VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SWIFTUI KOMPONENTA PRO VÝBĚR STRUKTUROVANÉHO VSTUPU OD UŽIVATELE NA PLATFORMĚ IOS

SWIFTUI USER INTERFACE COMPONENT FOR STRUCTURED USER INPUTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠIMON STRÝČEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Strýček Šimon**
Program: Informační technologie
Název: **SwiftUI komponenta pro výběr strukturovaného vstupu od uživatele na platformě iOS**
SwiftUI User Interface Component for Structured User Inputs
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte programování aplikací pro platformu iOS/macOS. Prostudujte koncepci komponent pro výběr hodnoty uživatelem (picker, slider apod). Zaměřte se na komponenty pro výběr číselné hodnoty z daného intervalu.
2. Navrhněte komponentu pro efektivní a komfortní uživatelský vstup založený na komponentě slider (lineární posuv po zadané škále, kombinace výběrové škály v jednom a dvou dimenzích). Zaměřte se na přesnost zadání hodnoty (např. dynamickou rekonfigurací měřítka škály). Předpokládá se kreativita v návrhu koncepce UI.
3. Implementujte komponentu ve SwiftUI. Zaměřte se na robustní návrh API (datasource, delegate, konfigurace vizuální stránky komponenty).
4. Demonstrujte komponentu v několika aplikacích, přinejmenším v aplikaci pro přehrávání videa, kde slider slouží pro přesné posouvání ve videu.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato práce se zabývá návrhem a implementací komponenty uživatelského rozhraní pro lineární výběr hodnoty v daném intervalu. Návrh komponenty je prováděn s ohledem na jednoduchou volbu číselné hodnoty a se zaměřením na vysokou přesnost výběru. Samotná komponenta je koncipována pro možnost výběru jak jednodimenzionální, tak i dvoudimenzionální hodnoty. Implementace je prováděna se zaměřením na platformu iOS s využitím technologie SwiftUI od společnosti Apple. Výsledkem práce je funkční balíček Swift Package komponenty s robustním rozhraním.

Abstract

This thesis deals with the design and implementation of user interface control for the linear selection of values at a given range. The component design considers an intuitive way of selecting numerical values with an emphasis on the high accuracy of its selection. The component itself is designed for selecting single-dimensional and two-dimensional values. The implementation is made for the iOS platform using the SwiftUI framework, made by Apple. The result is in the form of a Swift Package library with a robust interface.

Klíčová slova

SwiftUI, iOS, posuvník, komponenta, uživatelské rozhraní, knihovna, přesnost výběru

Keywords

SwiftUI, iOS, slider, component, user interface, library, accurate selection

Citace

STRÝČEK, Šimon. *SwiftUI komponenta pro výběr strukturovaného vstupu od uživatele na platformě iOS*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

SwiftUI komponenta pro výběr strukturovaného vstupu od uživatele na platformě iOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Hrubého, PhD. Další informace mi poskytli Ing. Filip Klembara a Ing. Michal Tomlein z firmy IN2CORE. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Šimon Strýček
9. května 2022

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Martinu Hrubému, PhD. za organizaci předmětu Programování zařízení Apple, dále bych rád poděkoval členům společnosti IN2CORE za aktivní podporu a odbornou pomoc.

Obsah

Kapitola 1

Úvod

Co se týče vývoje programů a aplikací na mobilní zařízení, existuje řada technologií pro různé platformy s podporou pro tvorbu unikátního obsahu. Takovéto technologie nabízí základní stavební bloky pro vytvoření vlastního uživatelského rozhraní dle představ vývojáře, přesto tyto sady nejpotřebnějších komponent pro tvorbu těchto rozhraní nemusí být vždy dostačující. Přestože je v praxi běžná improvizace v podobě vytváření vlastních grafických funkčních prvků, mít dostatečnou výbavu klíčových komponent dokáže značně urychlit takovýto vývoj.

Jednou z takto klíčových položek je komponenta pro výběr číselné hodnoty v daném intervalu. Ve většině nejpoužívanějších technologií je dnes dostupná právě taková komponenta v podobě zvyklého konceptu posuvníku s kulatým ukazatelem. Tato komponenta je pro většinu případů užití dokonalým adeptem, ovšem její nedostatky tkví v přesnosti výběru uživatelem požadované číselné hodnoty. Představme si klasické využití komponenty v podobě zvyklého posuvníku s použitím dotykového mobilního zařízení. Vezměme si jako příklad aplikaci pro úpravu a stříhání videí. V případě úpravy délky videa se meze pro střih volí pomocí komponenty posuvníku, jehož rozsah je určen délkou přehrávaného videa. Délka videa je v tomto případě sice proměnná, délka samotného posuvníku nikoli. Problém s přesností volby nastává v případě, kdy použijeme obsah o delší době trvání, což zapříčiní nemožnost uživatele správně zvolit požadovanou pasáž. Právě tento nedostatek je jednou z hlavních motivací této práce, jejímž cílem je rozšířit sadu dostupných komponent grafických uživatelských rozhraní pro platformu *iOS* o komponentu zaměřenou na přesnost výběru nejen jednorozměrných, ale i dvourozměrných hodnot.

Původní myšlenka tématu vzešla z oblasti profesionálních nástrojů pro práci s multimédii. Konkrétně vychází z frustrace vývojářů firmy *IN2CORE*, která se zabývá vývojem aplikací pro platformu *iOS* právě na toto téma. Žádná z aktuálně dostupných komponent na této platformě nenabízí možnosti dostatečné přesnosti se zachováním přívětivosti komponenty a jednoduchostí integrace do již existujících projektů, přestože se nejedná o příliš ojedinělé požadavky.

Výsledným produktem je tedy plnohodnotný balíček s komponentou uživatelského rozhraní využívající novou technologii *SwiftUI*. Tato technologie byla zvolena se záměrem udržovatelnosti komponenty s jejím následným rozvojem. Balíček ovšem stále obsahuje implementaci rozhraní pro snadné použití v aplikacích, využívajících nejen tuto technologii, nýbrž i jejího předchůdce.

Kapitola 2

Studium problematiky a průzkum

Před započetím samotného návrhu vlastního provedení komponenty bylo zapotřebí se seznámit s vývojem aplikací na platformu iOS a provést průzkum podobných řešení. V následujících kapitolách budou popsány důležité aspekty vývoje právě takovéto komponenty. Mimo studii technologií pro tento vývoj je zde popsán také průzkum existujících komponent pro obdobný výběr hodnot. Nalezené komponenty jsou zde kategorizovány a jejich vlastnosti zanalyzovány.

2.1 Studium problematiky

Implementaci komponenty předcházelo také studium problematiky programování aplikací pro platformu iOS. A to jak základy jazyka *Swift* a vývoje aplikací na tuto platformu, tak i základní principy nakládání s novější technologií *SwiftUI* a jejím předchůdcem, frameworkem *UIKit*. Podstatný základ do této studie poskytl převážně již absolvovaný kurz Programování zařízení Apple, vedený p. doktorem Martinem Hrubým.

Prostředky pro vývoj aplikací na platformu iOS

Mezi seznamování se se samotnými technologiemi pro vývoj uživatelských rozhraní a aplikací na platformu iOS ze stránky implementace bylo zapotřebí prozkoumání dostupných nástrojů pro jejich tvorbu. Použitým vývojovým prostředím je v tomto případě vývojové prostředí *XCode*, dostupné na zařízeních s operačním systémem macOS, které je výchozí pro vývoj aplikací na všechny podporované platformy společnosti Apple.

Mimo nástrojů pro vývoj aplikací bylo nutné se seznámit mimo jiné i s dostupnými prostředky pro testování projektu a jeho ladění. K těmto prostředkům patří výchozí aplikace *Simulator*, sloužící k emulaci zařízení s operačním systémem iOS. Dále prostředí *Live Preview*, dostupné jakou součást integrace technologie *SwiftUI* do vývojového prostředí *XCode*, poskytující živé interaktivní zobrazení komponent implementovaných pomocí této technologie. Následně také nástroj *Instruments*, zpřístupňující řadu nástrojů pro profilování výsledného projektu a získávání informací o náročnosti řešení na konkrétní prostředky zařízení pro snadnější optimalizaci implementace. *Instruments* je rovněž výchozí dostupný nástroj pro profilování aplikací, dostupný na platformě macOS.

Technologie pro vývoj GUI na platformě iOS

Hlavním předmětem studie vývoje aplikací na platformu iOS byl dříve zmiňovaný framework *SwiftUI*, jenž představila společnost Apple v roce 2019. Důvod volby této technologie je lepší udržovatelnost v průběhu jejího vývoje jakožto budoucího nástupce technologie *UIKit*. Dalším důvodem je potenciál v možnosti aplikovat komponentu také na novějších zařízeních platformy *macOS* či *watchOS* [9]. Pro správné porozumění toku dat a celé filozofie SwiftUI bylo zapotřebí prostudovat prostředí frameworku *Combine*.

Framework *Combine* při tvorbě aplikací za pomocí technologie SwiftUI hraje nepostradatelnou roli. Sama technologie staví na konceptu tvorby podoby grafického rozhraní okolo svého stavu. Tato implementace je ovšem možná s použitím frameworku *Combine*, který zajišťuje zaobalení těchto stavových proměnných do speciálního prostředí, poskytující možnost zasílání asynchronních notifikací o změnách těchto hodnot, a tím i okamžitou reakci s případným překreslením jednotlivých prvků grafického rozhraní v případě jejich změny [5].

Mezi další podstatná témata studie patřila technologie *UIKit* jakožto předchůdce samotného *SwiftUI*. Hlavní motivací studia tohoto tématu bylo řádné seznámení se s tvorbou uživatelských rozhraní pomocí této metody za účelem vytvoření co nejpřirozenějšího prostředí pro uživatele balíčku v aplikacích využívající právě tuto technologii. Další motivací pro toto studium bylo lepší pochopení motivací a záměrů za vývojem dříve zmiňovaného frameworku SwiftUI, který je postaven právě na základech jeho předchůdce.

Součástí seznamování se s těmito technologiemi pro tvorbu uživatelských rozhraní bylo také prozkoumání zvyklostí v implementaci takovýchto aplikací. Mezi tyto zvyklosti patří převážně návrh rozhraní pro práci s komponentami příslušného frameworku. Některé z důležitých prvků jsou například způsob stylování takovýchto komponent, volba výchozích hodnot a předávání dat mezi nimi. Hlavním zdrojem v tomto případě byl převážně průzkum výchozích komponent, dostupných v těchto technologiích, ale mimo jiné i průzkum již existujících řešení.

Mimo obeznámení se se samotnými nástroji pro vývoj grafického rozhraní pro platformu iOS, patřilo mezi nepostradatelné části studia také seznámení s programovacím jazykem Swift a architekturou výchozí komponenty pro výběr číselné hodnoty z určeného rozsahu.

Podstatné rozdíly technologií SwiftUI a UIKit

Důraz při tomto průzkumu byl kladen na rozdíly konceptů obou frameworků. Velikým rozdílem je zde mimo jiné způsob vzájemného předávání dat jejich součástí. Co se týče frameworku UIKit, zde je předávání dat povětšinou realizováno pomocí návrhových vzorů *DataSource* a *Delegate*. Předávání takovýchto dat má podobu funkčního bloku pro zpětné volání, či implementace metod protokolů, příslušícím dříve zmiňovaným vzorů. Na druhou stranu za tímto účelem framework SwiftUI naopak používá právě technologii *Combine*, která tuto implementaci zajišťuje pomocí tzv. *Publishers*.

Dalším významným rozdílem těchto dvou technologií je také jejich celková architektura. Zatímco UIKit je postaven na principu detailního popisu postupu vykreslování jednotlivých částí uživatelského rozhraní, framework SwiftUI je naopak deklarativního typu [15]. Určuje se zde, co a kde se má zobrazit, ovšem konstrukce rozhraní je již implicitní a provádí ji skrytá implementace frameworku, postavená mimo jiné i na implementaci svého předchůdce.

V neposlední řadě je podstatným rozdílem mezi těmito technologiemi architektura z ohledu použitých návrhových vzorů. Zatímco framework UIKit staví převážně na návrhovém vzoru *Model-View-Controller (MVC)* [1], SwiftUI na druhou stranu využívá dnes používanější návrhový vzor *Model-View-ViewModel (MVVM)* [13] [10]. Rozdíl mezi těmito

dvěma návrhovými vzory tkví také v odpovědnosti jejich jednotlivých složek. Složka *Controller* návrhového vzoru *MVC* zastává, jak z názvu vypovídá, roli kontroléru, jenž přímo provádí změny v části *Model*, a dle těchto změn mění také zobrazení aplikace. Na druhou stranu složka *ViewModel* návrhového modelu *MVC* v tomto případě zastává spíše role stavového objektu, který tvoří pouze komunikační vrstvu mezi daty a jejím zobrazením.

2.2 Průzkum komponent pro výběr jednorozměrných hodnot

Výběr komponent pro volbu jednorozměrné číselné hodnoty byl zaměřen převážně na použití v mobilních aplikacích. Tato skutečnost uvádí mezi hodnocené faktory také využití prostoru displeje mobilních zařízení a jednoduchost používání komponenty s využitím dotykového displeje.

Posuvník s kulatým ukazatelem

Klasický, dnes nejpoužívanější, posuvník pro pseudo-spojitý výběr hodnot, je uživatelsky přívětivou variantou pro výběr číselné hodnoty v požadovaném rozsahu. Za cenu jeho jednoduchosti má ovšem toto provedení své limitace.

Výhody:

- zvyklé provedení – uživatel ví, co od komponenty očekávat
- ve většině frameworků je součástí základního balíčku komponent
- velikost ukazatele usnadňuje použití na dotykových zařízeních

Nevýhody:

- pevně dané rozmezí komponenty – méně přirozený výběr čísel s periodickým oborem hodnot jako např. úhel rotace obrázku (ekvivalence hodnot 360° a 0°)
- omezená přesnost výběru dle vymezeného rozsahu (při rozmezí v řádech tisíců nelze volit hodnotu s přesností na několik desetinných míst)

Obrázek 2.1: Ukázka výchozího posuvníku, dostupného ve frameworcích pro tvorbu uživatelských rozhraní na platformu iOS.

