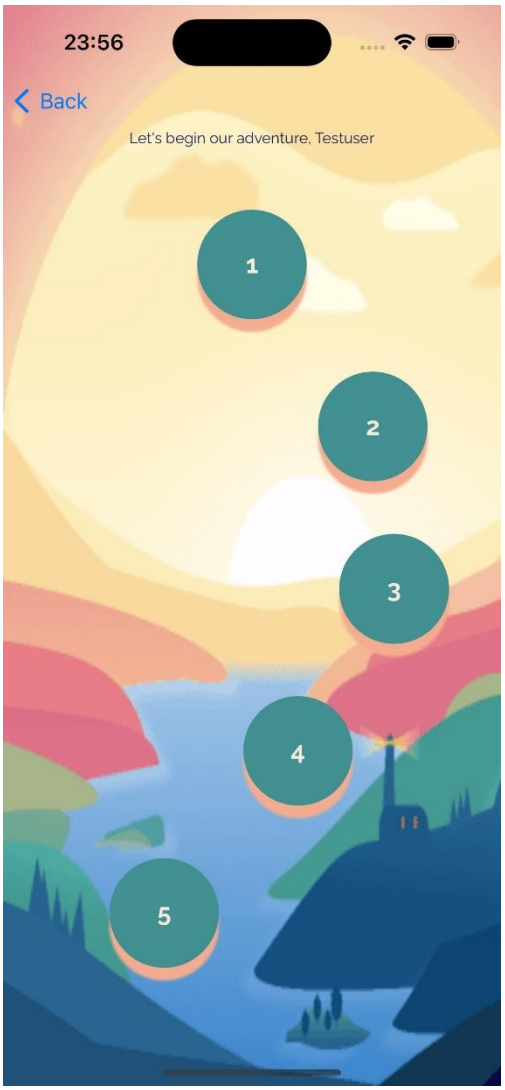


Figure 17. Setting view