Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Zpracování dat z vozů F1 v reálném čase s využitím rozšířené reality

Diplomová práce

Autor práce: Bc. Roman Auersvald
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Karel Mls, Ph.D.

Hradec Králové 2023

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 28.4.2023

.....

Bc. Roman Auersvald

Poděkování

Děkuji svému vedoucímu za metodické vedení práce a poskytnutí cenných rad. Dále děkuji své přítelkyni za inspirativní nápady a konstruktivní kritiku.

Anotace

Tato diplomová práce přináší nový způsob zpracování dat závodů Formule 1 vysílaných v reálném čase. Hlavním cílem práce je použití dostupného datového streamu pro vizualizaci závodního okruhu spolu se zobrazením aktuálních pozic závodníků.

V teoretické části je čtenář nejprve seznámen se světem Formule 1, včetně popisu rostoucího využívání dat pro optimalizaci výkonů vozů i závodníků. Následuje uvedení čtenáře do prostředí rozšířené reality, která je nedílnou součástí této práce, a úvod do problematiky streamování dat v reálném čase.

Praktická část práce obsahuje návrh řešení pro vizualizaci dat, jenž je vytvořen na základě zadefinovaných požadavků a podrobně popsán. Stěžejní částí této diplomové práce je následná implementace navrženého řešení. Veškeré problémy a překážky, které byly během vývoje aplikace zaznamenány, jsou náležitě uvedeny spolu s použitím alternativních řešení pro dosažení cíle práce.

Klíčová slova

Rozšířená realita, Formule 1, zpracování dat, SignalR, Swift, iOS

Anotation

Title: Real-time processing of data from F1 cars using augmented reality

This diploma thesis brings a new and innovative approach to processing real-time streamed Formula One data. The main goal is to utilize the available data stream in order to get the positions of racing drivers visualized in augmented reality.

In the theoretical part of the thesis, the reader is acquainted with the world of Formula One which uses data for the optimisation of both cars and drivers. Subsequently, both augmented reality technology and real-time data streaming, which are essential for this thesis, are properly introduced to the reader.

The practical part of the thesis contains the solution design of an application for visualizing the F1 data. The design based on the defined requirements is properly described. The fundamental part of this thesis is the implementation of the proposed solution. All of the difficulties encountered during the development are described with used solutions that satisfy the requirements of the implemented application.

Keywords

Augmented Reality, Formula One, data processing, SignalR, Swift, iOS

Obsah

1 Úvod

V posledních letech je prostředí Formule 1 hluboce závislé na datech. Tato data a umění data zpracovávat dávají týmům neměřitelný potenciál a konkurenční výhodu proti soupeřům. Závislost na datech je možné pozorovat na první pohled na téměř každé Velké ceně, kde se mezi sponzory objevuje na předních místech Amazon s platformou AWS. Její využití je vidět i během televizního přenosu, kde televizní produkce zobrazuje určité predikce. Jedná se například o predikci předjíždějícího manévru pro stíhajícího závodníka nebo kontroverznější grafiku, která zobrazuje zbývající životnost pneumatik. Poslední zmíněná grafika je kontroverzní z důvodu častého zobrazování opačného stavu pneumatik oproti tvrzení závodníka. Jezdci Formule 1 mají mnoho zkušeností s řízením konkrétního vozu a dokáží podle citu odhadnout míru přilnavosti a zbývající životnost pneumatik. Ne vždy má ale závodník dostupná všechna data o aktuální situaci, a proto se musí spolehnout na informace, které mu poskytne jeho tým.

Producenti televizního vysílání projevují snahu přiblížit data z vozů Formule 1 divákům za použití pokročilejších technologií, například rozšířené reality. Během vysílaných závodů mohou diváci sledovat překrytí „halo“ (ochranného prvku závodníka) prvku informacemi o rychlosti vozu, zařazeném rychlostním stupni, nebo aplikaci brzd či plynu. Další příklad uplatnění rozšířené reality je v situaci, kdy se stíhající závodník připravuje uskutečnit předjížděcí manévr. Během tohoto okamžiku, při zobrazení pohledu kamery z kokpitu, je stíhaný vůz označen virtuálním prvkem a pomocí virtuálních šipek je reprezentována vzdálenost mezi vozy.

Na základě stanovených požadavků je navržena aplikace, jež uživateli zpřístupní zcela nový pohled na závody Formule 1. Aplikace bude využívat připojení k serverům Formule 1 pro získání aktuálních dat, která bude možné využít k vizualizaci závodní trati a aktuálních pozic závodníků. Pro následnou implementaci navržené aplikace budou vybrány vhodné technologie (jako jsou například protokol SignalR, SwiftUI nebo použití rozšířené reality), jež zaručí dosažení požadovaných výsledků. Všechny použité technologie, metody, knihovny a další jsou v práci náležitě představeny.

2 Formule 1

2.1 Historie

Počátky Formule 1 sahají do první poloviny 20. století, kdy dochází ke vzniku organizačních struktur a formátu závodů. Hlavním útvarem, fungujícím dodnes, je Mezinárodní automobilová federace známá pod názvem FIA (z francouzského *Fédération Internationale de l'Automobile*). Poprvé byl také nazván závod jako „*Grand Prix*“, česky „Velká cena“. Konkrétně se jednalo o Velkou cenu Francie v Le Mans roku 1923. V letech 1935–39 se začíná rodit myšlenka poháru jezdců. Do uvedení v praxi bohužel zasahuje druhá světová válka a s ní je pozastaveno veškeré závodění v Evropě [1][2].

Samotné jméno *Formula One* je ustanoveno za oficiální až v roce 1946 a v roce 1947 je ustálena myšlenka poháru jezdců. Rok 1950 je významný uvedením poháru jezdců v praxi a to prvním závodem ve Velké Británii na okruhu Silverstone. V tomto roce se koná více než 20 závodů, nicméně do šampionátu jsou počítány pouze závody na následujících okruzích: Silverstone (Velká Británie), Monaco (Monako), Bremgarten (Švýcarsko), Spa-Francorchamps (Belgie), Remeš (Francie), Monza (Itálie) a Indianapolis 500 (Spojené státy americké). První ročník šampionátu vyhrál s celkovým počtem 30 bodů italský závodník Giuseppe Farina soutěžící za tým Alfa Romeo SpA. Bodové hodnocení bylo zavedeno pouze pro prvních 5 závodníků s bodovou dotací 8-6-4-3-2 a možností získání dodatečného jednoho bodu za zajetí nejrychlejšího kola závodu [3].

Mezi lety 1950 a 1958 došlo k rozšíření závodů na další dva kontinenty, konkrétně se jednalo o závody v Argentině (Jižní Amerika) a Maroku (Afrika). Rok 1954 přinesl omezení v maximálním možném objemu motoru a to na 2,5 litrů. První sezónu závodů, kde se mimo pohár jezdců uskutečnil také pohár konstruktérů, odstartoval rok 1958. Současně bylo v tomto roce zakázáno střídání jezdců v jednom voze, což bylo do této doby možné. Byla omezena i délka závodů z původních 500 kilometrů na 300 kilometrů, nebo na maximální dobu trvání závodu dvě hodiny [2].

Velkou změnou ve Formuli 1 bylo například představení závodního vozu s hliníkovým rámem stájí[1]. Lotus, která byla mimo jiné i první, jež na svůj vůz umístila reklamu. právě umístění reklamy odstartovalo příliv sponzorů do tohoto sportu. Došlo také k zařazení závodů v Japonsku a Austrálii. Dalším technologickým po-

[1]Označení pro závodní tým ve Formuli 1.

krokem bylo představení vozu stájí Renault, která do svého pohonného ústrojí integrovala turbodmychadlo [1].

Spolu s narůstajícím výkonem vozů bylo třeba dbát na bezpečí jezdců. Proto bylo v roce 1978 zavedeno pravidlo, že vždy první kolo po startu závodu následuje závodníky speciální lékařské vozidlo, které je v případě nehody schopno poskytnout první pomoc rychleji. O dalších 14 let později bylo zavedeno takzvané *safety car*, které je na závodní trať vysláno v případě nebezpečí na trati nebo nutnosti zpomalit závodníky po dobu oprav či úklidu trati. I přes tyto bezpečnostní kroky došlo v roce 1994 na okruhu San Marino ke dvěma tragédiím. Nejdříve během kvalifikace přišel o život Roland Ratzenberger a o den později během závodu pak i ikonický Ayrton Senna.

2.2 Éry Formule 1

Společně s vývojem pravidel, přidáváním dalších závodů a formováním závodních týmů docházelo k významným pokrokům ve vývoji závodních vozů. Je poměrně logické, že se týmy, motivované vítězstvím a vyhranými poháry konstruktérů, snaží o vývoj co nejlepších závodních vozů. Tento postupný vývoj lze rozdělit do několika etap podle toho, jaká technologie právě dominovala.

- **Éra aerodynamiky**

  Počátky aerodynamické éry odstartovalo použití prvních zadních přítlačných křídel v první polovině 60. let 19. století. Nicméně se ukázalo, že konstrukce těchto křídel nebyla plně vyhovující a docházelo k jejich oddělení od zbytku vozu a zapříčinění několika nehod. Nový design zadního přítlačného křídla byl použit následně v roce 1970 na vozech Lotus 72, kde vůz také disponoval předním přítlačným křídlem [1].

  Uvedení předních a zadních přítlačných křídel znamenalo výrazné zvýšení odporu vzduchu vůči vozu. To umožňovalo závodníkům projíždět zatáčky mnohem větší rychlostí, právě díky vyšší přilnavosti. Samozřejmostí při zvýšení odporu vzduchu u vozu je také zapříčinění snížení rychlosti na rovných úsecích trati. Proto je nutné najít kompromis v designu aerodynamických prvků v závislosti na druhu trati nebo na výkonu vozu [1].

  Využití aerodynamických prvků nekončí pouze implementací předních a zadních přítlačných křídel. Postupem času dochází k implementování funkčních otvorů za prostorem pro jezdce za účelem přivést více vzduchu ochlazujícího motor a ostatní součásti vozu. Zároveň se rozšiřuje utilizace podtlaku

pro zvýšení přilnavosti k vozovce. Skvělým příkladem přítlačného efektu byl vůz Lotus 78/79, který použitím obou křídel a speciálního podvozku docílil jako jeden z prvních velkého úspěchu a v roce 1978 vyhrál devět z celkových patnácti závodů [1].

Zmíněný podtlak vzniká aplikováním Bernoulliho efektu, který lze aplikovat jak na tekutiny, tak na plyny. V tomto případě se jedná o využití efektu na okolním vzduchu a na určitých prvcích závodního vozu. Obecně Bernoulliho efekt popisuje obtékání vzduchu kolem aerodynamických prvků v různých rychlostech. Strana, již obtéká vzduch pomaleji, generuje větší tlak než strana, kterou obtéká vzduch rychleji. Toto způsobí přitlačení obtékaného předmětu (vozu) směrem k vozovce. Vzduch mezi vozem a vozovkou je stlačen a proudí mezi těmito předměty rychleji, tím generuje podtlak. Tento princip je také využíván u křídel letadel, kde je ale požadován opačný efekt oproti vozu Formule 1 [4].