Posuvník s pohyblivou osou

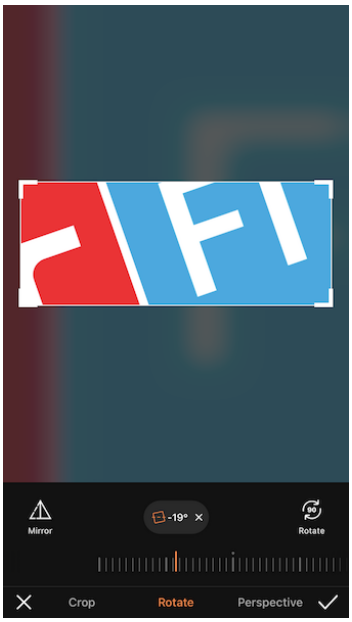
Posuvník s fixním ukazatelem a pohyblivou osou bývá proveden v podobě výrazného, přesto nerušivého, ukazatele a osy s vyznačenými stupni v podobě běžného pravítka. Nerušivý ukazatel bývá většinou přítomen v podobě tenké čáry kontrastní barvy umístěné na středu této komponenty. Samotná pohyblivá osa nebývá celá viditelná a vizuálně překračuje hranice zobrazení komponenty. Proto je ideálním adeptem pro výběr úhlů, jelikož je možné snadno vizuálně simulovat efekt nekonečnosti osy.

Výhody:

- nerušivý ukazatel příliš nezakrývá vybíranou hodnotu, výběr je přesnější než u předchozích variant
- palec při výběru na dotykovém displeji nepřekrývá vybíranou hodnotu
- jednoduchá realizace efektu nekonečného rozsahu

Nevýhody:

- osa není celá vidět, a proto nemusí být na první pohled zjevný její rozsah
- komponenta má stále fixní přesnost výběru, při větším rozsahu je potřeba zvolit kompromis požadované přesnosti a rozměru pohyblivé osy



Obrázek 2.2: Posuvník s pohyblivou osou v aplikaci AirBrush.[1]

[1] Aplikace AirBrush dostupná zdarma v obchodě AppStore https://apps.apple.com/us/app/airbrush-best-photo-editor/id998411110

Kruhový posuvník

Jedná se o velmi podobnou variantu posuvníku v porovnání s dříve zmíněnou verzí o pohyblivé ose. V tomto provedení má ovšem tato osa podobu kruhovitého kotouče a není zde opět celá viditelná. Je zobrazen pouze její výřez, který ve většině případů vyčnívá ze spodní části obrazovky zařízení.

Výhody:

- napomáhá vizualizaci rotace a nekonečnosti, což může působit přirozeněji při volbě specifických parametrů (např. přiblížení fotoaparátu)
- jistá úroveň dynamiky přesnosti volby – při vzdálení se prstem od pomyslného středu kruhu (osy) má výběr větší přesnost

Nevýhody:

- rozměry komponenty můžou být limitující na menších displejích mobilních zařízení
- oproti její nezaoblené variantě, tkví výhody této komponenty pouze v omezeném množství případů užití

Obrázek 2.3: Posuvník se zaoblenou pohyblivou osou v aplikaci pro úpravu fotografií Focos.[3]

[3] Aplikace Focos https://apps.apple.com/us/app/focos/id1274938524

Výběr pomocí klávesnice

Varianta výběru požadované hodnoty za pomocí klávesnice může mít opodstatnění na zařízeních s fyzickou klávesnicí. Co se mobilních aplikací týče, bývá většinou přítomna pouze jako kompenzace limitací běžných komponent pro výběr číselných hodnot. Tato varianta přístupu k výběru sice teoreticky poskytuje jeho neomezenou přesnost, ovšem zavádí do řešení další problémy jako např. nutnost validace platnosti hodnoty. Samotná validace hodnoty poté často spíše komplikuje rozhraní a činí jej náročnější na použití.

Výhody:

- teoreticky neomezená přesnost výběru

Nevýhody:

- na mobilních zařízeních je nutnost zadávání dat z klávesnice spíše komplikací
- nutnost validace vstupní hodnoty a explicitní kontroly mezí
- nevyužití potenciálu dotykových zařízení

Obrázek 2.4: Ukázka doplnění komponenty pro výběr číselné hodnoty použitím vstupu z textového pole, viz nástroj pro otáčení obrázků.[5]

[5]Nástroj pro otáčení obrázků https://pinetools.com/rotate-image

2.3 Průzkum způsobu realizace výběru vícerozměrných hodnot

Průzkum způsobu provedení komponent pro výběr vícerozměrných hodnot byl prováděn na širším spektru oborů aplikací a programů, než průzkum komponent pro výběr jednorozměrné hodnoty, a to zejména z důvodu menší četnosti nástrojů tohoto typu. Nejčastější případy užití výběru takovýchto hodnot byly realizovány za pomocí více komponent pro výběr jednotlivých složek.

Samostatná komponenta pro výběr jednotlivých složek

Jedná se o nejjednodušší a zároveň nejméně atraktivní řešení. Výběr často bývá doprovázen vizualizací výsledné kombinace jednotlivých složek. Realizace výběru hodnoty tímto způsobem zanáší nutnost interakce s celkovým řešením vícenásobně, a to pro každou její složku zvlášť. Pro výběr více než dvoudimenzionálních hodnot je ovšem toto řešení ve většině případů nevyhnutelné.

Výhody:

- nejjednodušší řešení z pohledu implementace
- oddělení výběru do jednotlivých složek může v některých případech dávat větší smysl

Nevýhody:

- výběr dvourozměrných hodnot se dá provést přívětivěji na dvourozměrném zobrazení displeje zařízení
- separace složek vybírané hodnoty
- nutná opětovná interakce pro každou složku hodnoty

Obrázek 2.5: Výběr barev za pomocí separátních komponent pro každou složku modelu RGB v aplikaci Pages. Aplikace je dostupná pro zařízení s operačním systémem iOS a macOS.

Přímá interakce s vizualizací kombinací složek

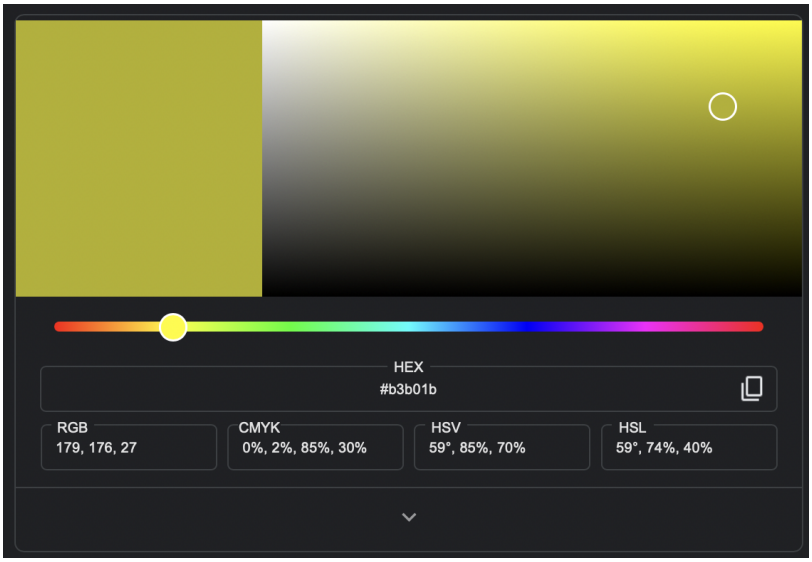
Výběr požadované hodnoty na dvojrozměrné ploše s vizualizovanou kombinací jejích složek. Nejčastěji je výběr uskutečněn pomocí kruhového ukazatele, který indikuje zvolenou hodnotu po provedení výběru dotekem/kliknutím. Nejčastěji bývá tato komponenta dostupná v podobě barevné palety s názvem *ColorPicker*.

Výhody:

- uživatel má přehled o tom, jak bude výsledná kombinace vypadat ještě předtím, než si ji vybere
- často již dostupná komponenta v základních balíčcích frameworků pro tvorbu uživatelských rozhraní

Nevýhody:

- při použití na dotykovém displeji prst zakrývá vybíranou hodnotu, což znesnadňuje její přesnější výběr
- chybí jakákoli dynamika volby přesnosti, při rozsáhlejších intervalech složek je přesnost omezena velikostí displeje



Obrázek 2.6: Komponenta pro výběr barev dostupná ve vyhledávači Google.

Interaktivní zobrazení s využitím gest

Způsob provedení komponenty nejčastěji používaný jako forma virtuálních prohlídek, či zobrazení fotografií v galeriích. Komponenta se skládá pouze z vizualizovaných dat. Výběr a veškerá interakce probíhá formou gest, a to nejčastěji gestem potažení pro manipulaci s vybíranou hodnotou a gestem roztažení dvou prstů (v případě ovládání za pomocí myši také jejím kolečkem) pro zjemnění měřítka složek. Výběr bodové hodnoty ovšem nebývá jednoznačný, jelikož vizualizace obsahuje širší rozsah kombinací hodnot a samotná komponenta povětšinou neposkytuje informaci o umístění bodu výběru v této ploše. Přesnost v tomto případě nebývá hlavní motivací pro tuto variantu. Doplnění ukazatelů či jiných pomocných prvků problém nejednoznačnosti jednoduše řeší.

Výhody:

- jednoduché provedení – není přítomen žádný rušivý element, který by zakrýval vizualizaci
- využití potenciálu gest dotykových zařízení
- vizualizace uživateli napomáhá s výběrem požadované hodnoty

Nevýhody:

- nejednoznačné označení vybírané bodové hodnoty – je zobrazeno širší spektrum kombinací hodnot a bez pomocných ukazatelů (např. použití křížku umístěném na středu vizualizace) může být výběr matoucí
- chybí zde označení vybíraných číselných hodnot – přesnost výběru tedy závisí pouze na „oku uživatele”

Obrázek 2.7: Virtuální prohlídka okolí Fakulty informačních technologií VUT v Brně pomocí služby Google StreetView.

Použití ovládacího panelu

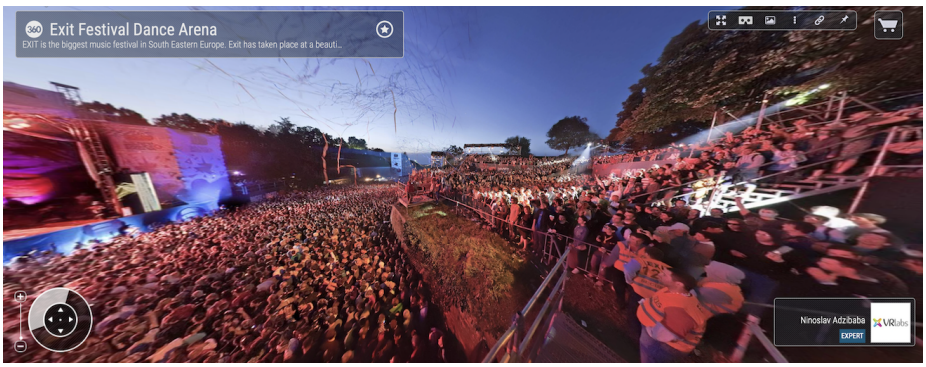
Jedná se o obdobné provedení předchozí varianty, komponenta je tentokrát doplněna specifickým ovládacím panelem. Panel bývá přizpůsoben konkrétnímu použití a často zobrazuje dodatečné informace o vybírané hodnotě pro lepší představu uživatele o jejím významu.

Výhody:

- doplnění vizualizace dodatečnými informacemi o vybírané hodnotě mohou velmi napomoct uživateli při jejich výběru
- ovládací panel může kompenzovat nedostatky zařízení s jinými způsoby ovládání (klávesnice a myš oproti dotykového displeje)

Nevýhody:

- je potřeba vybrat kompromis rozměrů ovládacího panelu – panel by měl co nejméně omezovat prostor vizualizace dat a zároveň poskytovat dostatek dodatečných informací pro snadný výběr



Obrázek 2.8: Ukázka využití ovládacího panelu ve webové aplikaci virtuální prohlídky 360Cities.[7]

[7] Virtuální prohlídky 360Cities https://www.360cities.net

Kapitola 3

Návrh vlastní komponenty

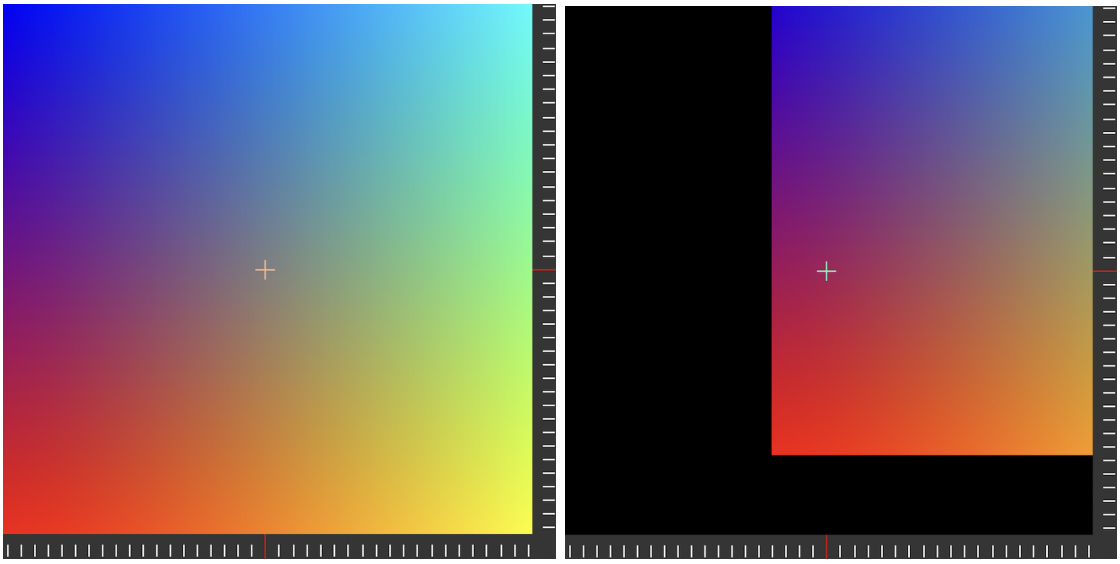
Návrh vlastní komponenty byl založen na analýze existujících řešení pro obdobný případ užití. Klíčové vlastnosti vychází právě z této analýzy a snaží se vyřešit jejich nejčastější nedostatky. Při návrhu byl kladen důraz na původně stanovený cíl přesnosti výběru při použití na zařízeních ovládaných dotykovými gesty na platformě iOS.

