Article

Adapting your game interface for smaller screens

Make text legible on all devices the player chooses to run your game on.

Overview

Great games create engaging, immersive experiences through thoughtful design. Develop a game that's responsive and easy to use by considering all aspects of design, including display sizes, device orientation, and text legibility. It's crucial to plan this phase as early as possible and allocate sufficient resources and time within your team to ensure effective adaptation. For platform-specific design guidance, see Human Interface Guidelines > Platforms.

Menus and labels — which can display a combination of text, symbols, or interface icons — are ubiquitous throughout a game's interface, so most players already know how to use them. Utilizing menus and labels consistently in your game can help make your experience feel familiar and easy to learn.

Consider the best practices below to ensure that your in-game menus and UI components interact well on all platforms:

- Adopt consistent layouts that adapt to various contexts.
- Render adaptable menus and labels.
- Handle multiple interaction methods.
- Test on all supported devices.

In addition to menus and labels, take advantage of high-resolution displays, Apple silicon, external game controllers, headphones, and speakers. To learn more about creating games for Apple platforms, see Games Pathway.

Render adaptable menus and labels

Apple devices have a variety of screen characteristics, such as sizes, resolutions, and aspect ratios. When people download your game from the App Store, they expect to enjoy it on all their devices, whether they use iPhone SE or iPad Pro. Making the menus and labels in your game look and work great on every screen is important.

When using Apple devices, people can change the text size of the whole system via the Settings app, and they expect the text in your menus and labels to stay consistent with the setting. Make sure the text in your menus and labels adapt to different settings.

The system-provided UI frameworks, such as SwiftUI, UIKit, and AppKit, have the capability to automatically lay out UI elements based on different screen characteristics, and can automatically adjust the layout for right-to-left languages such as Arabic and Hebrew. An app's UI that you implement with these frameworks automatically adapts to different screens and languages.

Whenever possible, use the menus and labels the UI frameworks provide to implement your app's UI, and configure them in the following way so they automatically adapt to the platforms:

- Adopt the layout mechanism the UI frameworks provide to let the system position your menus and labels. Avoid using fixed-size UI elements. For more information, see Human Interface Guidelines > Layout.
- Adopt text styles when setting up your menus and labels. Avoid using fixed-size fonts. For more information, see Scaling Fonts Automatically.

For apps that draw menus and labels with code or implement them using middleware, make the menus and labels adaptable by doing the following:

- Streamline design for smaller interfaces. Start with essential functionality for compact devices. Utilize the advantages of larger screens and enhanced capabilities to enrich the user experience with additional features and access options.
- Make your menus and labels adapt to different device orientations, if your app supports both portrait and landscape mode.
- Use aspect ratios to maintain proportion. Consider which UI elements need to maintain a consistent aspect ratio, preserving their width-to-height proportions across different screen sizes and resolutions.

When drawing text for your menus and labels:

- Choose an appropriate font and size so the text is comfortably legible. For more information, see Human Interface Guidelines > Typography.
- Update the text size when the user changes the preferred content size.
- Make the text no smaller than the recommended minimum text size for each platform so it's readable on all devices.

The system uses points to measure font size. *Points* are a logical unit independent from the physical screen size and resolution. If necessary, use the following code to convert a font size from points to drawable pixels:

Swift Objective-C

```
func sizeOfFontInDrawablePixels(with layer: CAMetalLayer, fontSizeInPoints: CGFloat)
    // Calculates how many drawable pixels are in one abstract point.
    // Assumes the CAMetalLayer.contentsGravity is set to its default value of 'resi
    let pixelsInPoint = Double(layer.drawableSize.width / layer.bounds.size.width)
    let fontSizeInPixels = pixelsInPoint * fontSizeInPoints
    return fontSizeInPixels
}
```

Handle multiple interaction methods

Consider how your game handles interactions with different input methods, like mouse, keyboard, game controllers, touch, and spatial user interactions. Take advantage of the Game Controller framework to create a unified approach to handling these interactions for your players. To learn more about receiving controller input by either polling or callbacks, see Handling input events.

Below are considerations for smaller devices:

- Consider menu navigation element size. Ensure that these elements are large enough, especially for touch input, even if the proportions remain unchanged.
- Use accessibility APIs and developer tools. Deliver high-quality experiences for everyone, including players with disabilities.

Take advantage of built-in accessibility features like Multi-Touch, VoiceOver, Switch Control, Guided Access, Text to Speech, closed-captioned or audio-described video, and more. See Accessibility for more information about building accessible games on Apple platforms.

Test on all supported devices

Use a physical device to test the behavior and user experience for your game's menus on every supported device. Otherwise, use Simulator or an approximated environment on Mac, with proper display resolution, scale, and interaction methods. For more guidance on testing your game in Simulator, see the resources below:

- Running your app in Simulator or on a device
- Testing in Simulator versus testing on hardware devices

- Testing complex hardware device scenarios in Simulator
- Identifying graphics and animations issues in Simulator

If you use middleware in your games, follow their guidelines on testing multiple form factors.

See Also

Presentation

Managing your game window for Metal in macOS

Set up a window and view for optimally displaying your Metal content.

Managing your Metal app window in iPadOS

Set up a window that handles dynamically resizing your Metal content.

Onscreen presentation

Show the output from a GPU's rendering pass to the user in your app.

HDR content

Take advantage of high dynamic range to present more vibrant colors in your apps and games.