Další efekt napomáhající přítlačné síle vozu Formule 1 je Venturiho efekt. Jedná je o jev, kdy proudění tekutiny je nepřímo úměrné rychlosti proudění tekutiny. Tekutinu v tomto případě nahrazuje vzduch. Tento jev je aplikovaný mimo jiné na podvozek, kde dochází k urychlení proudění vzduchu. Tím je generován podtlak a vůz získává dodatečnou přítlačnou sílu pro rychlý průjezd zatáčkou při zachování maximální možné rychlosti [5]. Zmíněné efekty hrají významnou roli i v současných vozech, proto je kladen důraz na vývoj vozu s co možná nejlepší aerodynamikou.

- **Éra turbodmychadel**

  Zavedení turbodmychadel do vozů Formule 1 přišlo společně s aerodynamikou. První stáj, která se začala turbodmychadly zaobírat, byl Renault s vozem RS01 v roce 1977. Začátky ale pro tuto stáj nebyly jednoduché. Turbodmychadla mají obecně problém anglicky nazývaný *turbo lag*, který můžeme přeložit jako prodlevu turbodmychadla. Ten se projevuje, jak již název napovídá, určitou prodlevou mezi sešlápnutím plynového pedálu závodníkem, roztočením turbíny v turbodmychadle a následným dodáním většího množství vzduchu do motoru (tam je využit pro získání lepšího spalovacího poměru mezi dodaným vzduchem a palivem) [1].

  Je třeba uvést také další technický problém spojený s použitím turbodmychadla. Vozy s turbodmychadly dosahovaly do té doby nevídaných výkonů. Například motor M12/13 od výrobce BMW dosahoval výkonu až 1 300 koní, což s motorem o obsahu 1 499 cc[2] byla kombinace, která nebyla dlouho

[2]Jednotka cc označuje objem motoru. Celým názvem je Centimetr krychlový.

udržitelná. Docházelo proto k použití různých nastavení řídicí jednotky pro určité situace. Například při kvalifikaci byl motor nastaven na nejvyšší možný výkon, kdy tlak produkovaný turbodmychadlem dosahoval 5–5,5 bar, a při závodu byl výkon motoru snížen na 850–900 koní [6]. Toto zvýšení výkonu při kvalifikaci dávalo jezdci možnost zajet nejlepší čas, aby si mohl zajistit co nejlepší možnou pozici pro start, a kdy zároveň nebylo nutné, aby motor v těchto extrémních podmínkách pracoval delší dobu. Naopak při závodu, kdy je zapotřebí, aby motor fungoval bezchybně, byl výkon snížen.

Použití turbodmychadel bylo v průběhu následujících let regulováno kontrolní organizací FIA. Došlo k omezení maximálního možného tlaku, který mohlo turbodmychadlo produkovat, a to na 4 bary. Zároveň bylo povoleno použití motoru bez turbodmychadla o objemu 3,5 litru. Až do roku 1989, kdy bylo použití turbodmychadel zakázáno, dokázaly stáje dosáhnout velkých úspěchů. Například stáj McLaren v roce 1988 s jezdci Ayrtonem Sennou a Alainem Prostem vyhrála 15 z celkových 16 závodů, což zajistilo nejen vítězství v poháru jezdců Ayrtonu Sennovi, ale také výhru v poháru konstruktérů pro McLaren [7].

Od roku 1989 až do konce roku 1994 byly povoleny pouze motory o objemu 3,5 litru bez turbodmychadel. Turbodmychadla se dočkala znovupoužití až v roce 2014 s příchodem hybridní éry [1].

- **Převodovka, odpružení a elektronika**

  Zásadní zlom v ovládání závodního vozu přišel společně s uvedením poloautomatické převodovky. Jako první ji při závodu představila stáj Ferrari s vozem Ferrari 640. Doposud byly využívány manuální převodovky, jež známe i z dnešních vozů, které mají řazení ve stylu písmene H [1].

  Poloautomatická převodovka umožňovala závodníkům nejen výrazně rychlejší řazení, ale také omezovala případné nechtěné zvýšené otáčky během standardního přeřazení. Využitím poloautomatické převodovky došlo k odstranění spojkového pedálu a jako řadicí element byla využita speciální tlačítka na volantu. Za vývojem použité poloautomatické převodovky stojí vrchní návrhář Ferrari John Barnard, který byl do té doby ve Formuli 1 známý díky navržení kompozitního těla vozu ještě za působení ve stáji McLaren. Lze tedy tvrdit, že to byl právě Barnard, jenž položil základy poloautomatické převodovky, která se postupem času začala využívat v ostatních odvětvích motorsportu [8][9].

Počátky 90. let 19. století byly prozatímním vrcholem technologie ve Formuli 1. Mimo uvedení poloautomatické převodovky několik let zpět došlo také k využívání aktivních prvků, které dále pomáhaly závodníkům na trati. Jedním z nich bylo využití aktivního odpružení. To mělo počátky ve snaze zabránit náklonu vozu vpřed a vzad při brždění a při akceleraci. Pozdější implementace již byla mnohem sofistikovanější a aktivní odpružení bylo hlídané elektronikou, která měla za úkol vyrovnávat vůz v zatáčkách, ve snaze udržet optimální podmínky pro fungování navržené aerodynamiky [1].

Pomocné systémy pro závodníky se dále skládaly z kontroly trakce, která omezovala excesivní protáčení poháněných kol. Toto zamezení protáčení umožňovalo větší přilnutí pneumatiky na trati a lepší využití výkonu vozu. Spolu s kontrolou trakce byl také využíván systém kontroly brzdění, dnešní ABS[3], který omezoval situace, kdy docházelo k zablokování kola při prudkém zpomalení [1].

Tato éra dospěla k poměrně rychlému konci, kdy v roce 1994 došlo k zakázání použití aktivního odpružení společně s ABS a kontrolou trakce. Mezi lety 2001 a 2008 byl krátce systém kontroly trakce opět povolen [10].

- **Hybridní éra**

  Během pěti let mezi roky 2001 a 2013 došlo ve Formuli 1 k určitým pokusům o integraci hybridního pohonného ústrojí. Týmy měly možnost integrovat do svého pohonného ústrojí systém KERS (*Kinetic Energy Recovery Systems*), který převáděl kinetickou energii generovanou během brždění vozu na energii elektrickou nebo mechanickou. Tu poté bylo možné uchovávat pro pozdější využití. Závodní vozy tak mohly během jednoho závodního kola využít 60 Kw (82 koní), a to po dobu 6,6 sekundy. Použití systému KERS nebylo povinné a ze všech závodních týmů jej během sezóny použily pouze následující čtyři: Renault, Ferrari, McLaren a BMW. Přijetí systému KERS bylo poměrně vlažné a následující sezónu 2010 nebyl tento systém využit žádnou stájí. Změna přišla společně se sezónou 2013, kdy už všechny závodní týmy disponovaly systémem KERS [11].

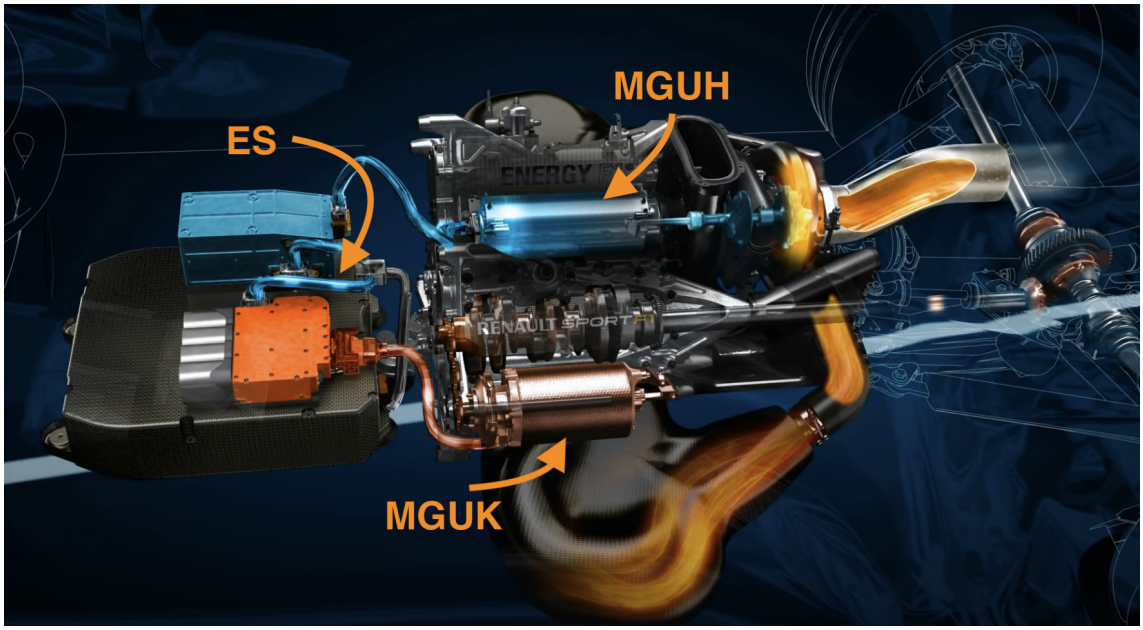
  Využití systému KERS všemi týmy v sezóně 2013 nebylo náhodou. V roce 2014 vstoupily v platnost regulace v podobě nových restriktivních omezení na pohonné ústrojí a na využití nových systémů pro hybridní pohon. Systém KERS prošel změnou a byl rozdělen na dva systémy sběru energie. Novým souhrnným názvem pro tyto systémy se stává ERS (*Energy Recovery System*).

[3]Systém aktivní bezpečnosti vozidla, který zabraňuje zablokování kol vozu během brždění.

Zmíněnými systémy, které nově zahrnoval systém ERS jsou následující:

- **MGUK** (*Motor Generator Unit – Kinetic*) – jedná se o pokračování technologie KERS, kde je získávána energie ze zadní nápravy vozu, která se následně uchová prostřednictvím baterie ES (*Energy Store*) a lze ji využít později.
- **MGUH** (*Motor Generator Unit – Heat*) – jednotka napojená na turbodmychadlo, kde se prostřednictvím výfukových plynů a samotného turbodmychadla generuje elektrická energie uchovaná v ES pro pozdější využití.

Uspořádání ERS lze vidět na obrázku 1, kde jsou vyobrazeny jednotky MGUK a MGUH spolu s baterií ES.



Obrázek 1: Vizualizace uspořádání MGUK, MGUH a ES spolu s motorem Renault V6, sezóna 2014 [12].

Celý systém ERS zůstává ve Formuli 1 od roku 2013 až do současnosti. Následující zamýšlená změna nastane v roce 2026, kdy vstoupí v platnost nové technické regulace a z ERS bude odstraněna jednotka MGUH [13].

2.3 Významné osobnosti

Pokud se někdo zmíní o Formuli 1, je velmi pravděpodobné, že si většina lidí vybaví určitá jména. Schumacher, Hamilton, Verstappen, Alonso, Vettel... Nejsou to ale pouze závodníci, kteří se proslavili ve světě Formule 1. Jsou to i lidé, kteří tento sport formovali a podíleli se na jeho vývoji, ne vždy ve světle reflektorů.

Jednou z těchto osobností je Bernie Ecclestone. Ecclestone roku 1971 koupil závodní stáj Brabham a poté, co vstoupil do světa současné Formule 1 a viděl velký potenciál v rozvoji fungování závodních týmů, rozhodl se jednat. Společně s Maxem Mosleym založil sdružení závodních týmů s názvem „*Formula One Constructors' and Entrants' Association*". Toto sdružení zastupovalo týmy a jednalo v jejich zájmu s pořadateli závodů. Odměnou byl podíl na výhrách, konkrétně 2 %. Do té doby jednaly jednotlivé týmy s pořadateli individuálně a vyřizovaly například smlouvy o účasti v závodě, aby měly možnost soutěžit o body do poháru jezdců a konstruktérů [14].