Výběr jednorozměrných a dvourozměrných hodnot je rozdělen do dvou separátních komponent. Přes svou separaci je ovšem v obou variantách zakomponován obdobný koncept a platí mezi těmito dvěma komponentami konzistence z pohledu interakce a jejich filozofie. Jednou z dříve nezmiňovaných vlastností, která byla v návrhu brána v potaz, je varianta pro výběr hodnoty z pseudo-nekonečného rozsahu. Případ užití této varianty je volba hodnoty periodicky se opakujících jevů, konkrétním využitím může být například složka odstínu (*hue*) barevného modelu *HSV*.

3.1 Aplikace klíčových poznatků průzkumu

Jelikož hlavním cílem projektu je vytvořit ovládací prvek pro přesný výběr číselné hodnoty, je potřebné eliminovat veškeré faktory, které činí přesný výběr na běžných komponentách obtížný. Prvním, a snad nejdůležitějším faktorem, je samotná interakce s takovou komponentou. Pokud uvažujeme použití právě na dotykovém displeji, nejpodstatnějším faktorem ovlivňujícím přesnost je samotná interakce s komponentou. Ve zvyklém provedení posuvníku se interakce provádí za pomocí potažení ukazatele na požadovanou hodnotu. Samotný ukazatel je poté zakryt prstem po celou dobu interakce, tudíž i vybíraná hodnota je částečně zakryta. V tomto ohledu se osvědčilo právě provedení použité v komponentě posuvníku s pohyblivou osou a nehybným ukazatelem, umístěným na středu celé komponenty.

Podoba ukazatele hraje také nepostradatelnou roli v úrovni přesnosti výběru. Ukazatel by rovněž neměl příliš bránit v pohledu na vybíranou hodnotu, proto je potřebné zvolit kompromis jeho viditelnosti a rozměru. Z tohoto důvodu byla pro roli ukazatele zvolena tenká čára na středu samotné komponenty o kontrastní barvě vzhledem k barvě pozadí osy. Z obdobného důvodu byl v návrhu komponenty pro výběr dvoudimenzionálních hodnot vytvořen kompromis v podobě ukazatele se schopností dynamické rekonfigurace své barvy, jež je možné vidět na obrázku 3.1. Tato barva rovněž odpovídá kontrastu, tentokrát ke zvolené části vizualizace.



Obrázek 3.1: Ukázka návrhu volby kontrastní barvy ukazatele (křížek ve tvaru + v samotném středu komponenty) vzhledem k barvě příslušné oblasti vizualizace.

Přes dříve zmíněná opatření může být volba hodnoty s jemnější přesností stále obtížná. Proto byl návrh obohacen o možnost dynamiky použitého měřítka. Změna měřítka je možná za pomocí gesta sevření či rozevření dvou prstů. V případě varianty komponenty pro výběr dvoudimenzionální hodnoty je toto gesto rovněž bráno v potaz s obdobným efektem na měřítka os. V tomto případě je komponenta navržena tak, aby podporovala změnu měřítka za použití jediného gesta, jak na každé ose zvlášť, tak na obou zároveň.

Se změnou měřítka osy, a mimo jiné pro zpřehlednění výběru hodnoty, návrh komponenty zahrnuje také popisky hodnot jednotlivých stupňů její osy. Pro ještě lepší přehled nad stavem osy jsou její jednotlivé jednotky velikostně rozděleny s ohledem na použité měřítko. Tato vlastnost řeší mimo jiné potencionálně vzniklou nejednoznačnost v případě manipulace s úrovní měřítka.

3.2 Výběr jednorozměrných hodnot

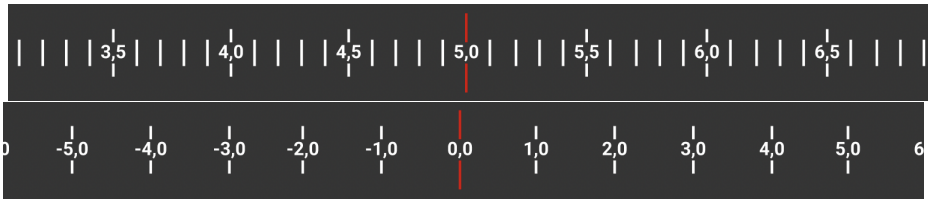
Komponenta pro výběr jednorozměrné hodnoty je postavena primárně na konceptu posuvníku s pohyblivou osou a fixním ukazatelem. Jak bylo dříve zmíněno, důvodem použití tohoto konceptu je odstranění odchylek způsobených překrytím vybírané hodnoty prstem uživatele v průběhu interakce. Komponenta ovšem tuto variantu obohacuje o další prvky hrající klíčovou roli v její přesnosti.

Vzhled a geometrie

Samotný posuvník má podobu obdélníku s vyznačenými stupni připomínající stupně pravítka. Jednotlivé stupně mají podobu úzkých čárek o maximální výšce rovné polovině rozměru tohoto posuvníku. Stupně jsou vertikálně zarovnány k jejímu středu. Obdobnou podobu má ukazatel aktuálně vybrané hodnoty, který je umístěný na středu posuvníku. Tento ukazatel má na rozdíl od zvýrazněných jednotek rozdílnou výšku, a to konkrétně 80 % výšky celého posuvníku. Rozměr ukazatele není jediným rozdílem oproti jednotlivým jednotkám,

dalším rozdílem je také jeho výchozí barva. Volba rozměru a barvy ukazatele má za účel jednoznačně rozlišit tyto dvě součásti posuvníku.

Použité jednotky nemají vždy stejný rozměr, jejich výška je mimo maximální limit omezena také dle toho, jakou hodnotu reprezentuje. Relativně k výchozí nulové hodnotě je každá pátá jednotka označena jako hlavní. Hlavní jednotka má pokaždé maximální možnou výšku (tzn. polovina výšky posuvníku) a je doplněna textovým označením hodnoty. Jednotky, které nejsou označeny jako hlavní, mají výšku stanovenou dle zvoleného měřítka. Při použití výchozího měřítka (1:1) je jejich výška nulová a jednotky nejsou viditelné. Se zvyšující se hodnotou měřítka se zvyšuje také výška těchto jednotek (viz obrázek 3.2). Při dosažení maximální možné výšky se jednotka stává hlavní a je doplněna jejím příslušným popisem.



Obrázek 3.2: Podoba vlastního návrhu komponenty zobrazená v různých úrovních měřítka.

Interakce pomocí gest

Veškerá manipulace s osou, včetně výběru požadované hodnoty, je koncipována pro použití na dotykových displejích za pomocí gest. Hlavními gesty v tomto případě jsou gesto potažení a gesto roztažení dvou prstů. Gesto potažení slouží pro posun pohyblivé osy a tím výběr požadované hodnoty. Gesto roztažení dvou prstů na ose posuvníku zapříčiní změnu měřítka ekvivalentně k délce gesta. Pro zajištění přirozenosti interakce s osou je u gesta potažení očekávána setrvačnost jejího pohybu. Za obdobným účelem byla do návrhu přidána možnost přesahu hranic osy ukazatelem. Přesah osy má v tomto případě pouze estetický význam a má podobu natahování, kterého si lze všimnout např. v aplikaci Safari při obnově aktuální stránky potažením prstem dolů. Po dokončení gesta potažení, kdy ukazatel přesáhl hranici osy, se osa vrátí zpět na úroveň hraniční hodnoty, přičemž výsledná hodnota po dobu celé interakce nepřesáhne tuto hranici (tzn. jde pouze o vizuální efekt, který se neprojeví ve vybírané hodnotě). Tento efekt bude rovněž použit v případě provedení gesta potažení, ukončeného před hranicí osy, jehož setrvačnost zapříčiní přesah hranice.

3.3 Výběr dvourozměrných hodnot

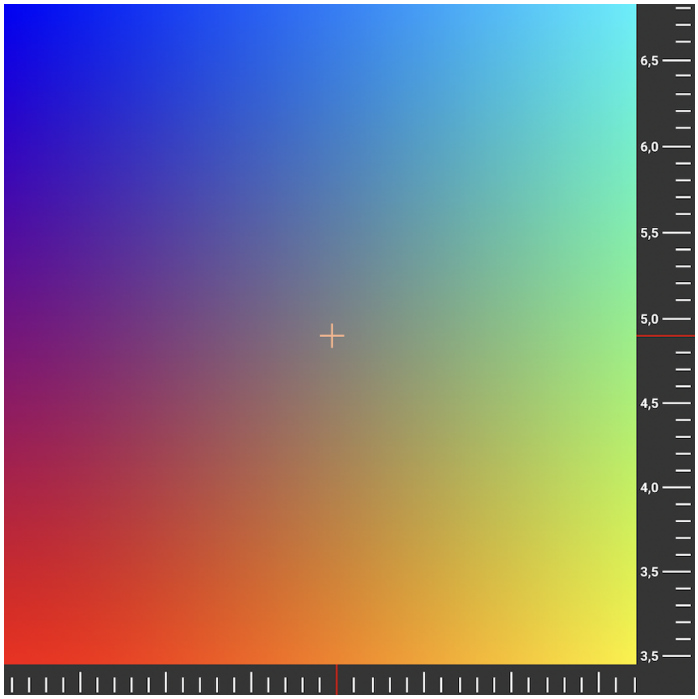
Varianta komponenty pro výběr dvourozměrné hodnoty (2D posuvník) je postavena mimo jiné na filozofii její jednorozměrné varianty. Hlavní částí 2D posuvníku je vizualizace kombinací obou složek její hodnoty, s níž je zamýšlena většinová část interakce. Pro zjednodušení přesného výběru je její nedílnou součástí také řada podpůrných prvků, přičemž nejpodstatnějším z nich je přítomnost os pro obě složky výstupu. Tyto osy právě používají dříve zmiňovaný koncept posuvníku, jsou ovšem uzpůsobeny použití v kombinaci s vizualizací výsledných hodnot.

Vzhled a geometrie

Celá komponenta se skládá ze tří klíčových částí. První, dříve již zmiňovanou částí, je vizualizace kombinací složek vybírané hodnoty. Tato vizualizace je pohyblivá a její posunutí v každé ose souřadnic je určeno aktuálně vybranou hodnotou příslušících složek. Tyto hodnoty je možné měnit za pomocí gest aplikovaných přímo na plochu vizualizace i mimo ni. Podoba této části je volena dle kontextu použití 2D posuvníku. Její obsah volí sám uživatel balíčku, přesto má omezení týkající se jejích rozměrů. Výchozí rozměr (při použití měřítka 1:1) je roven rozměru 2D posuvníku zmenšenému o velikost os ve výchozím (neaktivním) stavu.

Tyto osy znázorňují vybranou hodnotu jedné ze složek v kontextu jejího rozsahu. Osa využívá konceptu použitého v jednorozměrné variantě posuvníku, přesto zde bylo potřeba učinit několik specifických úprav. Pro zajištění co nejmenšího překryvu vizualizace, a tím zpřehlednění nejpodstatnější části 2D posuvníku, byl poměr velikosti osy k celé této komponentě zredukován. Tato redukce vedla mimo jiné k odstranění textového popisu hodnot všech jednotek. Pro zachování přehlednosti je za pomocí gest možné osu dočasně aktivovat, čímž se zvětší její plocha a zobrazí se jednotlivé popisky hodnot jednotek, jako je tomu u 1D posuvníku. Podoba neaktivní osy je kromě absence popisků rozložením totožná s návrhem osy jednorozměrného posuvníku. Jak je možné vidět také na obrázku 3.3, podoba aktivní osy ovšem oproti předchozí variantě rozdílná je, a to pozicí textových popisků i symbolických čárek reprezentujících pozici jednotky. Tyto čárky jsou v případě aktivní osy zarovnány k jejímu vnějšímu okraji a jsou odsazeny o přibližně 10 % výšky osy. Tato změna rozložení je provedena za účelem lepší efektivity rozdělení prostoru pro vložení textových popisků. Takto navržené osy mají výšku rovnu 1/10 kratšího rozměru celého 2D posuvníku, přičemž jejich umístění je na jeho kraji dle orientace dané složky. (Osa pro vertikální výběr je umístěna na pravém okraji komponenty a osa pro horizontální výběr je umístěna na spodním okraji.)

Další nepostradatelnou součástí této varianty je středový ukazatel aktuálně vybrané hodnoty. Tento ukazatel je umístěn na středu plochy vizualizace, je ovšem nepohyblivý. Při volbě hodnot tedy zůstává na této fixní pozici, zatímco je prováděna interakce s vizualizací hodnoty. Tento ukazatel má podobu znaku plus, kde každá jeho strana je zarovnána s příslušným středovým ukazatelem jednotlivých os. Výchozí barva tohoto ukazatele je kontrastní barvou k barvě jeho podkladu, tzn. je určena dle příslušící oblasti vizualizace dynamicky.



Obrázek 3.3: Ukázka návrhu vlastního 2D posuvníku s aktivní horizontální osou a neaktivní vertikální osou.

Interakce pomocí gest

Tato varianta posuvníku podporuje obdobná gesta pro změnu vybrané hodnoty a měřítka, jak je popsáno u jednorozměrného provedení. Jediným podstatným rozdílem z pohledu interakce je zde aktivace samotných os. Efekt použitých gest na stav komponenty je v tomto případě rozdělen dle místa aplikace. V případě využití gest potažení či roztažení dvou prstů na ploše vizualizace se aplikuje požadovaná akce na obou osách zároveň. V tomto případě je ovšem pro každou osu vybrána odpovídající složka gesta. Tzn. v případě potažení prstem na ploše vizualizace se pro vertikální osu použije složka gesta odpovídající posunu na ose y, kdežto pro horizontální osu se použije složka odpovídající délce posunu na ose x. Stejně tomu tak je i pro gesto roztažení dvou prstů. V případě aplikace gest přímo na osách 2D posuvníku se změna aplikuje pouze pro danou osu, přičemž se použije pouze příslušící složku délky gesta. Zároveň s touto interakcí je osa také animovaně aktivována. V tomto stavu setrvává po dobu dvou sekund, přičemž pokud uživatel provede opět kterékoli z gest, tento časovač je resetován a osa zůstává aktivní. Reakční plocha obou os posuvníku je záměrně zvětšena i mimo jejich viditelnou hranici, a to za účelem snazší interakce s vizuálně již tak zúženou osou. Tato plocha má rozměry aktivované osy a v případě její aktivace se tyto rozměry nadále nemění.

Kapitola 4

Rozhraní pro práci s komponentami

Oba posuvníky jsou dostupné v samostatném balíčku s názvem *SwiftUI-Sliders*, typu *Swift Package*. Zde jsou tato provedení posuvníků dostupné jako samostatné celky s názvy `PreciseSlider` a `PreciseSlider2D`. Toto provedení bylo zvoleno jednak pro jejich rozdílnost, ale také pro zjednodušení práce s celým balíčkem *SwiftUI-Sliders*. Balíčkovací systém *Swift Packages* je oficiálně podporovaným nástrojem společnosti Apple pro distribuci znovupoužitelných samostatných součástí s podporou také ve výchozím vývojovém prostředí pro tvorbu nejen iOS aplikací, XCode [14]. Obě varianty jsou implementovány pomocí technologie SwiftUI, tudíž je jejich rozhraní primárně koncipováno pro toto provedení. Rozhraní pro framework UIKit je tedy pouze nástavbou a tvoří vrstvu abstrahující původní logiku posuvníků. Hlavním rozdílem, který zároveň značně komplikuje provedení této vrstvy, je využití rozdílných návrhových vzorů k nakládání s jednotlivými technologiemi. Zatímco starší technologie UIKit využívá návrhového vzoru *MVC*, framework SwiftUI využívá naopak návrhového vzoru *MVVM*. Přestože tato novější technologie obsahuje prostředky pro snazší přechod ze svého předchůdce, jejich rozdíly jsou natolik významné, že bylo potřebné rozhraní posuvníků pro použití s využitím frameworku UIKit v jistých ohledech omezit. A to proto, aby bylo možné naprosto odstínit veškeré prvky její implementace a zajistit tím přirozenější formu práce s posuvníky za užití základních konceptů UIKit.

4.1 Výchozí rozhraní pro použití se SwiftUI

Jak již bylo dříve popsáno, technologie SwiftUI využívá návrhového vzoru *MVVM*. S tímto ohledem zaštiťuje část *ViewModel* hlavní logiku a zajištění stavu obou komponent balíčku *SwiftUI-Sliders*. Tento stav zahrnuje například aktuálně vybranou hodnotu, aktuální měřítko, konfiguraci logiky komponenty, jako např. maximální či minimální hodnotu, výchozí měřítko, aj. Část *View* je u jednotlivých variant odpovědná za veškerou vizualizaci a interakci s tímto stavem, případně předávání dalších částí zobrazení, jako např. popisky vyznačených jednotek. Další podstatnou částí, tentokrát mimo logiku návrhového vzoru *MVVM*, je stylovaní posuvníků. Pro tento účel v projektu slouží stylovací třídy s příponou `-Style` (`PreciseSliderStyle` a `PreciseSlider2DStyle`). Tyto třídy obsahují atributy, specifické pro oba posuvníky, sloužící k úpravě některých vizuálních vlastností. Tyto třídy stylů se následně předávají jako speciální proměnné prostředí.

ViewModel

Jak již bylo zmíněno, ViewModel v této implementaci obsahuje stav a výchozí hodnoty posuvníků. Většina implementace je pro tuto část společná jak pro `PreciseSlider`, tak i pro `PreciseSlider2D`. V případě 2D posuvníku je ViewModel obohacen o princip aktivace osy a je použit pro každou osu výběru jako samostatný objekt (viz kapitola 4.1). Konstruktor se tedy v těchto případech ničím neliší a má následovnou podobu:

```
init(
    defaultValue: Double = .zero,
    defaultScale: Double = 1.0,
    minValue: Double = 0,
    maxValue: Double = 100,
    minScale: Double = 1.0,
    maxScale: Double = .infinity,
    numberOfUnits: Int = 20,
    isInfinite: Bool = false
)
```

Parametry konstruktoru představují: výchozí hodnotu, výchozí měřítko, minimální a maximální hodnotu, minimální a maximální měřítko, počet jednotek osy a její volitelný nekonečný rozsah. Tyto atributy je možné měnit i za běhu, výchozí hodnota a měřítko jsou ovšem vlastnosti aplikovatelné primárně při konstrukci ViewModelu. Položka specifikující počet jednotek slouží ke konfiguraci rozložení komponenty. Udává počet jednotek, na který bude rozsah osy rozdělen při použití měřítka 1:1. Pokud tedy zvolíme počet jednotek rovný 20 na rozmezí hodnoty 0-200, bude jedna jednotka odpovídat hodnotě 10. Volitelný nekonečný rozsah osy je dodatečná implementace, umožňující výběr periodicky se opakujících hodnot. Tato vlastnost je dostupná v třídě `PreciseSliderViewModel` i `PreciseSlider2DViewModel`.

Mezi výstupní hodnoty poté patří aktuální měřítko, označené proměnnou `scale` a aktuálně vybranou hodnotu, stavovou proměnnou, označující, zda-li uživatel aplikace právě provádí výběr hodnoty. Aktuálně vybraná hodnota je dostupná ve dvou variantách pojmenovaných `value` a `unsafeValue`. Jak z názvu proměnné již vypovídá, hodnota `unsafeValue` je primárně určena pro použití vizualizací komponenty, její hodnota totiž může přesáhnout stanovené meze. Pro zajištění možnosti vizuálního přesahu těchto hranic byla zavedena tato proměnná, ze které následně vychází hodnota pojmenovaná `value`. Tato hodnota je ekvivalent dříve zmíněné `unsafeValue`, ovšem její přesah je ošetřen, a tudíž je vhodná pro použití z pohledu uživatele balíčku *SwiftUI-Sliders*. Tyto proměnné jsou v implementační části ViewModel anotovány klíčovým slovem `@Published`, což zapříčiní obohacení tohoto atributu o prostředky pro zasílání asynchronních notifikací o změnách hodnoty ostatním částem programu. (Viz implementace knihovny *Combine*[1].) Hodnota `value` je pro zajištění konzistence dat implementována jako tzv. *computed property*, což znemožňuje použití automatického generování těchto prostředků. Proto implementace obsahuje explicitně vytvořený objekt `valuePublisher`, typu `PassthroughSubject<Double, Never>`, sloužící právě ke kompenzaci tohoto nedostatku.

Mimo zmíněné proměnné obsahují třídy `-ViewModel` také metody pro korektní manipulaci se stavem posuvníků `PreciseSlider` a `PreciseSlider2D`. Tyto metody jsou `move(byValue)`, `zoom(byValue)`, `move(toValue)` a `zoom(toValue)`. První dvě metody

[1] Dokumentace frameworku Combine https://developer.apple.com/documentation/combine

slouží primárně pro změnu stavu z pohledu `View`, jejich funkce spočívá v posunu vybrané hodnoty, či použitého měřítka o délku příslušného gesta. Jelikož jsou tyto metody volány při změně délky gesta, používají na rozdíl od své druhé varianty privátní proměnnou pro ukládání stavu před započetím gesta (proměnné `prevValue` a `prevScale`). Druhá varianta (s parametrem `toValue`) slouží pro změnu stavu ze strany uživatele balíčku *SwiftUI-Sliders*. Tato metoda provádí okamžitou změnu stavu z původně zvolené hodnoty na hodnotu novou.

View

Tato část obsahuje téměř veškerou logiku zobrazení stavu ViewModelu. Jejím rozhraním je pouze konstruktor, který se liší dle varianty posuvníku. Samotný konstruktor v obou případech slouží pro předávání stavového objektu a generické předání volitelného obsahu. Konstruktor struktury `PreciseSliderView` obsahuje pouze jednu položku volitelného obsahu, a tou je popis hodnoty vyznačené jednotky. Primárně je zde očekáván textový popis v podobě instance struktury `Text`, parametrem konstruktoru ovšem může být jakákoli struktura implementující protokol `View`.

Obdobně tomu tak je u struktury `PreciseSlider2DView`, jejíž konstruktor vypadá následovně:

```
PreciseSlider2DView(
    axisX: PreciseAxis2DViewModel,
    axisY: PreciseAxis2DViewModel,
    content: (_ size: CGSize, _ scale: CGSize) -> Content,
    axisXLabel: (_ value: Double, _ step: Double) -> AxisXLabel,
    axisYLabel: (_ value: Double, _ step: Double) -> AxisYLabel
)
```

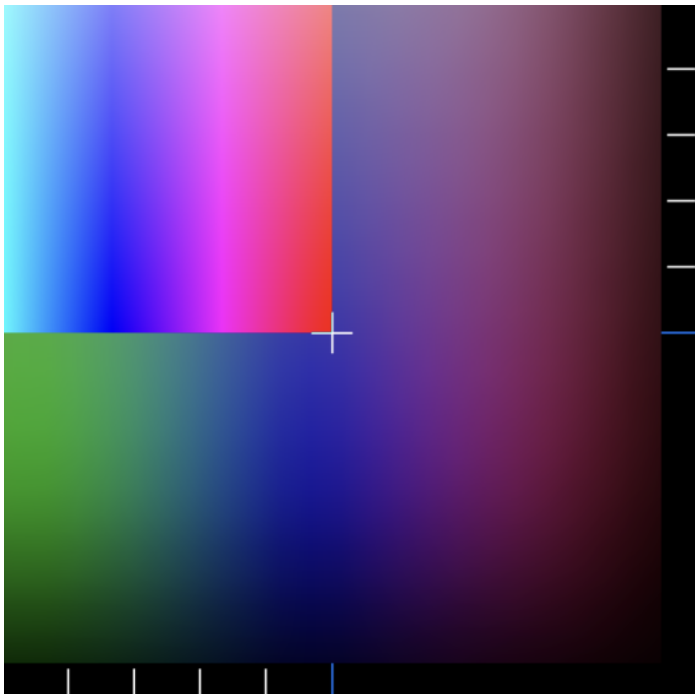
U této struktury se ovšem předávají dvě stavové proměnné, a to příslušné třídy `PreciseAxis2DViewModel` pro vertikální osu a pro horizontální osu. Dalším parametrem je u této varianty proměnná určující obsah zobrazené vizualizace (`content`). Posledními parametry poté jsou položky pro specifikaci popisu vyznačených jednotek, tentokrát pro každou osu opět zvlášť. Tato vizualizace je předávána jako funkční blok nebo také `closure`, jehož parametry jsou výchozí rozměr vizualizace (pro měřítko 1:1), a aktuálně použité měřítko. Parametr `scale` slouží uživateli komponenty pro případnou úpravu rozlišení vizualizace. Změna velikosti vizualizace dle zvoleného měřítka je již prováděna v implementaci této struktury. Pro volbu popisu vyznačených jednotek je rovněž použit funkční blok, v tomto případě s parametrem aktuálního stavu vybrané hodnoty a parametrem délky kroku pro případnou úpravu přesnosti textového zobrazení hodnoty. Návratové typy těchto funkčních bloků odpovídají asociovaným typům dle definice struktury:

```
struct PreciseSlider2DView<
    Content: View,
    AxisXLabel: View,
    AxisYLabel: View
>: View
```

Styly

Specifikace vizuálních stylů posuvníků se provádí předáním stylovacího objektu jako tzv. `Enviroment Value`. Jedná se o jeden z využívaných způsobů specifikace stylů s použitím technologie SwiftUI, a to díky faktu, že toto provedení umožňuje sdílet instanci této třídy v celé hierarchii daného zobrazení [11]. Atributy těchto tříd poté znázorňují vizuální vlastnosti jednotlivých komponent. Třída, příslušící 1D posuvníku, `PreciseSliderStyle`, konkrétně poskytuje možnost změny použitého pozadí osy, změnu barvy ukazatele a barvy jednotlivých jednotek. Barvu je pro každou jednotku možno volit zvlášť za pomocí funkčního bloku. Parametry tohoto bloku jsou hodnota, příslušící dané jednotce, a logická hodnota, označující, zda-li je daná jednotka vyznačená (obsahuje textový popis a je vizuálně větší).

Dvourozměrná varianta posuvníku poskytuje rovněž možnost změny barvy pozadí os a jejich ukazatelů, barvu je ovšem možné měnit pouze u obou os zároveň. Varianta mimo to poskytuje možnost volby pozadí vizualizace. Pro volbu pozadí vizualizace jsou dostupné dvě možnosti: statická barva pozadí a efekt rozostřeného obrazu vizualizace. Tento obraz je pohyblivý, má velikost 1,3násobku velikosti vizualizace kombinací složek a jeho posunutí je určeno poloviční hodnotou posunutí vizualizace. Pozadí tedy dodává efekt hloubky, přestože posuvník využívá konceptu plochého designu (viz. obrázek 4.1). Další volitelnou vlastností je barva středového ukazatele. Stejně jako u pozadí, možnost barvy ukazatele poskytuje rovněž dvě dostupné možnosti, a to statickou barvu a dynamickou volbu barvy. Dynamická volba barvy zapříčiní, že barva ukazatele bude vždy odpovídat invertované barvě jejího pozadí. Tato volba je vhodná pro obsah se širším spektrem barev jako například zobrazení barevných schémat. Spolu s barvou středového ukazatele je pomocí stylů možné změnit také jeho velikost. Výchozí velikost je stanovena na konstantu 10x10 jednotek. Poslední položkou třídy pro stylování komponenty `PreciseSlider2D` je volba barvy jednotlivých jednotek os dle konceptu zmiňovaného u předchozí varianty.