Poté, co se Ecclestone ujal vedení organizace, začaly provize růst. Z poměrně silné pozice vůči promotérům mohl například požadovat televizní práva s určitou provizí. Strategií s televizními právy zajistil Ecclestone rozšíření sportu do světa, a tím i možnost nárokovat si větší peníze od promotérů. Tento krok zajistil hlavní zdroj financí pro Formuli 1. Následně pak Ecclestone začal požadovat od televizních společností odvysílání celého závodu, a později i vysílání všech závodů v sezóně. Tím zajistil větší konzistenci v počtu diváků, které televize upoutala. Mohl také využít počty diváků pro zavedení reklamy přímo na okruzích, což přineslo značné financování nejen jemu, ale také všem týmům Formule 1. Ecclestonův vliv na finance ve Formuli 1 sílil a během jeho působení nastaly situace, kdy se jeho aktivity nelíbily ostatním kontrolním organizacím. Proto od roku 2014 ztrácel značnou část své působnosti, ve prospěch takzvané Strategické skupiny a Komise F1. Roku 2017 vystřídala Ecclestona společnost Liberty Media [14].

Lze konstatovat, že to byl právě Ecclestone, který tento sport dostal do výšin a přinesl takové prostředky, aby mohla Formule 1 fungovat na vysoké úrovni. I přes všechny spory, které za svou kariéru zaznamenal, patří Ecclestonovi velké uznání.

Mezi významné osobnosti patří bezesporu také závodníci, kteří psali dějiny Formule 1. V tabulce 1 jsou uvedeni šampioni, kteří jsou známi širší veřejnosti.

Závodník	Počet titulů	Vyhrané sezóny
Michael Schumacher	7	1994, 1995, 2000, 2001, 2002, 2003, 2004
Lewis Hamilton	7	2008, 2014, 2015, 2017, 2018, 2019, 2020
Sebastian Vettel	4	2010, 2011, 2012, 2013
Fernando Alonso	2	2005, 2006
Max Verstappen	2	2021, 2022
Kimi Räikkönen	1	2007
Jenson Button	1	2009
Nico Rosberg	1	2016

Tabulka 1: Tabulka známých šampionů Formule 1 s počtem titulů a roky, kdy byly tituly vyhrány [15].

2.4 Mezinárodní automobilová federace (FIA)

FIA, celým oficiálním názvem *Fédération Internationale de l'Automobile*, je organizace zastřešující řadu sportů, mezi které patří jak Formule 1, tak světový pohár v Rallye, Rallycrossu nebo také Formule E a další. FIA vydává pravidla, která jsou závodníci i závodní týmy povinni dodržovat. V šampionátu Formule 1 je možné sbírat body, které jsou započítávány do celkového pořadí v následujících dvou kategoriích [16]:

- **Pohár jezdců**
  Světovým šampionem Formule 1 se může stát závodník, který získá nejvyšší součet udělených bodů za umístění v závodech během sezóny. Tyto body jsou udělovány v rozmezí 1. až 10. místa během hlavního závodu závodního víkendu. Závodní víkend může obsahovat také sprint, ve kterém je uvažováno s udělením bodů za místa v rozmezí 1. až 8. [16].

- **Pohár konstruktérů**
  Pořadí týmů v poháru konstruktérů určuje součet bodů, které nasbírají za celou sezónu vozy příslušného týmu. Nejsou počítány přímo body jezdců daného týmu, protože může dojít k situaci, kdy hlavní jezdec daného vozu není schopen účasti v závodu a je nahrazen záložním jezdcem. Pokud záložní jezdec získá bodové ohodnocení, jsou tyto body započítány do poháru konstruktérů [16].

  V závislosti na umístění týmů v tomto žebříčku jsou po skončení závodní sezóny rozděleny finanční prostředky. Každý tým proto vyvíjí snahu získat co možná nejvyšší umístění. Rozdíly v ohodnocení jsou i v desítkách milionů dolarů.

FIA působí jako kontrolní orgán, který dohlíží na dodržování přísných regulací. V případě, že dojde k porušení určitých nařízení jezdcem, jsou uděleny penalizace formou bodů. Provinění proti regulacím může být různého charakteru. Může se jednat o ohrožení dalších jezdců na trati, zavinění nehody, či například nedostavení se na několik rozhovorů s novináři. V případě, že dosáhne závodník součtu deseti bodů za souvislých dvanáct měsíců, hrozí mu například zákaz účasti na závodě. Body penalizace mají trvanlivost dvanáct měsíců, po této době jsou pak ze závodníkova konta odečteny.

Penalizace však nepostihuje pouze závodníky. Pokud závodní tým poruší platná nařízení hrozí ve většině případů finanční pokuta. V extrémních případech hrozí odebrání bodů z poháru konstruktérů.

Závodní týmy mají také přidělené počty kusů náhradních dílů pro závodní vozy. Jedná se především o omezení počtu dílů pro pohonné ústrojí. Pokud chce závodní tým vyměnit například převodovku určitého vozu, má možnost takto učinit například čtyřikrát za sezónu. V případě, že je počet výměn překročen, je aplikována penalizace za každou další výměnu. Tato penalizace je obvykle ponížení startovní pozice daného vozu v hlavním závodě.

2.5 Harmonogram sezóny

Závodní sezóna je rozdělena na několik period, ve kterých se odehrává určitá aktivita s vozy Formule 1. Následuje seznam těchto period spolu s krátkými popisy:

- **Testování**
  Perioda testování se odehrává v rozmezí od 1. února daného roku a musí končit nejméně čtyři dny před konáním prvního závodu sezóny. Každý ze závodních týmů má možnost využít k testování právě jeden závodní vůz každý den průběhu testování.

  Další příležitosti testování jsou během sezóny za účelem testu nových pneumatik pro jejich výrobce. Všechny termíny, které lze využít k testování, jsou dále definovány v platných regulacích pro konkrétní rok [16].

- **Závody**
  Seznam všech závodních víkendů je definován před začátkem sezóny. V roce 2023 obsahuje závodní sezóna celkem 24 závodů, z čehož 6 víkendů bude obsahovat závod ve sprintu.

Klasický závodní víkend je složen z následujících událostí v daném pořadí:

- Volný trénink 1 – pátek
- Volný trénink 2 – pátek
- Volný trénink 3 – sobota
- Kvalifikace – sobota
- Hlavní závod – neděle

Závod ve sprintu je formát, při kterém je závodní vzdálenost snížena na 100 kilometrů a bodové ohodnocení je dostupné pouze pro prvních osm závodníků. Výsledné umístění ve sprintovém závodu je přeneseno do startovních pozic v závodě hlavním. Složení závodního víkendu se závodem ve formátu sprintu je následující:

- Volný trénink 1 – pátek
- Kvalifikace pro sprint – pátek
- Volný trénink 2 – sobota
- Sprint – sobota
- Hlavní závod – neděle

- **Letní přestávka**
  Platná nařízení zahrnují letní přestávku, ve které nesmí být vykonávány jakékoliv činnosti v oblasti vývoje vozů. Tato přestávka musí být přesně čtrnáct souvislých dnů a musí být realizována v měsících červenci a/nebo srpnu. Pro rok 2023 je definována druhá nucená přestávka ve výrobním procesu začínající 24. prosince a trvající devět dní.

2.6 Data ve Formuli 1

Společně s technickým vývojem vozů Formule 1 je kladen stále větší důraz na sběr a následné využití dat. Tato data jsou zaznamenávána prakticky kdykoliv, kdy je závodní vůz nasazen do akce.

Zdrojem dat ve voze Formule 1 jsou senzory umístěné na zájmových místech. Senzory mohou zaznamenávat široký rozsah dat od měření teploty a tlaku po kontrolu fungování převodovky nebo ERS jednotky. Tyto senzory jsou napojeny do řídicí jednotky prostřednictvím CAN[4] sběrnice. Těchto sběrnic je v závodním

[4] *Controller Area Network* – automobilový standard pro spojení více zařízení pomocí jedné síťové linky.

voze několik a jsou rozděleny na tematické okruhy. Například řídicí jednotka nepotřebuje pro svoji činnost znát teplotu pneumatik, ale více využije třeba informace o teplotě oleje. Během závodního víkendu je v závodním voze umístěno až na 250 senzorů. Valná většina těchto dat je pouze pro interní použití v rámci analýzy konkrétního týmu a na veřejnost je publikován jen zlomek dat [17].

Množství zaznamenaných dat během závodního víkendu dosahuje až jednoho terabytu. Tento objem dat zahrnuje jak data z řídicích jednotek, tak video záznamy a další podpůrná data. Jakmile dojde ke zpracování těchto dat, může se celkový objem znásobit až třikrát. Data ale nejsou generována pouze během závodů. Podstatná část objemu dat je také vytvářena během interního testování ve vývojovém centru, například při testování ve větrném tunelu, nebo při testování zatížení pohonné jednotky [17].

Zmíněné množství dat je stěžejní při vývoji simulačních modelů a analýze chování vozu na trati. Je kladen velký důraz právě na sběr dat. Každá minuta, kdy je závodní vůz v provozu, znamená opotřebení součástek a spolu s omezenými možnostmi testování podle regulací FIA je nutné určité úkony simulovat. Každý závodní tým má k dispozici výkonné datacentrum s týmem odborníků, kteří zajišťují analýzu dat a jejíž výsledky předávají části týmu, která je s vozy na závodním okruhu.

Samotní jezdci jsou do procesu analýzy dat také zapojeni. Jakmile opustí závodní vůz, jsou jim prezentována data s vysvětlením, kde je možný prostor pro zlepšení. Důležitá je také zpětná vazba od závodníka pro dodání kontextu k určitým datům.

3 Rozšířená realita

Rozšířená realita je jednou z moderních technologií, které můžeme vidět ve svém okolí téměř každý den. Jedná se o zakomponování virtuálních objektů do prostředí reálného světa. Využití rozšířené reality můžeme vidět například v marketingu, kde je u určitých produktů nabízena možnost zobrazení jejich virtuální podoby v pohodlí domova. Například obchodní řetězec IKEA vydal aplikaci *IKEA Place*, která umožňuje umístění výrobků z katalogu na konkrétní místo v domácnosti, a tím umožní zákazníkovi vyzkoušet, jak se mu bude produkt zakomponovaný do domácího prostředí líbit. Dalším, stále častějším, využitím rozšířené reality v marketingu je možnost zobrazit si prohlížené šperky přímo na těle [18].

Nemusí se jednat pouze o nabídku produktů k prodeji. Rozšířená realita, nebo také augmentovaná realita (z anglického *Augmented Reality*, AR), nás dále provází například ve světě sociálních sítí. V této oblasti ji můžeme vidět aplikovanou v různých filtrech, které s využitím senzorů mobilního zařízení umožní uživateli přidat určité prvky na svůj obličej či jiné části těla.

O popularizaci rozšířené reality na mobilních zařízeních se také postarala hra pro mobilní telefony *PokemonGo*, která hráčům umožňovala chytat virtuální Pokémony v rozšířené realitě.