Obrázek 4.1: Volba dynamické barvy středového ukazatele a pozadí s efektem rozmazání vizualizace.

4.2 Rozhraní pro použití s technologií UIKit

Rozhraní pro použití s technologií UIKit je rozděleno na dva hlavní díly dle základních konceptů této technologie. Jedná se konkrétně o část *ViewController* a *View*. Rozhraní je realizováno s využitím pomocného prostředku pro integraci kódu, využívajícího novou technologii SwiftUI, do aplikací postavených na frameworku UIKit. Tímto rozhraním je konkrétně třída `UIHostingViewController`, jež zaobaluje třídy `PreciseSliderView` a `PreciseSlider2DView` vytvořené za pomocí SwiftUI a poskytuje zvyklé rozhraní pro manipulaci v aplikacích využívajících jeho starší variantu. Vlastní implementace rozhraní ovšem za účelem přívětivější manipulace s komponentou zabaluje prostředí `UIHostingViewController` do vytvořené implementace částí View a ViewController. Toto vlastní provedení následně nabízí prostředí pro modifikaci parametrů a vlastností komponenty způsobem zvyklým pro framework UIKit.

(UI)View

Tato část zastřešuje konstrukci objektu `UIHostingViewController`, přesto je zde využit pouze jeho atribut `view`. Třída `UIHostingViewController` v tomto případě představuje vrstvu pro použití komponent, vytvořených za pomocí technologie SwiftUI v prostředí frameworku UIKit.[2] Část View (implementace v souborech `UIPreciseSlider.swift` a `UIPreciseSlider2D.swift`), poskytuje uživateli balíčku možnost stylování komponenty. Všechny atributy stylovacích tříd jsou tímto rozhraním zaobaleny, přesto implementace interně pracuje s instancemi `PreciseSliderStyle` a `PreciseSlider2DStyle`. Za účelem zpřístupnění atributů stylovací třídy v této vrstvě byla přesunuta finální konstrukce pomocného prostředí a grafického rozložení komponenty právě zde. Třída této části implementace dědí z třídy `UIView`, je tedy použitelná dle zvyklých principů jako plnohodnotná implementace tohoto datového typu.

ViewController

Jak je již z názvu možné poznat, třídy `UIPreciseSliderViewController` a `UIPreciseSlider2DViewController` zastávají roli správy dat (v tomto případě dat stavu, viz. kapitola 4.1) a notifikací o jejich změně. Mimo jiné tyto třídy slouží pro korektní konstrukci objektů tříd zmíněných v předchozí kapitole. Primární zodpovědnost této části je tedy nakládání se zobrazením typu `UIView` a konfigurace komponenty na její datové úrovni, poskytnutím atributů `dataSource` a `delegate`. Atributy zde slouží pro konfiguraci výchozích hodnot, mezí a další logiky posuvníku (atribut `dataSource`), a pro přijímání zpráv o změně stavu dat posuvníku (atribut `delegate`). Tento způsob předávání dat a volby konfigurace je možné nalézt v řadě komponent technologie UIKit.

DataSource

Protokoly `PreciseSliderDataSource` a `PreciseAxis2DDataSource` zde slouží k zaobalení téměř veškeré implementace části ViewModel, výchozího rozhraní, pro využití s frameworkem SwiftUI. Tato implementace poskytuje volbu všech parametrů konstruktoru ViewModelu spolu s parametry pro volbu podoby textového popisu jednotek os. Z důvodu

[2] Viz. dokumentace vývojářů https://developer.apple.com/documentation/swiftui/uihostingcontroller

rozdílnosti těchto dvou frameworků bylo ovšem zapotřebí vytvořit kompromis mezi složitostí implementace tohoto rozhraní a poskytnutí plné volnosti z ohledu volby podoby této části. Jako kompromis byla zvolena fixní implementace textovým popisem s možností volby písma, textu a barvy popisků, čemuž odpovídají metody `unitLabelText`, `unitLabelColor` a `unitLabelFont`. Parametry těchto metod jsou shodné s parametry funkčních bloků, předávaných při konstrukci komponenty za pomocí SwiftUI rozhraní. Typy návratových hodnot těchto bloků jsou pro přirozenější práci s tímto rozhraním voleny v jejich variantě, odpovídající výchozím typům frameworku UIKit. Převod těchto hodnot do příslušných typů frameworku SwiftUI následně provádí implementace v souborech s příponou `-ViewController`. Obdobný kompromis bylo nutné učinit mimo jiné i pro obsah vizualizace 2D posuvníku. Ta se předává pomocí separátní třídy `PreciseSlider2DDataSource`. Jako požadovaný datový typ je v tomto případě typ `UIImage`, jenž reprezentuje zobrazení obrázku. Díky nástrojům pro převod grafických komponent uživatelského rozhraní do tohoto typu, které jsou dostupné ve frameworku UIKit, je možné dosáhnout téměř totožné funkcionality, jako za využití výchozího rozhraní. (Viz použití 2D posuvníku v souboru `ColorPicker` ukázkového projektu s názvem `ColorPickerExample-UIKit`.)

Delegate

Část *delegate* tvoří rozhraní, jenž v novější technologii nahradil framework `Combine`. Toto rozhraní totiž slouží k předávání zpráv o změně stavu komponenty, převážně vyvolaných interakcemi uživatele grafického rozhraní. Stejně jako je tomu u položky *DataSource*, soubory os s příponou `-Delegate` sdílejí stejnou implementaci, jak u 2D posuvníku, tak i u jednorozměrného provedení. Tato implementace nabízí možnosti notifikace o změně hodnoty, měřítka a notifikace o započetí či ukončení interakce s vybíranou hodnotou (neplatí pro změnu měřítka). Poslední dvě zmiňované informace mohou být užitečné například pro řízení přehrávání videa, kdy je potřeba přehrávání videa po dobu interakce pozastavit. (Použití v implementaci ukázkového projektu `VideoPlayerExample-UIKit`.)

Kapitola 5

Významné pasáže implementace

V následující kapitole budou probrány detaily a záměry způsobu implementace posuvníků `PreciseSlider` a `PreciseSlider2D`. Přestože se jedná o zdánlivě jednoduché provedení posuvníků, pro jejich efektivitu a přívětivost bylo potřeba v implementaci provést řadu speciálních opatření.

5.1 Znovu používání jednotek

Za účelem optimalizace zobrazení os obou variant posuvníku byl vytvořen speciální mechanismus pro nakládání s jejími jednotkami. Jelikož jsou komponenty určeny pro co nejuniverzálnější použití, počet jednotek os není předem známý a může teoreticky nabývat hodnot, kdy jejich ukládání do paměti může být téměř nereálný úkol. Stejný problém nastává rovněž při manipulaci s měřítkem os, pro zachování přehlednosti komponenty je totiž potřeba vizuálně generovat nové jednotky, jejichž počet může obdobně nabývat neudržitelných čísel.

Řešením tohoto problému bylo zavedení systému, který zajistí, že počet jednotek, držených v paměti zařízení, je limitován pouze na co nejmenší. Idea tohoto principu přístupu k nakládání s jednotkami stojí na způsobu pozicování každé z jednotek osy. Po celou dobu zobrazení a interakce uživatele s osou je totiž zobrazen pouze limitovaný počet stále stejných jednotek, jejichž pozice je omezena na rozmezí vzdálenosti šířky osy. V případě překročení této vzdálenosti jednotkou se její posunutí sníží/zvýší o vzdálenost šířky osy, jejíž je součástí. Tento mechanismus provádí metoda `unitOffset`, přítomná v souborech `PreciseAxisView` a `PreciseAxis2DView`, za pomocí operace modulo dle následujícího principu:

```
if offset > max {
    offset = offset % max
}
else if offset < 0 {
    offset = max + (offset % max)
}
```

Proměnná `max` značí maximální posun jednotky a proměnná `offset` její pozici bez ošetření přesahu mimo zobrazitelnou část. Zobrazení správných hodnot jednotlivých jednotek je následně přepočteno z aktuálního stavu vybrané hodnoty (proměnná `unsafeValue`) a identifikátoru dané jednotky.

Obdobný koncept je použit i co se týče změny měřítka. Při změně měřítka je vzdálenost mezi jednotkami rozšiřována do hraniční úrovně, kde je následně opět resetována do výchozího stavu. Rozložení jednotek se tedy při překročení této hraniční hodnoty opět vrátí do

původní podoby. Pro zajištění efektu generování jednotek je poté zaveden princip změny velikosti jednotky v závislosti na aplikovaném měřítku. Ve stavu měřítka 1:1 je výška každé páté jednotky rovna její maximální možné výšce dle rozložení komponenty (viz. kapitoly 3.2 a 3.3), kdežto ostatní jednotky mají výšku rovné nule. S rostoucím měřítkem roste výška těchto jednotek dle vztahu $(scale - 1)/3 * max$, kde $scale$ představuje měřítko a max maximální výšku jednotky (viz. metoda `minUnitHeightRation` v souboru `PreciseAxisView`). Tento princip zajišťuje plynulý přechod působící dojmem, že s rostoucím měřítkem přibývají stále nové jednotky.

5.2 Implementace animací

Nepostradatelnou částí vizuální stránky 1D a 2D posuvníku je také jejich provedení animací. Samotná osa posuvníku `PreciseSlider` či osa pouze jedné složky posuvníku `PreciseSlider2D` podporuje celkem tři druhy animací. Animace jsou provedeny s využitím prostředků poskytovaných jako součást frameworku SwiftUI. Implementace za pomocí těchto prostředků je poměrně přímočará a snadná, přesto pro využití jejího plného potenciálu bylo zapotřebí zanést úpravy do implementace těchto komponent.

Protokol Animatable

Pro správné provedení animace je potřeba specifikovat, která hodnota je jejím cílem. Tato specifikace je provedena implementací protokolu `Animatable` a konkrétně její hodnoty `animatableData`. Jak už z názvu vypovídá, proměnná `animatableData` v tomto protokolu slouží pro výběr cíle animace, kterým je v případě os vybraná hodnota `unsafeValue` a v případě vizualizace kombinací složek dvojice obou vybraných hodnot pro příslušící osy [3]. (SwiftUI podporuje také animaci vícerozměrných hodnot s využitím typu `AnimatablePair`[1].) Jelikož byl pro správný způsob animování osy použit tento protokol, byla podstatná část logiky osy přesunuta do samostatného souboru. Tato část pouze zobrazuje stav dané osy předávaný pomocí konstruktoru v podobě jednotlivých parametrů. Jedná se o struktury `PreciseAxisView` a `PreciseAxis2DView`. Interakci s osou za pomocí gest a zpětnou vazbu na svůj stavový objekt ViewModel poté zaštiťuje nadřazená vrstva v hierarchii zobrazení komponenty.

Díky této jednoduché specifikaci položky k animování je použití animací poměrně přímočaré. Provedení jakékoli změny hodnoty s použitím animovaného přechodu je možné za pomocí zaobalení této části kódu do bloku `withAnimation`. Funkce `withAnimation` očekává dva parametry, konkrétně parametr specifikace typu animace (jaký vizuální efekt či funkce animace se má použít) a blok kódu, který provádí změnu, jež je zapotřebí zanimovat.[2]

Použité animace

Co se týče použitých typů animací, jak již bylo dříve zmíněno, osy posuvníků podporují tři různé druhy. Prvním typem je animace typu uvolnění, použitá pro animování pohybu osy v rámci jejích stanovených mezí. Druhým typem je animace pohybu osy z pozice, patřící do jejího rozmezí, do bodu mimo její rozmezí. A posledním typem je animaci z pozice mimo interval osy zpět k nejbližší hraniční hodnotě. První z animací, a zároveň nejjednodušší,

[1] Viz dokumentace typu `AnimatablePair` https://developer.apple.com/documentation/swiftui/animatablepair

[2] Funkce `withAnimation` https://developer.apple.com/documentation/swiftui/withanimation(_:_:)

je animace uvolnění, používající typ `easeInOut`. Animace směrem mimo hranici osy je implementována použitím typu `spring`, jenž zavádí do průběhu animace efekt natahování pružiny. Tento druh animace byl použit s motivací dočasného efektu přesažení hranic osy, a tím zajištění interaktivního zvýraznění přítomnosti hraniční hodnoty. Obdobný efekt byl využit při přesahu hranic komponenty použitím gesta, v tomto případě je animován pouze progres ukazatele zpět na hraniční hodnotu. Jako parametr konstruktoru této animace je pouze konstanta 0,75 pro její položku `dampingFraction`. Tento parametr určuje intenzitu útlumu zpětného pohybu animace, způsobeného efektem pružiny. Volba hodnoty 0 tohoto parametru má za následek stálou oscilaci animace. Zvolená konstanta 0,75 představuje kompromis vysoké úrovně útlumu oscilace, a přesto zachování možného přesahu hranice osy. Takto vysoká hodnota útlumu rovněž zapříčiní téměř nulový přesah této hranice při zpětném pohybu směrem k její opačné hraniční hodnotě. Předposlední animací je pohyb osy zpět na hraniční hodnotu při jejím přesahu za použití gesta. Tato animace je rovněž typu `spring`, tentokrát s parametrem `dampingFraction` rovným kritické hodnotě 1, jenž představuje nejrychlejší volbu této animace pro návrat bez použití efektu oscilace. Popis chování animace typu `spring` s parametrem `dampingFraction` byl přejat z [7]. V poslední řadě je v implementaci přítomna animace aktivace a deaktivace os komponenty pro výběr dvourozměrných hodnot, která používá variantu `easeInOut`.

5.3 Přesah hranic osy

Pro přirozenější podobu interakce s komponentou byl zaveden efekt měkké zarážky hraničních hodnot posuvníku. Měkkou zarážkou je v tomto kontextu myšlena možnost dočasného přesahu maximální či minimální hodnoty osy její vybranou hodnotou. Toto chování je možné pozorovat mimo jiné v řadě systémových aplikací a v samotném prostředí operačního systému iOS (např. obnovení obsahu stránky v aplikaci Safari). Převážnou část tohoto efektu tvoří dříve popsaná sada animací, přesto se nejedná o jediné nutné opatření k dosažení úplnosti tohoto efektu.

Nepostradatelnou součástí je rovněž možnost přesahu těchto hranic za pomocí aktivně prováděného gesta, tudíž umožnění uživateli „přetáhnout" osu mimo dosah ukazatele aktuálně zvolené hodnoty, jak je ukázáno na obrázku 5.1. Tohoto efektu bylo docíleno implementací, která skutečně dovoluje uživateli vybírat hodnotu i mimo tento rozsah. Za tímto účelem slouží právě rozdělení stavové proměnné aktuálně vybrané hodnoty na proměnnou `unsafeValue` a `value`. Skutečný stav udává proměnná `unsafeValue`, která již dle názvu napovídá, že nabývá hodnot právě i mimo zvolený interval. Hodnota `value` poté znázorňuje požadovanou hodnotu, jež odpovídá `unsafeValue`, ošetřené právě o tento možný přesah.

Obrázek 5.1: Ukázka překročení mezní hodnoty osy ukazatelem. Při dokončení gesta se spustí animace pro přechodu k nejbližší mezní hodnotě (hodnota 100).

Tento efekt informuje uživatele o významu právě této hraniční hodnoty, přesto by interakce nebyla dostatečně jednoznačná, nebýt další vlastnosti tohoto efektu, a to zavedení efektu působení síly proti směru tahu směřujícímu mimo hranici osy. Implementace tohoto jevu je provedena aplikací odmocninné funkce na tento přírůstek. V samotné implementaci

metody `move(byValue)` je případný přesah detekován a výsledná délka přepočítána dle reálné délky přesahu za pomocí vzorce $\sqrt{|x| + \frac{1}{4}} - \frac{1}{2}$, kde x je reálná délka přesahu. Nová hodnota přesahu je ve výsledku přičtena, či odečtena k příslušící hraniční hodnotě.

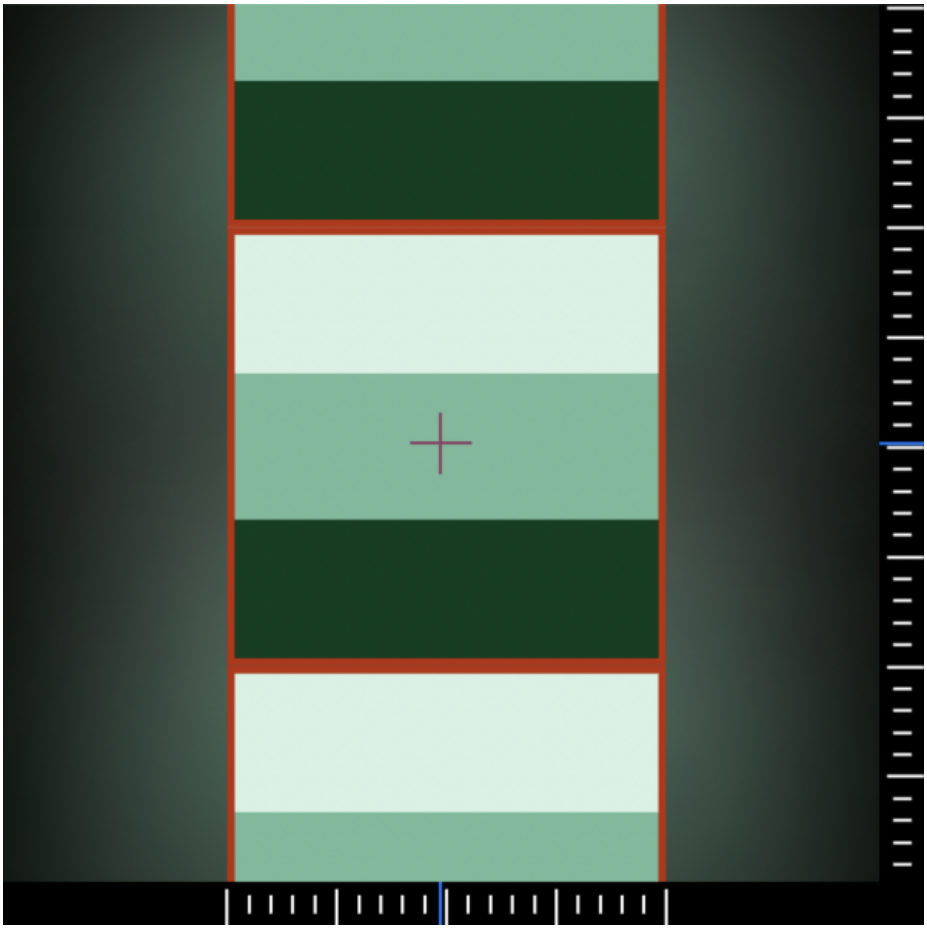
5.4 Pseudo-nekonečný rozsah osy

Dodatečnou částí implementace posuvníků je podpora pro výběr hodnot z oboru periodického charakteru. Jedná se například o volby rotačního úhlu, složky některých barevných modelů nebo času. Očekávaným chováním osy je v tomto případě kruhové propojení hraniční minimální hodnoty s její maximální, a tím umožnění neomezeného rozmezí posunu osy (viz. obrázek 5.2). V této konfiguraci komponenty tedy i přes její nekonečný rozsah hraje minimální a maximální hodnota stále podstatnou roli.

Obrázek 5.2: Kruhové propojení maximální hranice osy s minimální.

Z implementační stránky tento koncept podpořil způsob nakládání s jednotlivými jednotkami (viz kapitola 5.1), přesto se nejedná o jediné opatření. Pro zajištění pseudo-nekonečného rozsahu osy bylo zapotřebí ošetření přetečení identifikátorů jednotlivých jednotek. Řešením tohoto problému bylo zavedení nového chování osy při překročení jejího limitu. Při překročení hranic osy za pomocí gesta je zvolená hodnota posunuta o celé rozpětí posuvníku, což zapříčiní návrat k její protější hraniční hodnotě. Takto provedená změna je realizovaná operací modulo. Změna se provádí okamžitě a spolu se zajištěním odpovídajících označení jednotek je ze strany uživatele grafického rozhraní nerozpoznatelná.

Co se týče komponenty `PreciseSlider2D`, implementace nebyla tak přímočará jako u její jednorozměrné varianty. Přestože chování os v tomto případě sdílí příslušící část implementace, hlavním problémem byla naopak přítomná vizualizace. Jak je ukázáno na obrázku 5.3, efektu nekonečnosti rozsahu bylo v tomto případě dosaženo vícenásobným zobrazením totožného obsahu těsně vedle sebe. Pro zajištění optimálního provozu posuvníku a zároveň zachování tohoto efektu je v případě volby pseudo-nekonečného rozsahu osy složky vizualizace zobrazena pouze s co nejmenším počtem opakování. Počet opakování je odvozen od použitého měřítka vztahem $x = \frac{1}{scale} + 2$, kde x je počet jednotek a *scale* představuje použité měřítko. Výsledek tohoto vzorce je následně zaokrouhlen nahoru a upraven tak, aby jeho hodnotou bylo liché číslo. Toto provedení zajišťuje viditelnost obsahu i mimo obě stanovené hraniční hodnoty. Zobrazení chybějícího obsahu mimo plochu stanoveného počtu vizualizací je následně ošetřeno obdobným způsobem, jako tomu je u implementace jednotlivých os. Posunutí celého seskupení vizualizovaných dat je ošetřeno modulem rozměru jedné vizualizace, tudíž zde nedojde k odkrytí jejich hraničních hodnot a chybějícího obsahu.



Obrázek 5.3: Ukázka konfigurace pseudo-nekonečného rozsahu vertikální osy 2D posuvníku. (Pro měřítko 2:1.) Červený okraj označuje jednu kopii vizualizace.

5.5 Interakce za pomocí gest

Interakce pomocí gest tvoří jednu z nejpodstatnějších částí obou posuvníků z pohledu uživatele grafického rozhraní. Jejich implementace pomocí technologie SwiftUI je poměrně přímočará, přesto komponenty `PreciseSlider` a `PreciseSlider2D` obsahují řadu specifických implementačních detailů.

Jedním z nich je například způsob změny stavu vybrané hodnoty za pomocí gesta potažení. Technologie SwiftUI umožňuje přidávat reakce na gesta za pomocí tzv. modifikátorů, které umožňují jak modifikaci podoby vizuálních prvků, tak i přidávání dodatečné funkcionality [16]. Takovýmto modifikátorem je modifikátor `gesture`, sloužící k implementaci reakce na dotyková gesta, vztahující se ke konkrétnímu vizuálnímu prvku. Modifikátor `gesture` očekává jako parametr instanci typu `Gesture`. Na instanci konkrétního typu gesta je možno poté pomocí modifikátoru animace aplikovat akci, která bude provedena při změně jeho stavu. Výchozí implementace poskytuje konkrétně tři takovéto modifikátory, jimiž jsou `onChanged`, `onEnded` a `updating`. Akce, předávaná těmto modifikátorům, má podobu funkčního bloku s parametrem pro předávání instance stavového objektu gesta. Při změně stavu gesta se tomuto bloku předává aktualizovaný stavový objekt. V případě gesta potažení

prstem se jedná například o aktualizaci jeho délky a aktuální polohy prstu v souřadném systému displeje zařízení. [2] Pozici posuvníku je ovšem nutno měnit vzhledem k jeho pozici před započetím gesta, k čemuž slouží proměnná s názvem `prevValue`. Zavoláním metody `move`, nacházející se v souboru `PreciseSliderViewModel.swift`, se stav aktuálně vybrané hodnoty nahradí hodnotou odpovídající součtu stavu před započetím gesta a délky gesta. Pro správnou funkci tohoto mechanizmu je ovšem zapotřebí provedení aktualizace stavové proměnné `prevValue` při dokončení interakce.

Modifikátor `onEnded`, jak již z jeho názvu vypovídá, umožňuje přidat vlastní funkční blok, který bude proveden při dokončení gesta. Problém ovšem nastává v případě přerušení interakce, které může nastat např. zobrazením upozornění o stavu baterie. V případě přerušení není funkční blok, předaný prostřednictvím modifikátoru `onEnded`, volán, což může vést k nekonzistenci stavu posuvníku. Z tohoto důvodu byla implementace reakce na gesta provedena za pomocí kombinace modifikátoru `updating` a speciální stavové proměnné s označením `@GestureState`. Změna této stavové proměnné je prováděna pomocí funkčního bloku předávaného za pomocí zmiňovaného modifikátoru. Podstatnou vlastností je zde ovšem automatické nastavení výchozí hodnoty této stavové proměnné v případě přechodu gesta do neaktivního stavu [6]. Za pomocí této vlastnosti je tedy možné detekovat také přerušení gesta a správně provést aktualizaci stavu posuvníku.

5.6 Dynamická rekonfigurace barev ukazatele a pozadí

Za účelem lepší viditelnosti středového ukazatele v kontrastu s pozadím aktuálního stavu vizualizace, a tím zpřehlednění výběru požadované hodnoty pomocí grafické komponenty `PreciseSlider2D`, byla již v návrhu zavedena možnost automatické rekonfigurace barvy ukazatele dle barvy příslušné části vizualizace. Co se realizace této možnosti týče, implementace se od návrhu lehce liší. Barva celého ukazatele při této volbě není jednotná, jedná se totiž o inverzi barev plochy, kterou daná část ukazatele zakrývá. Toto provedení přináší oproti původnímu návrhu zásadní výhodu. Volbou jednotné barvy celého ukazatele by byla limitována jeho rozpoznatelnost od obsahu vizualizace pouze na její jediný bod. V případě obsahu s prudšími přechody barev může tato vlastnost naopak snížit poměr kontrastu ukazatele.

Tento koncept je uskutečněn zavedením vrstvy, pokrývající plochu ukazatele. Tato vrstva odpovídá obsahu vizualizace včetně aplikace měřítka a jejího posunu, ovšem s dodatečným efektem inverze barev. Provedení inverze barev bylo možné za použití modifikátoru `colorInvert`. Tato invertovaná vrstva byla aplikována obdobným způsobem, použitím modifikátoru `overlay`.

Co se volby dynamického výběru pozadí týče, byla mimo možnost statické barvy pozadí v implementaci zavedena také varianta `blurredContent`. Jak již z názvu vypovídá, jedná se o zrcadlení obsahu vizualizace na plochu, náležící mimo její hranice. Tato varianta byla zavedena z čistě estetického důvodu za účelem zvýraznění mezí vizualizace s volbou konečného rozsahu os. Zrcadlení obsahu je ovšem doplněno o řadu vizuálních efektů, jako rozmazání, a tím jeho prolnutí do konstantní barvy pozadí os 2D posuvníku, zatmavení, zvětšení oproti zobrazené vizualizaci a prostorový efekt, dosažený zavedením koeficientu posuvu.

Z implementační stránky je tato část podobná zobrazení vizualizace, rozdílem je použití koeficientů posunu a zvětšení. Mimo tyto konstanty je zobrazení ztmaveno, konkrétně o 30%, a rozmazáno pomocí modifikátoru `blur` s parametrem `radius` o konstantě 50. Výplní plochy, nezakryté tímto zobrazením, je barva pozadí os komponenty dle aplikovaných

stylů. Díky efektu rozmazání zmiňovaného zobrazení kopie vizualizace je přechod do této barvy plynulý. V případě použití možnosti nekonečného rozsahu v obou osách toto řešení postrádá význam, jelikož k odhalení plochy pod vizualizací nedochází. Z tohoto důvodu byla za účelem optimalizace zavedena do implementace podmínka, zajišťující použití této varianty pouze v případě, kdy je tato plocha viditelná.