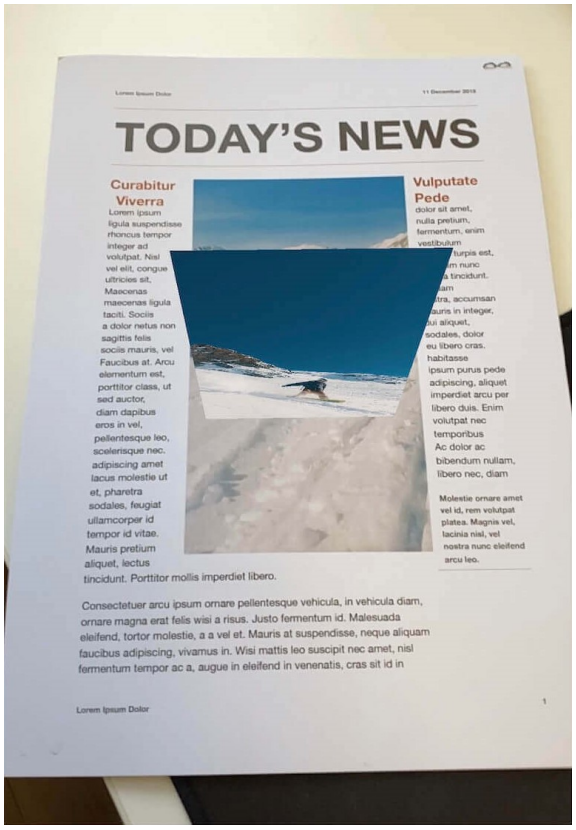
Rozšířená realita je často zaměňována za realitu virtuální. I přes společné znaky, kdy obě technologie využívají zobrazování virtuálních předmětů, je mezi nimi mnoho věcí odlišných. Virtuální realita využívá pro zobrazení objektů buď mobilní telefon, který je doplněn pro zobrazení obsahu určitým adaptérem, nebo specializovaný hardware skládající se z několika součástí. Těmi jsou většinou speciální brýle v kombinaci s ovladači, jež se drží v rukou a s jejichž pomocí je možné s objekty ve virtuální realitě interagovat. Znamená to tedy, že je uživatel obklopen celou virtuální scénou a není nijak spojen s reálným světem. To je hlavní rozdíl oproti rozšířené realitě, kde je uživatel přímo spojen s reálným světem a virtuální objekty mohou s objekty reálného světa interagovat. Rozšířená realita také nevyžaduje specializovaný hardware, ale stačí mobilní telefon, který je vybavený fotoaparátem.

Přes menší náročnost na hardware oproti virtuální realitě využívá realita rozšířená více senzorů, které jsou pro zobrazení scén nutné. Jedná se především o fotoaparát zajišťující snímání scény, která je následně pomocí algoritmů pro rozpoznání obrazu zpracovávána, ale také o GPS, která může dále zpřesňovat pozici uživatele. Dalším senzorem, jenž značně pomáhá zobrazení rozšířené reality, je

LiDAR[5]. Tato technologie dokáže mapovat hloubku snímané scény, a tím zpřesnit nejen umístění virtuálních modelů, ale i detekci ploch. Senzor LiDARu promítá na své okolí lidským okem neviditelné paprsky, které se od cílového předmětu odrazí zpět do senzoru. Ten měří celkový čas cesty paprsků a na základě tohoto údaje dokáže určit vzdálenost konkrétního předmětu.

3.1 Druhy rozšířené reality

Rozšířená realita lze rozdělit na několik druhů v závislosti na parametrech, které jsou používány pro detekci objektů, nebo použité technologii.



Obrázek 2: Demonstrace využití rozšířené reality a obrázku jako kotevního bodu [19].

- **S využitím kotevních bodů** (*Marker-based AR*)
  Tento druh rozšířené reality funguje na principu hledání známých objektů (tzv. kotevních bodů). Kamera neustále skenuje okolí, a jakmile zachytí známý kotevní bod, dojde k vyvolání příslušné akce. Akcí může být například animace nebo zobrazení virtuálního objektu v blízkosti kotevního bodu.

[5]LiDAR (*Light Detection and Ranging* – světelná detekce a měření rozsahu) je senzor měřicí čas návratu vyslaných paprsků.

Rozšířené kotevní body mohou využívat podobu QR kódu, nebo i podobu jakéhokoliv reálného objektu či obrázku. Právě využití obrázku jako kotevního bodu je možné vidět na obrázku 2, kde je využit mobilní telefon Apple a framework ARKit. Dochází k již zmíněnému rozpoznání obrázku a zobrazení virtuálního objektu, v tomto případě videa, v určité vzdálenosti nad obrázkem. Virtuální objekt je umístěn relativně vůči kotevnímu bodu, proto je možné libovolně pohybovat mobilním zařízením a virtuální objekt zůstává na místě [20].

- **Bez využití kotevních bodů** (*Markerless AR*)
  Rozšířená realita bez využití kotevních bodů spoléhá na informace o poloze z dalších senzorů na zařízení, například údaje z GPS. Mimo tyto údaje o poloze pracuje také s rozpoznáváním prostoru bez využití konkrétních objektů jako kotevních bodů. Dále je možné tuto kategorii rozdělit na následující podkategorie [20]:
  - **Projekce rozšířené reality na reálné objekty** (*Projection-based AR*)
    Tento druh rozšířené reality využívá projekce světla na určitá místa sledovaného prostoru. Vytváří tak pohled, který může například zvýraznit chybně vyrobená místa na výrobku, a tím usnadnit obsluze práci. V určitých aplikacích je také možné s promítaným obrazcem interagovat.
  - **Rozšířená realita vázaná na lokaci uživatele** (*Location-based AR*)
    Lokační rozšířená realita je vázána na specifické prostory. Umožňuje tak detekovat pozici uživatele v daném prostoru a přehrávat určité animace nebo zobrazovat virtuální objekty. Právě do této kategorie patří již zmiňovaná mobilní hra *PokemonGo*.
  - **Překrývaná rozšířená realita** (*Overlay AR*)
    Jedná se o režim rozšířené reality, při kterém jsou virtuální objekty umístěny na povrch objektů reálných. Tento druh je možné využít například pro zobrazení virtuálních textů, jež doplňují reálný objekt.
  - **Rozšířená realita využívající kontury objektů** (*Contour-based AR*)
    Posledním poddruhem bezkotevní rozšířené reality je rozšířená realita, jež pro orientaci v prostoru využívá siluety, které poté zvýrazní. Využití nachází především v počítačovém vidění pro vývoj systémů zabývajících se bezpečností na vozovce.

3.2 3D modely v rozšířené realitě

Obecně jsou 3D modely definovány třemi rozměry – výškou, šířkou a hloubkou. Využití modelů v rozšířené realitě tuto definici nijak nemění. 3D modely jsou hojně využívány v různých odvětvích, například pro 3D tisk, pro tvorbu animací, nebo i ve filmu pro přidání speciálních efektů. Modely se vyskytují v mnoha formátech a ne vždy je možné daný formát použít pro zamýšlenou aplikaci. S tím souvisí způsob, jakým jsou modely ve formátech reprezentovány. Určité formáty reprezentují model pouze pomocí sítě trojúhelníků, další například pomocí bodů, některé mohou obsahovat několik podružných modelů spolu s definicí barev pro určité části nebo dokonce animací a zvláštních povrchů. V následujících bodech je představeno několik formátů, které se pro modely v rozšířené realitě obecně používají:

- **OBJ**
  Soubor typu OBJ lze poznat podle stejnojmenné přípony *.obj*. Původně tento formát vznikl pro účely společnosti Wavefront k použití v aplikaci *Advanced Visualizer*. Tento formát podporuje reprezentaci modelu pomocí několika druhů polygonální geometrie (jako jsou body, čáry a další) společně s reprezentací pomocí geometrie volného tvaru (jako jsou křivky a plochy). Formát OBJ nepodporuje animace ani neumožňuje reprezentaci informací souvisejících se světlem nebo polohou scén. Často se s tímto formátem můžeme setkat například v oblasti 3D tisku [21].

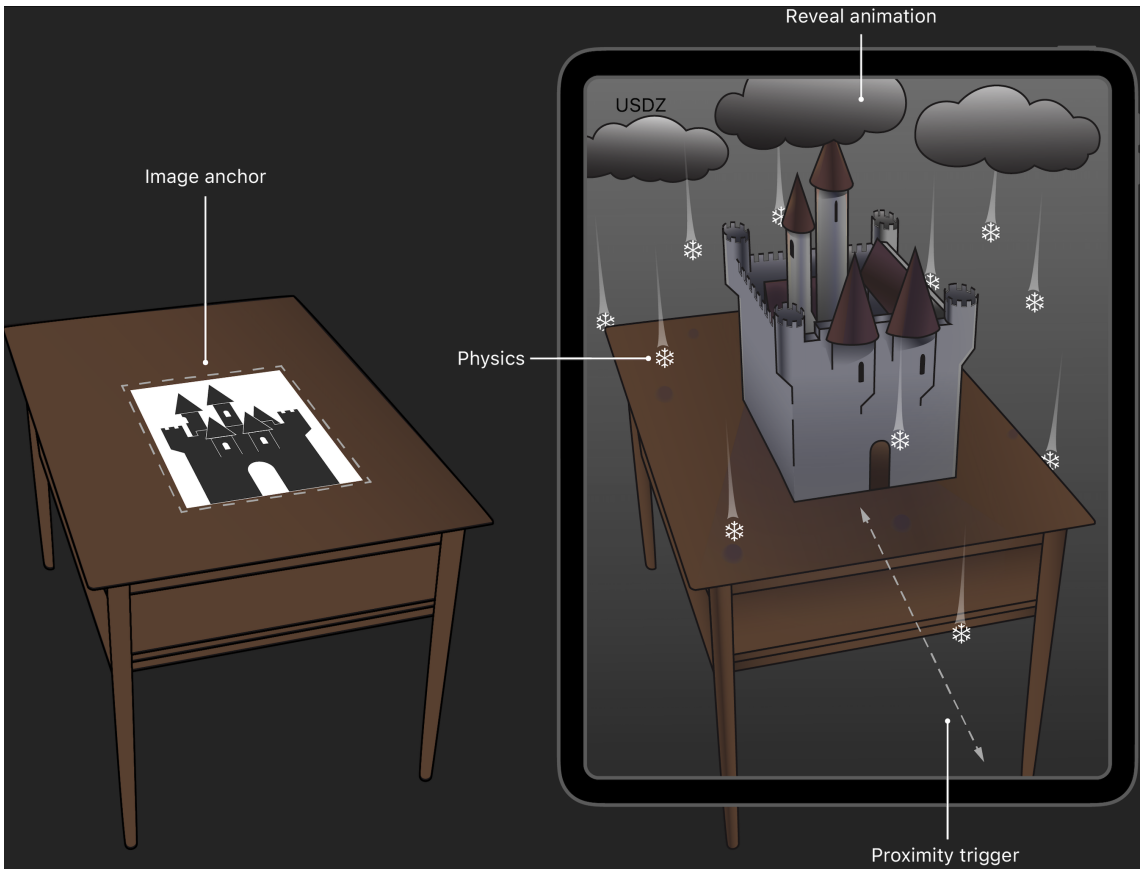
- **glTF**
  Formát souborů glTF (*GL Transmission Format*) ukládá informace o 3D modelu ve formátu JSON[6]. Díky reprezentaci modelu v JSON formátu jsou tyto soubory malé velikosti, což optimalizuje jejich načítání i distribuci. Jsou vhodné například pro načtení za běhu aplikace, kde jejich zpracování nezabere mnoho prostředků zařízení. Formátem glTF mohou být definovány celé scény obsahující modely, hierarchii uzlů, materiály, kamery, animace a další. Zároveň tento formát podporuje referencování dalších modelů, a tím si zajišťuje svoji malou velikost. Využití tohoto formátu je široké, můžeme jej najít například v aplikacích využívajících WebGL[7], ale i Unity a další. Mezi společnosti, které integrují tento formát pro své aplikace, patří například Facebook, Oculus, Microsoft a Sketchfab [21][22].