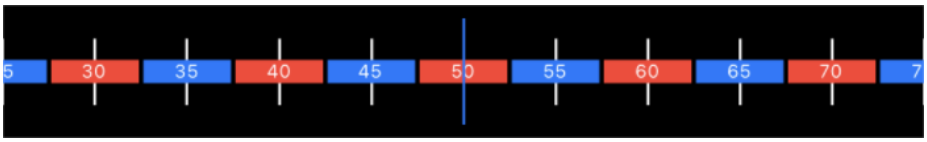
5.7 Generické předávání obsahu

Za účelem flexibilnější volby obsahu vizualizace komponenty `PreciseSlider2D` či volby jednotlivých popisků jednotek os, byl využit přístup generického předávání libovolného zobrazení. Takto předávaný obsah je typu funkčního bloku s návratovým typem `View`, představující jakoukoli položku grafického rozhraní frameworku SwiftUI. Takováto volnost slouží pro lepší přizpůsobení komponenty specifickému použití, přesto tento obsah má jistá omezení, která zajišťují správné rozložení komponenty.

Co se týče implementace posuvníku `PreciseSlider`, jedná se pouze o jedinou položku, a to obsah popisku jednotlivých os. Předávaný obsah má podobu funkčního bloku s návratovou hodnotu jakéhokoli typu, implementující protokol `View`. Samotná motivace za provedením této asociace pomocí generického typu tkví ve způsobu implementace tohoto protokolu. Protokol `View` totiž vyžaduje alespoň implicitní specifikaci typu jedním z výchozích elementů sady grafických prvků [4]. V tomto případě má požadovaný blok následující podobu:

```
valueLabel: (_ value: CGFloat, _ stepSize: CGFloat) -> ValueLabel?)
```

Typ `ValueLabel` je asociací generického typu, stanoveného v záhlaví struktury `PreciseSlider`. Předávání těchto hodnot je popsáno v kapitole 4 o rozhraní jednotlivých posuvníků. Jediným omezením předávaného grafického prvku jsou jeho rozměry. V implementaci grafického rozhraní komponenty `PreciseSlider` je zvolen maximální rozměr tohoto popisku pomocí modifikátoru `frame`, přičemž tento rozměr odpovídá největší možné šířce vzhledem k pozici nejbližší jednotky i s ohledem na popisek vyznačených sousedních jednotek. Demonstraci těchto maximálních rozměrů je možné vidět na obrázku 5.4. Maximální možná výška popisku není stanovena, přesto je omezena maximální výškou celé komponenty. Mimo přesah rozměru posuvníku je toto omezení ovšem možné anulovat použitím modifikátoru `fixedSize`.



Obrázek 5.4: Demonstrace omezení šířky popisků jednotek osy.

Posuvník `PreciseSlider2D` mimo generických parametrů pro volbu popisků jednotek svých os, obsahuje navíc také obdobný parametr pro specifikaci obsahu vizualizace. Tento vizuální prvek má oproti dříve zmiňovaným prvkům omezení rozměru, které není možné anulovat ani s využitím modifikátoru `fixedSize`. Důvodem tohoto striktního omezení je zachování konzistence rozložení posuvníku, a to konkrétně s ohledem na možnou volbu nekonečného rozsahu alespoň jedné z os. Rozložení 2D posuvníku totiž vyžaduje, aby největší přípustný rozměr obsahu ve výchozím měřítku (1:1) odpovídal velikosti celého posuvníku zmenšeného o plochu jeho os.

5.8 Optimalizace pomocí frameworku Metal

Vykreslování jednotlivých jednotek a kopií vizualizace kombinací složek s aplikací různých efektů může v případě jejich většího počtu způsobit přílišnou náročnost obou posuvníků na dostupné prostředky zařízení. Obecně je proto důležité ze strany konfigurace komponenty u hodnot `numberOfUnits` a `minScale` volit kompromis mezi požadovaným chováním a efektivitou běhu aplikace. Za účelem minimalizace dopadu tohoto kompromisu byl v implementaci částí, které můžou tento jev způsobovat, použit modifikátor `drawingGroup`. Tento modifikátor zajistí vykreslování skupiny prvků těchto částí jako samostatné zobrazení položky uživatelského rozhraní v bitmapovém formátu, a tím umožní zpracování za pomocí frameworku *Metal* [8]. Samotný framework *Metal* využívá k vykreslení hardwarovou akceleraci, která může napomoct právě při vykreslování většího počtu takovýchto zobrazení [12].

Kapitola 6

Použití balíčku a komponent

Tato kapitola obsahuje stručnou demonstraci použití již hotového řešení. Toto zahrnuje import balíčku *SwiftUI-Sliders* do existujícího projektu a jednoduchou ukázku aplikace jeho posuvníků za použití rozhraní pro technologie UIKit i SwiftUI. Samotné řešení poté obsahuje sadu demonstračních projektů s ukázkami použití balíčku a jednotlivých posuvníků v obou zmiňovaných frameworcích.

6.1 Použití balíčku ve vlastním projektu

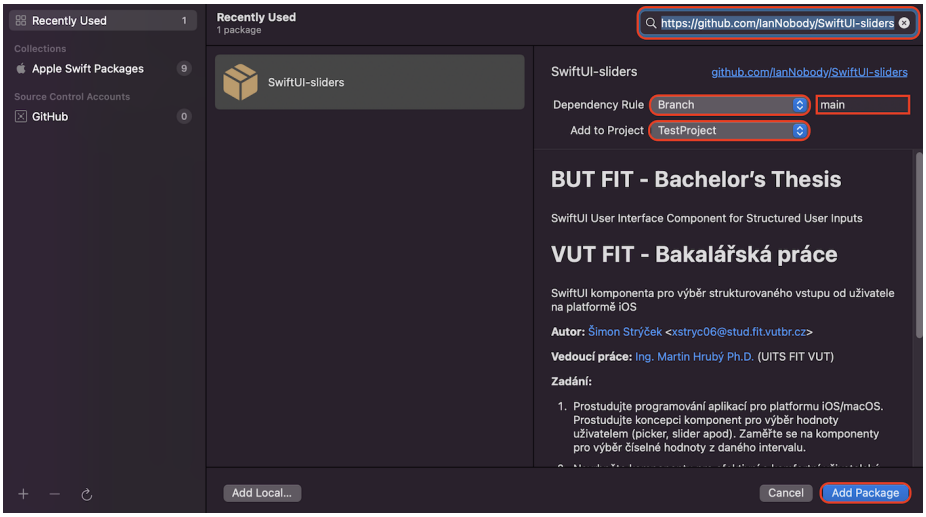
Výsledný produkt práce má podobu balíčku, balíčkovacího systému *Swift Package*, který umožňuje jednoduchou přenositelnost nezávislých součástí implementace. Samotné výchozí prostředí pro tvorbu aplikací na platformu iOS, XCode, obsahuje plnou podporu pro jednoduchou práci s tímto systémem. Vložení balíčku do existujícího projektu je proto jednoduché. Použití balíčku *SwiftUI-Sliders* ve vlastním řešení je možné dvěma různými způsoby. Jedním z těchto způsobů, a zároveň preferovaným způsobem, je vložení tohoto balíčku ze vzdáleného veřejného repozitáře. Je zde také druhá možnost, a to jeho vložení z lokálního zdroje. Vložení balíčku za pomocí vzdáleného repozitáře je preferovanou variantou z důvodu možných nadcházejících aktualizací, ovšem i přes případné aktualizace je možno při vkládání specifikovat požadovanou verzi.

Import balíčku ze vzdáleného repozitáře

Díky integraci podpory balíčkovacího systému přímo v grafickém rozhraní vývojového prostředí XCode, je přidání balíčku ze vzdáleného repozitáře poměrně přímočará věc. K tomuto účelu slouží nabídka, zobrazená na obrázku 6.1, dostupná v sekci *File > Add Packages...* Zde je nejprve potřeba do pole v pravém horním rohu zadat adresu URL veřejného repozitáře balíčku *SwiftUI-Sliders*.[1] Po zadání adresy následně vybrat parametry pro specifikaci jeho verze, konkrétního bodu, či větve repozitáře a cílový projekt pro import balíčku. Po zvolení parametrů pak stačí pouze výběr potvrdit, přičemž se provede stažení ze vzdáleného repozitáře. Následně lze využít tento balíček v kódu. Pro více informací o přidávání balíčků ze vzdáleného repozitáře viz oficiální manuál.[2]

[1] URL veřejného repozitáře https://github.com/IanNobody/SwiftUI-sliders

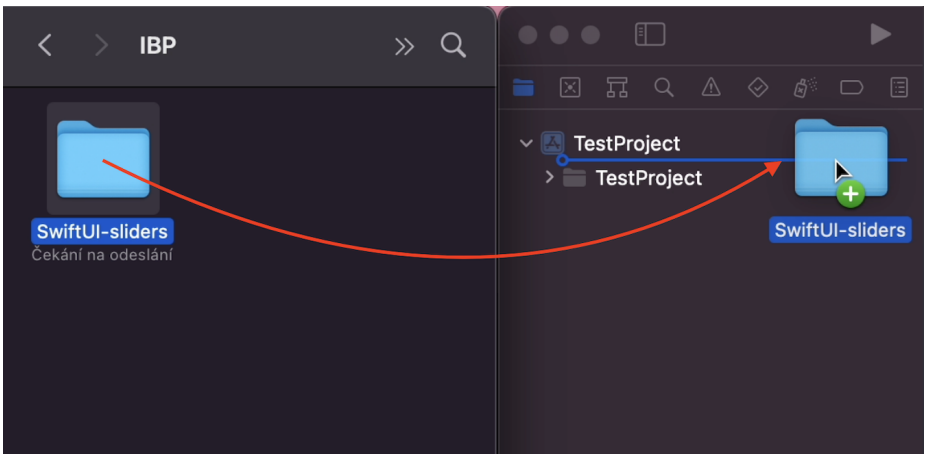
[2] Manuál pro přidávání závislostí do vlastních aplikací https://developer.apple.com/documentation/swift_packages/adding_package_dependencies_to_your_app



Obrázek 6.1: Ukázka importu balíčku ze vzdáleného repozitáře pomocí nástroje správce balíčků. Červeně jsou vyznačeny podstatné části okna.

Import balíčku z lokálního zdroje

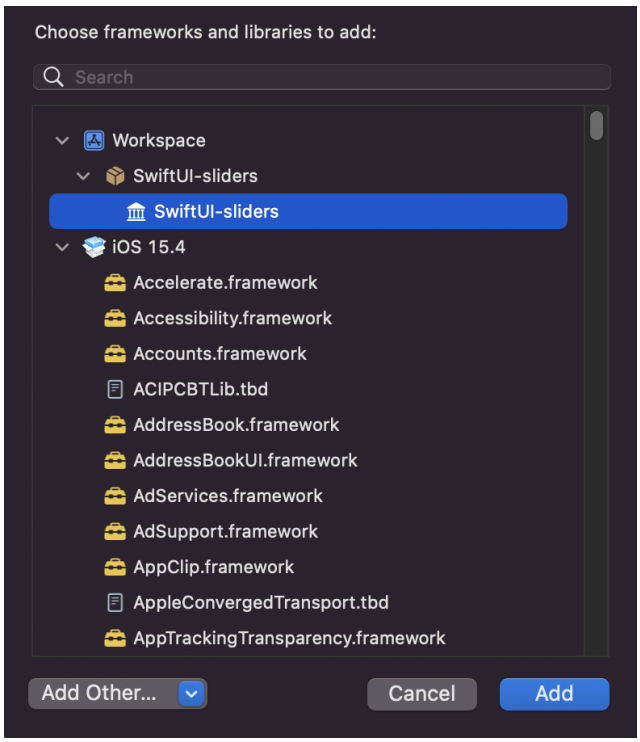
V případě ručního stažení ze vzdáleného repozitáře či jiného zdroje je možné provést import z lokálního úložiště. V tomto případě je provedení možné dvěma různými způsoby. Jedním, zároveň nejjednodušším způsobem, je vložení balíčku přímo potažením kořenového adresáře s názvem *SwiftUI-Sliders* z okna aplikace Finder do podokna *Project Navigator* aplikace XCode (viz. obrázek 6.2).



Obrázek 6.2: Ukázka vložení balíčku *SiwftUI-Sliders* z lokálního zdroje. Část okna vlevo je lokální umístění balíčku v aplikaci Finder a vpravo se jedná o příslušnou část okna vývojového prostředí XCode.

Druhý způsob vložení je opět prováděn za pomocí nástroje správce balíčků, dostupném v horní liště v sekci *File* > *Add Packages...*, tentokrát s použitím možnosti *Add Local...* ve spodní části tohoto okna (viz obrázek 6.1). Tento způsob je obdobně přímočarý, po zobrazení okna pro výběr umístění lokálního zdroje stačí potvrdit zvolené umístění balíčku, který bude následně přidán mezi zdroje projektu.

Co se týče vložení jakéhokoli balíčku z lokálního zdroje, je před jeho použitím v kódu nutno, oproti importu ze vzdáleného repozitáře, podstoupit jeden krok navíc, a to přidání závislosti tohoto balíčku v konfiguraci příslušného projektu. Přidání je možné v konfiguraci daného cíle v záložce *General* v sekci *Frameworks, Libraries, and Embedded Content* (tlačítko +). Po otevření nabídky, znázorněné na obrázku 6.3, poté stačí nalézt dříve vložený balíček, v tomto případě s názvem *SwiftUI-Sliders*, a výběr potvrdit.



Obrázek 6.3: Ukázka okna pro výběr importovaného balíčku jako závislost vlastního projektu.

Použití jednotlivých komponent

Balíček *SwiftUI-Sliders* obsahuje dvě separátní komponenty, a to komponentu `PreciseSlider` a komponentu `PreciseSlider2D`. Každá z těchto variant zároveň obsahuje dvě různá rozhraní pro použití s technologiemi SwiftUI a UIKit. Použití jednotlivých posuvníků importovaného balíčku v kódu je možné až po jeho importu v aktuálním souboru. Takovýto import může vypadat následovně: `import PreciseSlider`. Pro úspěšné použití posuvníků ve vlastní implementaci je nejprve nutné provést správné vložení balíčku do projektu dle předchozích sekcí. Popis jednotlivých rozhraní komponent je možné nalézt v kapitole 4, zaměřené na daná rozhraní. Případné příklady použití je následně možné nalézt v demo projektech, které jsou součástí praktické části práce.

Co se týče posuvníku `PreciseSlider`, jeho použití je možné za pomocí standardního rozhraní s technologií SwiftUI. K tomuto slouží konstruktor struktury `PreciseSliderView`, jenž zaštiťuje vizuální reprezentaci stavu, určeného instancí třídy `PreciseSliderViewModel`. Tento stavový objekt je možno předat zobrazení pomocí konstruktoru spolu s vyžadovaným parametrem pro specifikaci popisků jednotek os. Použití je tedy možné v implementaci těla vlastního rozhraní za pomocí volání

`PreciseSliderView(viewModel: viewModel, unitLabel: closure)`, kde parametr `viewModel` odpovídá dříve zmíněnému stavovému objektu a `closure` je funkční blok pro specifikaci popisku jednotek. Parametry konstruktoru stavového objektu jsou popsány v kapitole 4.1, každý z těchto parametrů má ovšem vlastní výchozí hodnotu, tudíž žádný z nich není explicitně vyžadován. Specifikace stylu je možná předáním objektu třídy `PreciseSliderStyle` jako proměnnou prostředí, použitím metody `preciseSliderStyle` definované jako rozšíření protokolu `View`. Využití stylů je poté možné v aplikaci, využívající technologii SwiftUI, provést následovně:

```
var body: some View {
    ...
    PreciseSlider(...)
        .preciseSliderStyle(
            PreciseSliderStyle(...)
        )
    ...
}
```

Použití rozhraní této komponenty v technologii UIKit je poté možné vytvořením instance třídy `UIPreciseSliderViewController`. Pro přidání posuvníku do hierarchie zobrazení je možné využít výchozí atribut `view` (např. zavoláním `myView.addSubview(slider.view)`). Pro specifikaci stylů a vizuální podoby je nutné tyto změny provádět nad atributem `preciseSlider`, který obsahuje tutéž instanci, obohacenou o specifikací typu `UIPreciseSlider` pro vyloučení nutnosti přetypování. Typ `UIPreciseSlider` tedy obsahuje vlastní implementaci třídy `UIView`, obohacenou o možnost stylování. Konfigurace výchozího stavu komponenty je následně možná vlastní implementací protokolu `PreciseSliderDataSource` a jejím předáním vytvořenému objektu typu `UIPreciseSliderViewController` (konkrétně jejímu atributu `dataSource`). Pro získání aktuálního stavu a případných notifikacích o jeho změnách je poté možné obdobně využít vlastní implementaci protokolu `PreciseSliderDelegate` a přiřadit ji příslušnému atributu `delegate`.

Co se týče rozhraní posuvníku *PreciseSlider2D*, je postup v obou případech téměř totožný. Jedním z mála rozdílů v rozhraní pro framework SwiftUI je dualita v podobě specifikace popisku jednotek a stavového objektu typu `PreciseAxis2DViewModel` pro obě osy zároveň. Třída stavového objektu této varianty sdílí implementaci se stavovým objektem posuvníku *PreciseSlider*. Dalším dodatečným parametrem konstruktoru struktury `PreciseSlider2DView` je volba obsahu vizualizace. Třída `PreciseSlider2DStyle`, jako tomu je u varianty příslušící komponentě *PreciseSlider*, poskytuje rozhraní pro úpravu vizuální podoby. Jeho aplikace je tentokrát možná za pomocí metody `preciseSlider2DStyle`, opět aplikovatelné přímo nad výsledkem volání konstruktoru zobrazení.

Co se týče rozhraní pro platformu UIKit, liší se opět dualitou, tentokrát předáním objektů `PreciseAxis2DDelegate` a `PreciseAxis2DDataSource` pro obě osy atributům `axisXDataSource`, `axisXDelegate`, `axisYDataSource` a `axisYDelegate`. Zde ovšem přibývá dodatečný atribut `dataSource` s očekávaným typem `PreciseSlider2DDataSource`, sloužící pro specifikaci vizualizace kombinací složek. Volba stylů je opět možná nad atributem `preciseSlider2D` typu `UIPreciseSlider2D`, dostupném po konstrukci objektu `UIPreciseSlider2DViewController`.

6.2 Demonstrace použití

V rámci praktické části práce byla vytvořena sada demonstračních aplikací posuvníků *PreciseSlider* a *PreciseSlider2D*. Jedná se konkrétně o čtyři projekty pro ukázku použití obou těchto komponent s rozhraním pro technologie SwiftUI a UIKit. Demonstrace využití 2D posuvníku je předvedena v aplikacích pro volbu složek *hue* a *saturation* barevného modelu *HSV*. Tyto projekty mají název `ColorPickerExample-SwiftUI` a `ColorPickerExample-UIKit`. Pro ukázku použití posuvníku `PreciseSlider` byly vytvořeny projekty `VideoPlayerExample-SwiftUI` a `VideoPlayerExample-UIKit`, které demonstrují jeho aplikaci s využitím pro indikaci a ovládání progresu přehrávaného videa.

Projekt VideoPlayerExample-SwiftUI

Aplikaci posuvníku v projektu `VideoPlayerExample-SwiftUI` můžeme nalézt v souboru s názvem `VideoPlayer.swift`. V tomto případě manipulaci se stavovým objektem zaobaluje třída `VideoPlayerViewModel`, zajišťující konzistenci stavu videa a posuvníku. Tato třída provádí inicializaci stavového objektu dle dostupných informacích o zvoleném videu. Stavový objekt je struktuře `VideoPlayer` předáván za účelem získávání notifikací o jeho změnách použitím klíčového slova `@ObservedObject`. Použití komponenty v tomto projektu je následovné:

```
PreciseSliderView(
    viewModel: sliderViewModel,
    valueLabel: { value, step in
        ...
    }
)
.onChange(of: sliderViewModel.isEditing) { isEditing in
    ...
}
.preciseSliderStyle(
    PreciseSliderStyle(
        backgroundColor: ...,
        unitColor: { value, _ in
            ...
        }
    )
)
```

Důvod zmiňovaného způsobu předávání stavového objektu tkví v použití modifikátoru `onChanged`. Ten zajistí případné pozastavení přehrávání videa v případě notifikace o změně atributu `isEditing` stavového objektu. Jako parametr `valueLabel` je předáván příslušně formátovaný popisek každé jednotky. Barva textových popisků, jednotlivých jednotek a pozadí osy je volena dynamicky dle zvoleného systémového tématu, což naznačuje použití položek `backgroundColor` a `unitColor` při konstrukci stylovacího objektu.

Projekt VideoPlayerExample-UIKit

V tomto projektu můžeme vidět využití posuvníku *PreciseSlider* a jeho rozhraní pro framework UIKit. Použití posuvníku je možné nalézt v souboru `VideoPlayer.swift`. Konfigurace jeho vizuální stránky je prováděna téměř okamžite po konstrukci `UIPreciseSliderViewController` za pomocí atributu `preciseSliderView`. Zde je prováděna obdobná změna této stránky, ovšem přímou manipulací s tímto atributem:

```
let sliderViewController = UIPreciseSliderViewController
...
sliderViewController.preciseSliderView.axisBackgroundColor = ...
sliderViewController.preciseSliderView.unitColor = ...
```

Konfigurace výchozích hodnot se provádí předáním instance třídy `VideoView`, jež implementuje protokol `PreciseSliderDataSource`. Získávání dat a průběžná notifikace o jejich změnách je poté zajištěna implementací protokolu `PreciseSliderDelegate`, kterou provádí třída `VideoPlayer`. Tyto instance jsou v implementaci třídy `VideoPlayer` předávány komponentě `PreciseSlider` takto:

```
let videoPlayer = VideoView()
...
sliderViewController.dataSource = videoPlayer
sliderViewController.delegate = self
```

Projekt ColorPickerExample-SwiftUI

V tomto demonstračním projektu byla komponenta *PreciseSlider2D* použita způsobem modálního okna, jež zakrývá celou plochu aplikace. Důležité momenty jsou tentokrát pouze okamžiky, kdy je komponenta *PreciseSlider2D* zobrazena a opět skryta. Její použití je možné nalézt v souboru `ColorPickerModal.swift` a je následovné:

```
PreciseSlider2DView(
    axisX: hueAxis,
    axisY: saturationAxis,
    content: { _, _ in
        ...
    },
    axisXLabel: { value, _ in
        ...
    },
    axisYLabel: { value, _ in
        ...
    }
)
```

Zde položky `hueAxis` a `saturationAxis` představují instance třídy `PreciseAxis2DViewModel`, jež jsou inicializovány v konstruktoru třídy `ColorPickerModal`. Toto umístění je zvoleno za účelem předávání výchozích hodnot, odpovídajících dříve zvolené barvě z barevného modelu. Po potvrzení volby barvy uživatelem jsou data jednorázově přečtena ze stavových objektů `hueAxis` a `saturationAxis` a předána o úroveň výš.

Projekt ColorPickerExample-UIKit

Použití 2D posuvníku se v tomto projektu nachází v souboru `ColorPicker.swift` v metodě `initSlider`. Konfigurace posuvníku je zde prováděna předáním instance jednoduché implementace protokolu `PreciseAxis2DDataSource` s názvem `AxisDataSource`. Výchozí hodnoty jsou v tomto případě voleny jako konstanty. Předávání instance `PreciseAxis2DDelegate` bylo vyměněno za jednorázové přečtení vybraných hodnot přímo z objektu třídy `PreciseSlider2DViewController` (atributy `axisXValue` a `axisYValue`) při dokončení výběru uživatelem. Atributem `dataSource` je zde třída `ColorPicker`, jež implementuje metodu pro předání podoby obsahu vizualizace. Vizualizací je barevné schéma kombinací složek hue a satuaration, provedené jako `CAGradientLayer`. Jelikož protokol `PreciseSlider2DDataSource` vyžaduje, aby tato hodnota odpovídala typu `UIImage`, je následně proveden převod hodnoty na tento formát.

Kapitola 7

Závěr

Záměrem práce bylo vytvoření prvku grafického uživatelského rozhraní pro volbu jednorozměrné i dvourozměrné číselné hodnoty z daného intervalu, s cílem vysoké přesnosti. Výsledným produktem je zde plnohodnotný balíček obsahující implementaci dvou komponent, svým provedením zaměřených právě na tuto vlastnost. Obě komponenty jsou výběrem nejpodstatnějších vlastností hrající roli v oboru přesného výběru, a zároveň její návrh zohledňuje přítomné vady doposud dostupných řešení. Mimo to, balíček obsahuje robustní rozhraní pro jeho použití ve dvou aktuálních technologiích pro vývoj aplikací na platformu iOS. Toto rozhraní zároveň doplňuje sada demonstračních projektů pro snadné seznámení se způsobem jeho použití.

Práce byla mimo jiné užitečná i z hlediska prohloubení vlastních zkušeností s vývojem aplikací na platformu iOS a prozkoumání podstatných rozdílů frameworků SwiftUI a UIKit. Mimo zkušenosti s technologiemi mi dodala práce také detailnější pohled do světa vývoje uživatelských rozhraní.

Práce na komponentě přesto není u konce. S průběhem vývoje nové technologie SwiftUI bude zapotřebí tuto komponentu stále udržovat a ladit její případné nalezené nedostatky. Přestože provedení již obsahuje řadu kompromisů pro co nejmenší náročnost na prostředky zařízení, pouhá komponenta je pouze součástí většího, potenciálně náročnějšího celku. Z tohoto důvodu je jedním z budoucích plánů její pokročilá optimalizace. Jejím budoucím použitím bude také integrace do již existujících aplikací firmy *IN2CORE*, s jejíž spoluprací byla komponenta vyvíjena.

Literatura

[1] *About App Development with UIKit* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit.

[2] *Adding Interactivity with Gestures* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/documentation/swiftui/adding-interactivity-with-gestures.

[3] *Animatable in SwiftUI* [online]. SwiftOnTap, 2022 [cit. 2022-29-04]. Dostupné z: https://swiftontap.com/animatable.

[4] *Declaring a Custom View* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/documentation/swiftui/declaring-a-custom-view.

[5] Friese, P. *Why SwiftUI and Combine will help you to build better apps* [online]. Peter Friese, 13. září 2019 [cit. 2022-29-04]. Dostupné z: https://peterfriese.dev/posts/swift-combine-love/.

[6] *GestureState* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/documentation/swiftui/gesturestate.

[7] GetStream. *SwiftUI Spring Animations Cheat Sheet for Developers* [online]. GitHub, 2022. Dostupné z: https://github.com/GetStream/swiftui-spring-animations.

[8] Hudson, P. *Enabling high-performance Metal rendering with drawingGroup()* [online]. Hudson Heavy Industries, 13. listopadu 2021 [cit. 2022-29-04]. Dostupné z: https://www.hackingwithswift.com/books/ios-swiftui/enabling-high-performance-metal-rendering-with-drawinggroup.

[9] Hudson, P. *Frequently asked questions about SwiftUI* [online]. Hudson Heavy Industries, 23. září 2021 [cit. 2022-29-04]. Dostupné z: https://www.hackingwithswift.com/quick-start/swiftui/frequently-asked-questions-about-swiftui.

[10] Hudson, P. *Introducing MVVM into your SwiftUI project* [online]. Hudson Heavy Industries, 11. prosince 2021 [cit. 2022-29-04]. Dostupné z: https://www.hackingwithswift.com/books/ios-swiftui/introducing-mvvm-into-your-swiftui-project.

[11] Jabrayilov, M. *Styling custom SwiftUI views using environment* [online]. 2022 [cit. 2022-29-04]. Dostupné z: https://swiftwithmajid.com/2020/12/09/styling-custom-swiftui-views-using-environment/.

[12] *Metal* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/documentation/metal/.

[13] Naumov, A. *Clean Architecture for SwiftUI* [online]. GitHub, 4. listopadu 2019 [cit. 2022-29-04]. Dostupné z: https://nalexn.github.io/clean-architecture-swiftui/.

[14] *Package Manager* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://www.swift.org/package-manager/.

[15] *SwiftUI Overview* [online]. Apple Inc., 2022 [cit. 2022-29-04]. Dostupné z: https://developer.apple.com/xcode/swiftui/.

[16] *ViewModifier* [online]. Apple Inc., 2022 [cit. 2022-02-05]. Dostupné z: https://developer.apple.com/documentation/swiftui/viewmodifier.