[6] JSON (*JavaScript Object Notation*) je textový formát popisu objektů.

[7] WebGl je multiplatformní knihovna používaná pro zobrazení 3D grafiky ve webovém prohlížeči.

- **USDZ**

  Formát USDZ vychází z formátu USD, který vyvinula společnost Pixar. Samotné USD (*Universal Scene Description*) lze přeložit do češtiny jako „univerzální popis scény“. Konečné „Z“ z názvu USDZ značí kompresi, kterou tento formát využívá. Oproti původnímu USD je přidáno několik možností, které mohou daný model obohatit. Jedná se například o možnost definovat kotevní prvek, reagovat na vstupy z okolního reálného světa, přidat simulaci fyziky, připojit k modelu zvukové efekty, nebo anotovat model textovým popisem. Jedná se tedy o formát, jenž „zapouzdřuje“ celou zamýšlenou scénu, která je vázána k určitému modelu.



Obrázek 3: Demonstrace možných akcí, které zapouzdřuje formát USDZ [23].

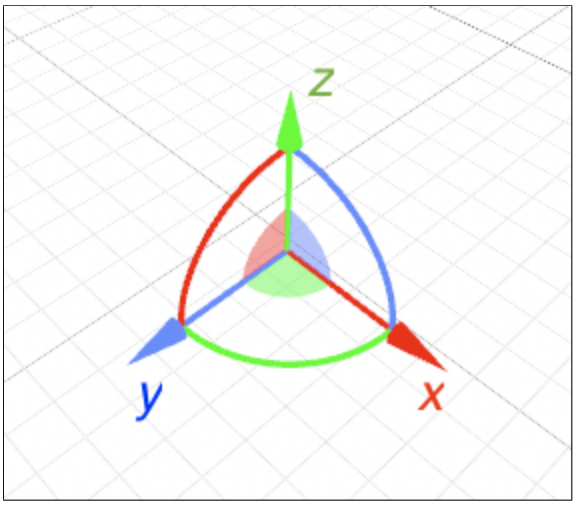
Na obrázku 3 je možné vidět objekt reálného světa (v tomto případě stůl), na kterém je umístěn 2D obrázek hradu. Ten slouží jako kotevní bod pro rozšířenou realitu. Na zařízení vpravo je vidět, jak se bude chovat model v rozšířené realitě. Zařízení pozná, že obrázek na stole je odpovídající kotevní bod, a v okamžiku, kdy se ke stolu přiblíží uživatel držící zařízení, dojde k umístění virtuálního modelu hradu na místo kotevního bodu. V okolí virtuálního hradu dojde dále ke spuštění animace, v jejímž rámci se objeví nad

hradem mraky a z nich padající sněhové vločky. Na vločky působí zemská gravitace a po dopadu do blízkosti reálného objektu – stolu, zmizí. Jednotlivé vločky jsou také modely, které jsou dále zapouzdřené v původním USDZ souboru [23].

3.3 Orientace v prostoru s rozšířenou realitou

Jak již bylo zmíněno v předchozí podkapitole o formátu USDZ, lze nastavit spuštění animací v případě, že se uživatel přiblíží do určité vzdálenosti od kotevního bodu nebo od samotného modelu. Z pohledu vývojáře jde především o reprezentaci objektů v souřadnicích. Práce se souřadnicemi je popsána v následujícím odstavci.

Vývojář, který doposud pracoval se 2D prostorem, kde jsou souřadnice pouze v osách x a y a počátek souřadnicového systému je umístěn v levém horním rohu se souřadnicemi $[0, 0]$, bude muset přidat do úvahy další osu, osu z. Ve 3D prostoru vyjadřuje osa z obvykle výšku daného objektu. Například při modelování objektů pro 3D tisk v nástroji *Autodesk Fusion 360* lze z 2D skici vytvořit 3D model právě extruzí[8] plochy v ose z. Vyobrazené osy jsou vidět na obrázku 4, kde osa y je vyobrazena modrou barvou, osa x červenou barvou a osa z barvou zelenou.

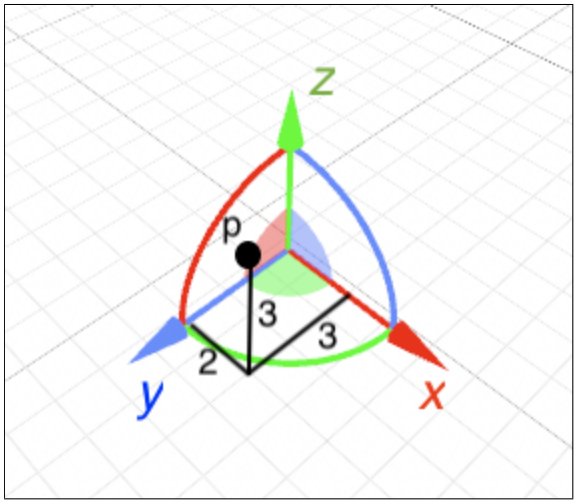


Obrázek 4: Osy 3D prostoru.

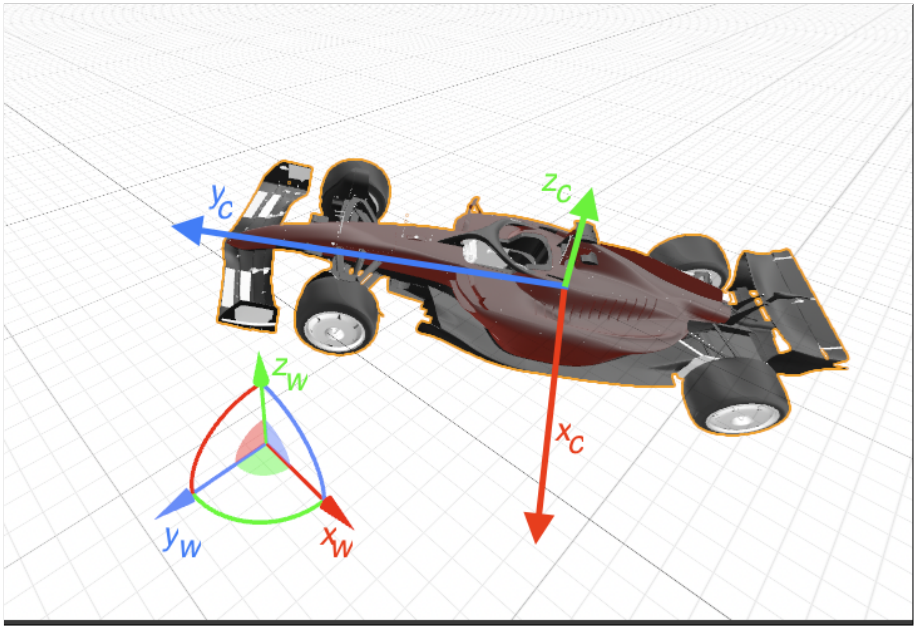
Použité jednotky ve virtuální a rozšířené realitě odpovídají jednotkám metrickým. Tedy 1 virtuální jednotka = 1 metr v reálném světě. Na obrázku 5 je vidět umístění bodu p, který má souřadnice $[3, 2, 3]$ ($[x, y, z]$). Znamená to, že bod je posunutý oproti počátku v ose x o 3 metry, v ose y o 2 metry a v ose z o 3 metry.

[8]Extruze je termín používaný v oblasti 3D modelování. Jde o „vytažení“ plochy objektu v určité ose. Tím vzniká 3D model.

Toto umožňuje programátorovi přesně umisťovat objekty v reálných jednotkách a usnadňuje mu práci s případným přepočtem z jiného souřadnicového systému.



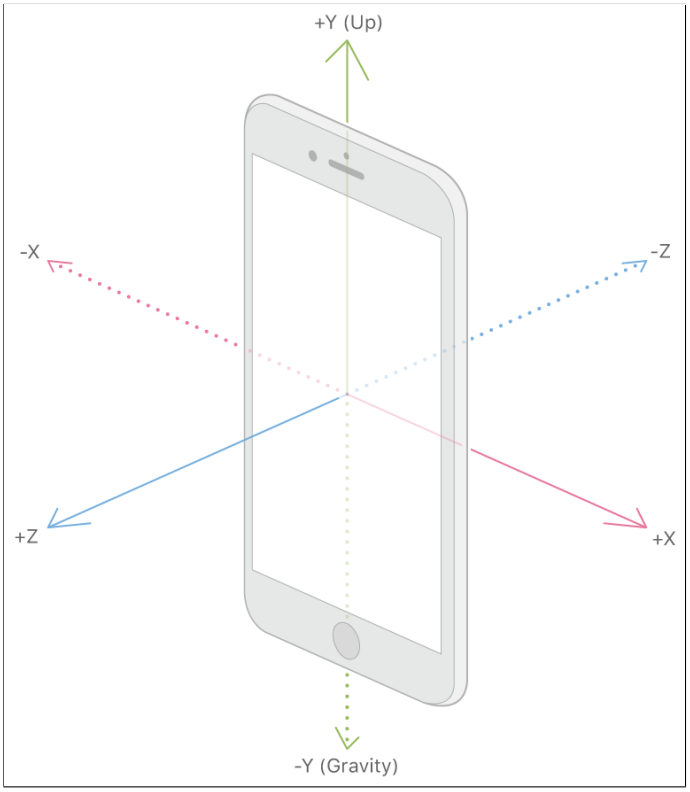
Obrázek 5: Umístění bodu p v $[x, y, z]$.



Obrázek 6: Zobrazení os světa a os modelu.

Jakmile je virtuální model přidán do scény, je umístěn na souřadnice definované oproti počátku scény. Pokud bychom ale k danému modelu přidali potomka, bude tento potomek umístěn v souřadnicích relativních vůči nadřazenému objektu. Znamená to, že pokud by byl umístěn například textový popisek k formuli na obrázku 6, bude mít uvažovaný popisek střed vozu jako výchozí bod, ke kterému budou vztažené souřadnice. Na obrázku je tento relativní souřadnicový systém označen osami x_c, y_c a z_c. Zároveň je na obrázku také zobrazen souřadnicový systém světa (osami x_w, y_w a z_w), vůči kterému je umístěna samotná formule.

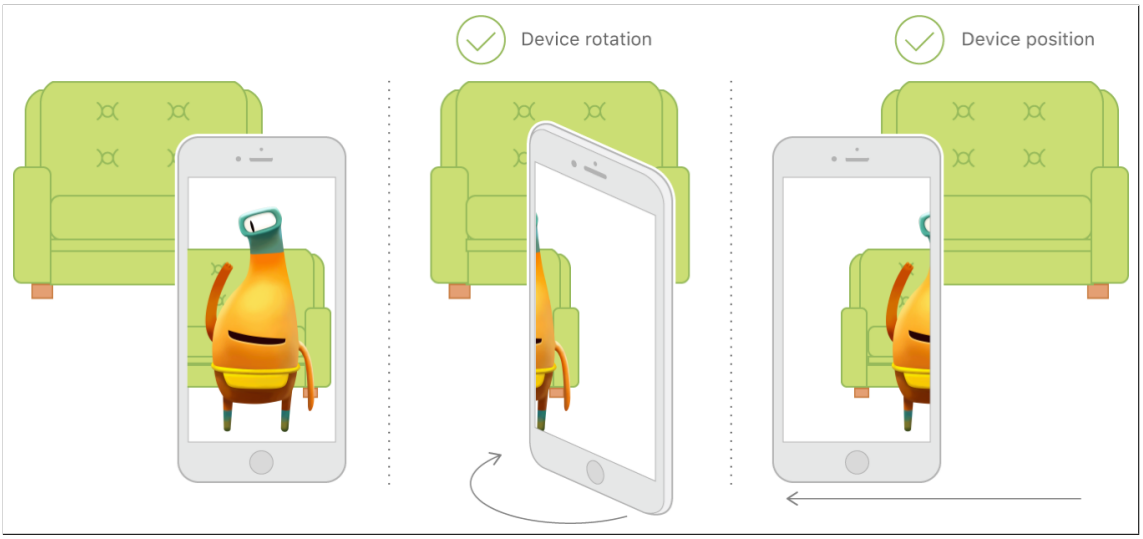
Při renderování scény se v počítačové grafice využívá kamery, jež je umístěna ve scéně a má nastavený směr, kterým scénu snímá. Tuto pozici kamery lze například animovat tak, aby bylo dosaženo požadovaného průchodu scénou. V počítačových hrách je především využívána pozice kamery, která simuluje pohled hráče. Můžeme se také setkat s umístěním kamery v pozici „třetí osoby“, kdy je hráčem ovládaný předmět viděn osobou stojící za ovládaným předmětem. Tyto pozice kamer jsou poté ovládány hráčem tak, aby simulovaly pohled (například pohled postavy hráče) ve scéně.



Obrázek 7: Zobrazení os v rozšířené realitě na mobilním zařízení [24].

Rozšířená realita má pozici kamery jasně danou. Využívá totiž pozici fyzické kamery v mobilním zařízení a na základě pohybů zařízení je zobrazovaná scéna renderována. Pokud chce vývojář umístit virtuální objekt tak, aby jej viděl uživatel před zařízením, je nutné upravit souřadnice objektu v ose z. Jak je možné vidět na obrázku 7, počátek virtuální scény je ve středu zobrazovacího zařízení. Pro umístění modelu před uživatele musí tedy vývojář aplikovat posunutí modelu v záporném směru zmíněné osy z. Z obrázku 7 je též možné vyčíst, že směr udaný osou y od počátku do záporných hodnot je směrem, ve kterém působí na zobrazované předměty gravitace.

Na obrázku 8 je umístěn virtuální objekt před zařízením, které se dále pohybuje. Je vidět zachování pozice objektu v reálném prostředí nezávisle na pohybu zařízení.



Obrázek 8: Umístění objektu a pohyb zařízení [25].

3.4 Rozšířená realita na zařízeních Apple

Každá vývojová platforma má svá specifika. Zařízení Apple využívají primárně formát USDZ (již zmíněn v sekci 3.2, která pojednává o formátech souborů pro rozšířenou realitu). Společně s uvedením formátu USDZ představil Apple další vývojové nástroje, které usnadní přípravu modelů a vývoj aplikací pro rozšířenou realitu. Těmito nástroji jsou aplikace pro počítače Mac popsány v následujícím seznamu:

- ***Reality converter***
  Aplikace *Reality Converter*, jak už anglický název napovídá, zajišťuje konverzi 3D souborů pro použití v rozšířené realitě. Konkrétně zajišťuje převod souborů typu *.obj*, *.gltf*, *.usd* a dalších do formátu USDZ. Tyto konvertované soubory je dále možné upravit například změnou materiálů, úpravou osvětlení a úpravou metadat. Jedná se o jednoduchou aplikaci, kde jsou původní soubory importovány jednoduše přetažením a v případě jejich úspěšné konverze je zobrazen výsledek [26].

- ***Reality composer***

  Reality Composer je v porovnání s aplikací *Reality Converter* více robustní. Tato aplikace slouží především k vytváření a editaci celých scén, které jsou součástí jednoho USDZ souboru. Je zde možné tvořit hierarchii objektů a jednotlivé objekty rozmístit ve scéně. Dále aplikace nabízí možnost definování interakcí mezi modely a uživatelem. Může se jednat o interakci dotykem nebo například o interakci spuštěnou změnou vzdálenosti uživatele od scény. Tato aplikace je dostupná jak pro počítače Mac, tak pro mobilní zařízení iPhone a iPad, kde nabízí možnost nahrát scénu v podobě senzorických dat z fotoaparátu. Tato data lze poté opětovně využít při tvorbě scény. Aplikace *Reality Composer* umožňuje finální exportování scény do formátu USDZ, který lze následně použít ve vývojovém prostředí XCode [27].

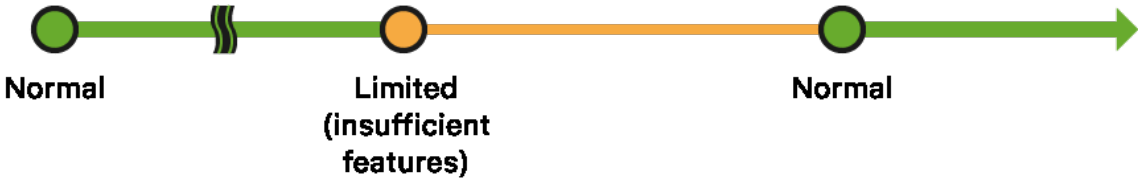
Dnešní modelová řada mobilních zařízení Apple již nabízí řadu mobilních telefonů a tabletů, součástí jejichž snímacího zařízení je i LiDAR senzor. Ten je v případě rozšířené reality důležitým prvkem v mapování scény. Na zařízeních, které tento senzor nemají, trvá inicializace mapování prostředí podstatně delší dobu. Pokud poté dojde k umístění virtuálního objektu na konkrétní místo, může docházet při pohybu zařízením k malým nepřesnostem v renderování. To je zapříčiněno právě absencí LiDAR senzoru. Zařízení totiž spoléhá na algoritmické rozpoznání obrazu pouze ze snímků fotoaparátu, a může tak mít problémy v rozpoznání splývajících ploch nebo v mapování prostředí se sníženou viditelností.

Naopak na zařízeních, která tento zmíněný senzor mají, dochází k podstatnému zlepšení času, který je potřebný jak k mapování scény, tak k celkové detekci ploch a překrývajících se objektů. Tento proces měření vzdálenosti senzorem LiDAR je neustále opakován, a tak je vytvářena 3D mapa snímané scény. Tato mapa umožňuje zařízení lepší porozumění reálného obsahu scény a identifikaci ploch a kotevních bodů. Také napomáhá detekci scény za zhoršených podmínek, jako je například špatné osvětlení.

Obrázek 9: Inicializace mapování scény rozšířené reality [28].

Než dojde k umístění virtuálních objektů do scény, je nutné danou scénu zmapovat a připravit tak data, která umožní přesné umístění objektů. Zmíněné mapování scény je spuštěno ihned po přijímání dat z fotoaparátu a pomocných

senzorů. Na obrázku 9 je vidět přechod ve stavech relace až do doby, kdy je scéna zmapována a plně k dispozici pro umístění objektů.



Obrázek 10: Změny kvality relace rozšířené reality [28].

Během renderování scén může dojít z jakéhokoliv důvodu ke zhoršení kvality mapování, a tím i k nepřesnému umístění objektů. Tento stav je vyobrazen na obrázku 10, kde je oranžově označena část, ve které došlo ke zhoršení kvality. Jakmile je kvalita mapování obnovena, pokračuje relace opět v plném fungování.

Obrázek 11: Vizualizace plného přerušení relace rozšířené reality [28].

Na mobilních zařízeních může dojít ke stavu, kdy je aktuální aplikace zobrazující rozšířenou realitu pozastavena a odsunuta do pozadí (například zobrazením jiné aplikace), a tím dojde k jejímu úplnému přerušení. Tento stav je zobrazen na obrázku 11, kde je na začátku časové osy zaznamenané normální fungování relace, jenž je přerušeno se změnou stavu na „nedostupné“. Po navázání je relace v omezeném stavu a je prováděna opětovná inicializace spolu s mapováním. Jakmile je relace plnohodnotná, dojde k normálnímu zobrazení virtuálních objektů.

Aby byla zajištěna lepší uživatelská zkušenost[9] s aplikacemi využívajícími rozšířenou realitu, mají vývojáři možnost uložení aktuálně mapovaného světa. V okamžiku, kdy dojde k odsunutí aplikace do pozadí, je o tomto stavu aplikace programově upozorněna a může dojít k uložení aktuálních informací do lokálního úložiště. Při opětovném spuštění dojde k načtení těchto informací a tím ke snížení času, který aplikace potřebuje pro přechod do plně funkčního stavu.

[9]Uživatelská zkušenost je často nazývána anglickým názvem *User Experience* (UX).

4 Streamování dat v reálném čase

Pojem streamování dat obecně znamená práci s proudem dat – jak příjem, tak i odesílání. Tento proud dat může v praxi znamenat sdílení videa, hudby, nebo jen surových dat. Odesílatel a příjemce v této komunikaci očekává téměř nulovou či minimální odezvu. Je tak zajištěna odpovídající interakce mezi odesílatelem a příjemcem. V praxi je dostupných mnoho nástrojů pro implementaci přenosu dat v reálném čase. Většina těchto nástrojů má určité společné znaky, jako je například sdílení stejné architektury nebo stejných principů (např. RPC[10]). Konečné rozhodnutí pro použití konkrétního nástroje má tedy vývojář, který vybírá na základě svých požadavků. Určité nástroje například nemají odpovídající podporu koncových zařízení nebo nevyužívají odpovídající formát zpráv. Následuje popis několika rozšířených nástrojů umožňujících komunikaci v reálném čase.

4.1 WebSocket

Pojem WebSocket označuje komunikační protokol zajišťující mezi serverem a klientem plně obousměrnou komunikaci s nízkou latencí. Tento protokol, který byl standardizován v roce 2011, položil základy pro mnoho dalších nástrojů, které jeho principu využívají. Samotný protokol byl navržen pro efektivnější komunikaci mezi serverem a klientem. Využívá standardních portů 80 a 443 a typu spojení koncových zařízení pomocí HTTP[11] či HTTPS[12] [29].

Samotná komunikace začíná úvodní zprávou s žádostí o připojení skrze obyčejný přenosový protokol TCP[13], kde jsou specifikovány speciální hlavičky protokolu `Upgrade: WebSocket` a `Connection: Upgrade`. Díky těmto přidaným hlavičkám pozná server pokus o navázání spojení za použití protokolu WebSocket. V případě, že server tento typ komunikace podporuje, následuje výměna údajů o verzích protokolu, případné ověření uživatelů a nastavení šifrování. Po dokončení úvodní fáze přenosu je vytvořen tzv. tunel mezi zařízeními, který je stálý a umožňuje obousměrnou komunikaci [29][30].

Příklad jednoduchého serveru využívajícího protokol WebSocket je možné vidět na ukázce kódu 1. Prvním krokem po importu `WebSocketServer` je vytvoření

[10] *Remote Procedure Call* (RPC), neboli vzdálené volání procedur, umožňuje zúčastněným stranám v konverzaci vyvolat definovanou proceduru na druhé straně připojení.

[11] *Hypertext Transfer Protocol* – internetový protokol pro komunikaci mezi webovými stránkami a servery.

[12] *Hypertext Transfer Protocol Secure* – šifrovaná bezpečnější verze HTTP

[13] *Transmission Control Protocol* - Protokol transportní vrstvy ze sady protokolů TCP/IP.

konstanty `wss` obsahující referenci na vytvořený server s komunikačním portem 8080. Dále je registrováno naslouchání metodě `connection`, ke které se připojí klient s požadavkem o připojení. Uvnitř této metody je pro příklad definována metoda, jenž může volat klient (`message`) a v jejímž rámci dojde na straně serveru k vypsání zprávy do konzole. Poslední funkcí použitou v ukázce je `ws.send()`, která umožňuje odeslat zprávu všem připojeným klientům.

```
1   import { WebSocketServer } from 'ws';
2
3   const wss = new WebSocketServer({ port: 8080 });
4
5   wss.on('connection', function connection(ws) {
6     ws.on('error', console.error);
7
8     ws.on('message', function message(data) {
9       console.log('received: %s', data);
10    });
11
12    ws.send('something');
13  });
```

Ukázka kódu 1: Příklad jednoduchého WebSocket serveru naprogramovaného s použitím Jode.JS knihovny [31].

4.2 Socket.io

Knihovna Socket.io umožňuje implementaci aplikací s nízkou latencí na základě komunikace vyvolané událostmi mezi serverem a klientem. Tato knihovna staví na základech komunikačního protokolu WebSocket, jež rozšiřuje o další metadata v každém paketu a také o dodatečnou funkcionalitu zjednodušující implementaci. Tato knihovna podporuje mnoho možností jak serverové implementace (například JavaScript, Java, Python, Golang), tak implementace na straně klienta v programovacích jazycích JavaScript (prohlížeč, Node.JS, nebo React Native), Java, C++, Swift, Dart, Python, .Net, Rust, Kotlin [32].

Oproti použití samostatné technologie WebSocket nabízí Socket.io následující vylepšení:

- **Využití HTTP *long-polling* v případě selhání**
  Pokud nastane chyba připojení pomocí WebSocket, Socket.io automaticky přejde na záložní použití technologie HTTP *long-polling*[14]. Toto vylepšení bylo v začátcích WebSocket, kdy jej mnoho prohlížečů nepodporovalo, důvodem, proč vývojáři upřednostňovali Socket.io.

- **Automatické navázání spojení**
  V případě náhlého ukončení připojení mezi serverem a klientem dojde k automatickému opětovnému navázání spojení. Socket.io implementuje mechanismus posílání takzvaných *heartbeat* zpráv, které periodicky kontrolují stav připojení.

- **Fronta nevyzvednutých zpráv**
  Toto vylepšení navazuje na předchozí zmíněné. V případě, že dojde k odpojení klienta od serveru, je zajištěno ukládání neodeslaných zpráv do zásobníku. Poté, co je klient opětovně k serveru připojen, jsou data ze zásobníku odeslána. Je tak zajištěno odeslání všech zpráv.

- ***Acknowledgements* – zpětná vazba**
  Jedná se o zjednodušení psaní kódu potřebného pro odesílání zprávy a doručení zpětné vazby po přijetí zprávy. V následujících ukázkách kódu 2 a 3 je možné vidět proces zpětné vazby. Server odešle zprávu a jako zpětnou vazbu obdrží data v parametru `response`, která jsou zpracována přidruženým blokem kódu. Příjemce zprávy `Receiver` odešle po obdržení zpět serveru zpětnou vazbu formou volání funkce `callback` s parametrem `"got it"`.

```
1  socket.emit("hello", "world", (response) => {
2      console.log(response); // "got it"
3  });
```

Ukázka kódu 2: Odeslání zprávy s připraveným blokem kódu, který zpracuje zpětnou vazbu [32].

[14]Varianta spojení mezi serverem a klientem, kdy spojení zůstává otevřené po dobu, dokud nejsou dostupná data.

```
socket.on("hello", (arg, callback) => {
    console.log(arg); // "world"
    callback("got it");
});
```

Ukázka kódu 3: Příjem zprávy a odeslání zpětné vazby přes funkci `callback` [32].

- **Odesílání zpráv**
  Na straně serveru implementuje Socket.io jednoduchý způsob odesílání zpráv určitým skupinám uživatelů na základě jejich příslušnosti do skupiny, nebo možnost odeslání zpráv všem připojeným uživatelům (takzvaný *broadcasting*). Příklad *broadcastingu* je zobrazen v ukázce kódu 4, kde je zpráva nejdříve odeslána všem připojeným a poté je jiná zpráva odeslána pouze připojeným uživatelům do kanálu `news`.

```
// to all connected clients
io.emit("hello");

// to all connected clients in the "news" room
io.to("news").emit("hello");
```

Ukázka kódu 4: Příklad odeslání zpráv všem připojeným uživatelům a uživatelům připojeným pouze do kanálu `news` [32].

- **Multiplexing**
  Při využití této funkcionality je umožněno oslovit danou zprávou pouze ověřené uživatele. Jak je vidět na úryvku kódu 5, je možné přidáním jednoduchého parametru `.of("/admin")` před metodu `.on` zajistit oslovení pouze skupiny uživatelů s označením `admin`.

```
io.on("connection", (socket) => {
    // classic users
});

io.of("/admin").on("connection", (socket) => {
    // admin users
});
```

Ukázka kódu 5: Příjem zprávy a odeslání zpětné vazby přes funkci `callback` [32].

4.3 Pusher

Platforma Pusher nabízí dva hlavní produkty. Prvním je Channels, jenž zajišťuje komunikaci v reálném čase s využitím kanálů. Druhým produktem je Beams, který nabízí možnost tvorby programovatelných push notifikací pro ostatní platformy – jako například mobilní zařízení či webové prohlížeče.

Prvně zmiňovaný produkt Channels je velmi podobný dříve popisovanému frameworku Socket.io. Oba dva jsou totiž určitou nadstavbou protokolu WebSocket, nabízí stejné typy komunikací – možnost využití kanálů pro komunikaci ve skupinách nebo pouze jeden na jednoho. Především tyto nástroje zajišťují komunikaci mezi serverem a klientem v reálném čase. Rozdílem platformy Pusher oproti Socket.io je například uzavření zdrojového kódu. Socket.io je open-source řešením[15], kdežto Pusher je komerční platforma, která má svou základnu zdrojového kódu uzavřenou a pro zákazníky nabízí své služby formou subskripcí [33].

```
1  const Pusher = require("pusher");
2
3  const pusher = new Pusher({
4    appId: "APP_ID",
5    key: "APP_KEY",
6    secret: "APP_SECRET",
7    cluster: "APP_CLUSTER",
8  });
9  const channels = ["my-channel-1", "my-channel-2", "my-channel-3"];
10 pusher.trigger(channels, "my-event", {
11   message: "hello world",
12 });
```

Ukázka kódu 6: Příklad konfigurace odeslání zprávy více kanálům přes platformu Pusher [34].

Možnosti implementace serverové části jsou složeny z několika kroků vyobrazených v ukázce kódu 6, který je připraven pro implementaci v jazyce Node.js. Prvním krokem je import knihovny `Pusher`, která je následně využita k vytvoření instance dané třídy. V části kódu, kde dochází k vytvoření této instance, je možno vidět nastavení parametrů například `appId:` značící identifikátor aplikace a `cluster:` definující příslušnost do tematického segmentu. Po inicializaci instance je pro přehlednost kódu vytvořeno pole obsahující všechny požadované kanály, ve kterých dojde k rozeslání zprávy. Posledním krokem v ukázce je odeslání zprávy do požadovaných kanálů s definovanou metodou `„my-event"` a obsahem zprávy.

[15] Open-source je způsob zpřístupnění zdrojového kódu veřejnosti.

4.4 SignalR

SignalR je open-source knihovna z dílen společnosti Microsoft, která usnadňuje integraci komunikace v reálném čase do aplikací. Funkcionalita komunikace v reálném čase je umožněna díky kódu spouštěném na serveru, který má možnost předávat data dál klientům bez prodlevy.

SignalR je vhodným kandidátem na integraci například v následujících aplikacích:

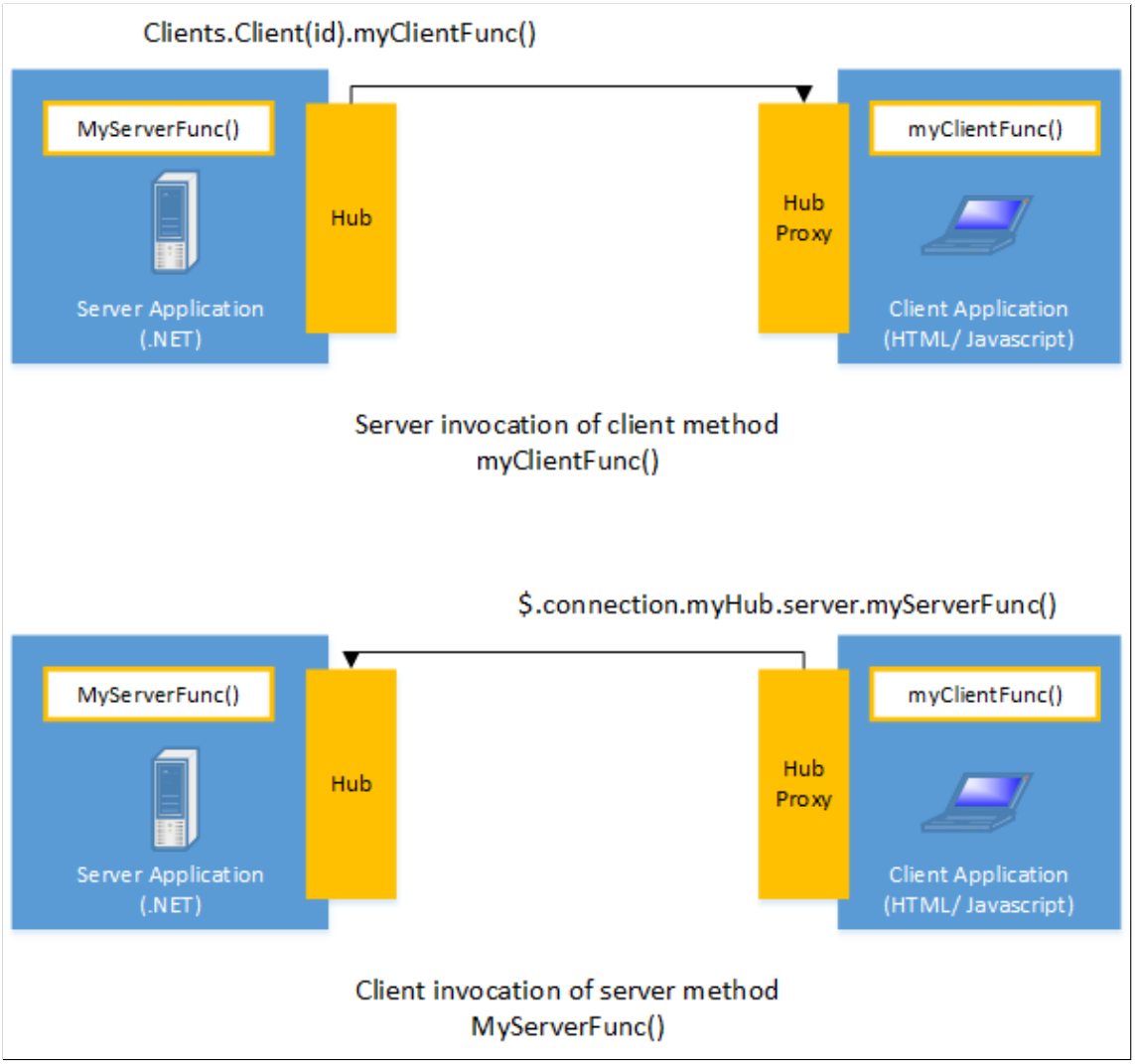
- Aplikace vyžadující aktualizaci dat s vysokou frekvencí – například hry, sociální sítě, aukce, nebo aplikace využívající polohové služby.
- Přehledové stránky, které slouží k monitorování událostí aplikací – například přehled s prodejem produktů nebo přehled stavu funkcionality jednotlivých prvků aplikace.
- Aplikace využívající sdílené prostředky – například kolaborativní spolupráce na prezentacích a dokumentech.
- Aplikace vyžadující notifikace – například sociální sítě, emailový klient, hry, nebo obecně mobilní aplikace využívající notifikace k předání aktuálních dat uživateli.

SignalR využívá RPC, neboli vzdálené volání procedur (popsáno v sekci 4.1). Stěžejní funkcionalitou je implementace SignalR Hub protokolu, který již zmíněnou RPC komunikaci umožňuje integrovat na obou zúčastněných stranách. Je tak možno z klienta volat předem definované funkce na serveru a ne jen opačným směrem.

Zmíněný SignalR Hub protokol využívá tak zvaný Hub, jenž zajišťuje obousměrnou komunikaci v reálném čase. Lze používat silně typové parametry metod, které umožňují modelové mapování, či parametry metod založené na formátu JSON nebo binárním protokolu MessagePack [35].

Tím, že je knihovna SignalR vyvíjena společností Microsoft, je nutné pro její integraci na straně serveru využít programovacího jazyku .Net nebo .Net Core. Jednotlivé programovací jazyky mají jinou podporu s frameworkem SignalR. Je to zapříčiněno tím, že .Net Core je novější vývojová platforma a podporuje novou verzi CoreSignalR. Ta určité funkcionality přidává, ale také pozměňuje nebo ruší. Použití dané vývojové platformy je vždy na vývojáři, jeho možnostech a požadavcích vývoje.

Obrázek 12 popisuje vzájemné vyvolání zpráv mezi serverem a klientem. Server prostřednictvím funkce `Clients.Client(id).myClientFunc()` vyvolá na určitém klientovi konkrétní událost. Opačně funguje komunikace analogicky tak, že klient má možnost vyvolat funkci prostřednictvím navázaného spojení k serveru `$.connection.myHub.server.myServerFunc()$`. Volání těchto metod musí předcházet navázání spojení mezi zúčastněnými stranami, kde si server uloží referenci na klienta do seznamu klientů pod unikátním klíčem a naopak klient vytvoří referenci na spojení, díky které může komunikovat se serverem [35].



Obrázek 12: Znázornění volání metod mezi serverem a klientem s využitím frameworku SignalR [36].

Mezi podporované platformy pro použití CoreSignalR patří JavaScript, .Net klient a Java klient. Přičemž .Net klient vyžaduje pro fungování, aby platforma podporovala ASP.NET Core. Dále je možné využít experimentální či neoficiální klienty pro C++ a Swift [37].

5 Návrh řešení pro vizualizaci živých dat

Pro dosažení stanovených cílů je zapotřebí zmapovat dostupné zdroje dat a definovat požadavky na zamýšlenou aplikaci, spolu s návrhem uživatelského rozhraní a architekturou aplikace.

5.1 Zdroje dat

Existuje několik cest k získání aktuálních dat ze závodů Formule 1. Od řešení, která využívají takzvané scrapování[16] dat z oficiálních webových stránek Formule 1, přes dostupné datové streamy až k API[17] třetích stran. Všechny streamy, ať už volně dostupné, nebo placené, neposkytují záruku stoprocentně validních dat. Mimo zdroje, jež poskytují streamování aktuálních dat, existují také nástroje, které zpřístupňují data historická. Tato data lze použít například k predikcím vítězů nebo k analýze jednotlivých kol závodníků.

- **Scraping**
  Pomocí takzvaného scrapingu lze získávat aktuální data o pozicích vozů Formule 1 použitím nástroje pro čtení webových stránek nebo oficiální aplikace Formule 1. Tuto metodu používá například projekt dostupný v repozitáři GitHub[18] uživatele Kurt Moran, který pomocí softwaru pro rozpoznání obrazu extrahuje data o jezdci z oficiální mobilní aplikace.

  Metoda scrapingu ovšem nezaručuje zajištění aktuálních dat. Vždy bude existovat prodleva mezi přijetím dat aplikací, vykreslením dat v aplikaci a následným zpracováním metodou čtení obrazu.

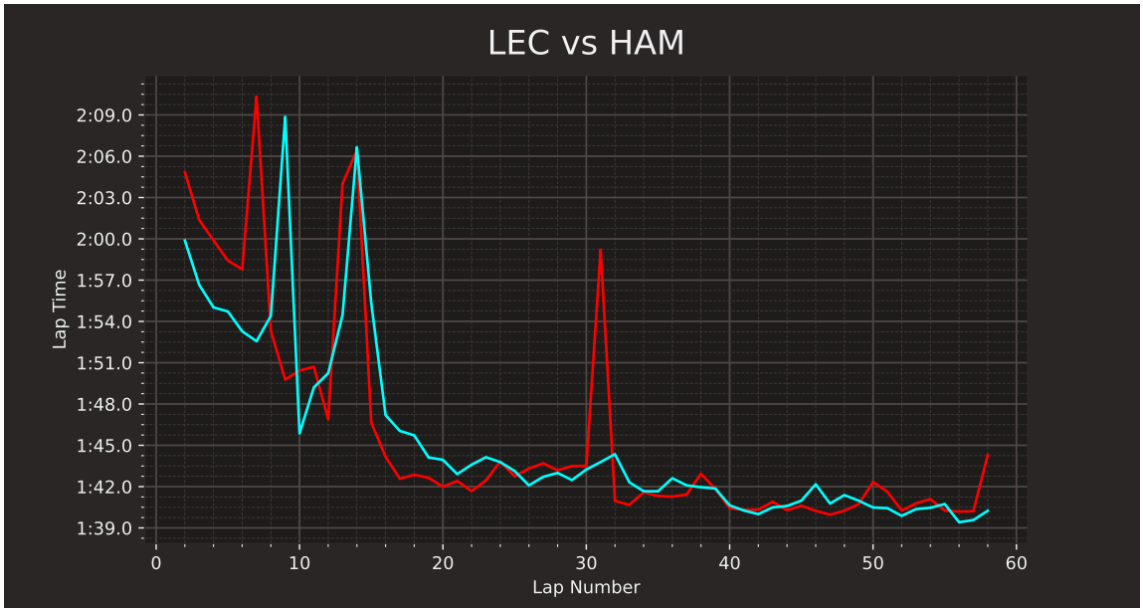
- **Historická data**
  Zdrojů dat, které poskytují záznamy o již proběhlých závodech Formule 1, je mnoho. Jedním z často používaných je Ergast Developer API z webu Ergast.com[19], který poskytuje historická data o závodech Formule 1 od roku 1950. Nejenže Ergast poskytuje prostřednictvím API vývojářům data zdarma, ale také je možné všechny záznamy stáhnout ve formátu .csv pro následnou analýzu jinými nástroji.

[16] Web scraping je způsob programatického získávání dat z webových stránek.
[17] API (*Application Programming Interface*) – v programování se jedná o označení pro komunikační rozhraní.
[18] Repozitář je dostupný na adrese: `https://github.com/KurtMoran/F1-Data-Extractor`
[19] Webová stránka pro stažení dostupných historických závodů: `http://ergast.com/mrd/`

Nástrojem, který historická data z portálu Ergast.com používá, je knihovna Fast F1 psaná v jazyce Python. Tato knihovna je dostupná v repozitáři GitHub uživatele theOehrly[20] . Prostřednictvím Fast F1 je možné vyhodnocovat jednotlivé jezdce v konkrétních závodech a zároveň tvořit podrobné analýzy jejich telemetrie. Příklad využití této knihovny je v ukázce kódu 7 a na obrázku 13.

```
fig, ax = plt.subplots()
ax.plot(lec['LapNumber'], lec['LapTime'], color='red')
ax.plot(ham['LapNumber'], ham['LapTime'], color='cyan')
ax.set_title("LEC vs HAM")
ax.set_xlabel("Lap Number")
ax.set_ylabel("Lap Time")
plt.show()
```

Ukázka kódu 7: Programatické vykreslení grafu zobrazujícího porovnání dat z telemetrie [38].



Obrázek 13: Graf porovnávající telemetrii jednoho kvalifikačního kola dvou jezdců [38].

Součástí Fast F1 je také možnost nahrávání live dat ze serveru Formule 1 a jejich následné uložení do textového souboru. Využití těchto live dat pro analýzu prozatím není možné.

[20]Repozitář FastF1 je dostupný na adrese: `https://github.com/theOehrly/Fast-F1`

- **Datový stream Formule 1**
  Jak již bylo v předchozím bodě zmíněno, existuje volně dostupný zdroj dat v podobě serveru SignalR Formule 1. Tento server zajišťuje streamování dat vozů Formule 1 pro účely vizualizace ve webové a mobilní aplikaci. Pro své účely jej využívá mnoho aplikací třetích stran, a to i přes fakt, že k němu není k dispozici jakákoli oficiální dokumentace.

5.2 Požadavky

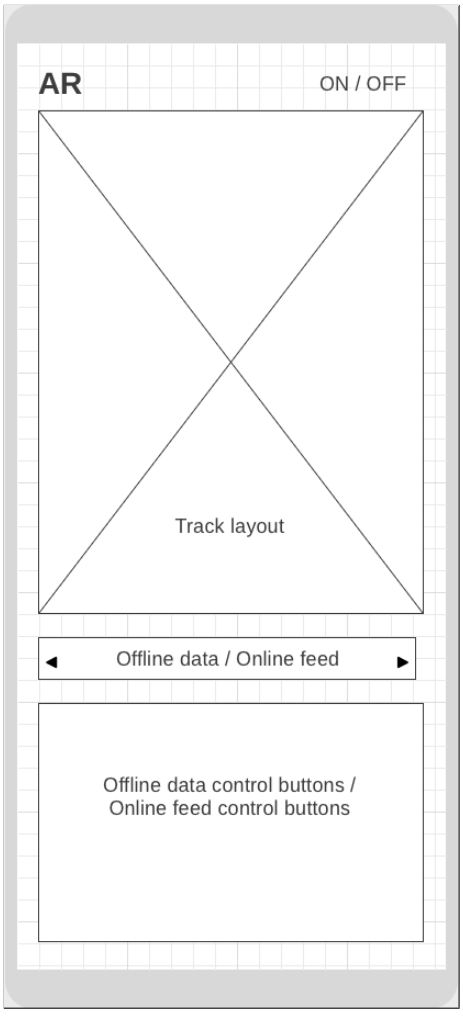
Pro návrh řešení je nutné stanovit požadavky, které bude výsledný systém splňovat. Tyto požadavky lze rozdělit do následujících dvou kategorií:

- **Funkční požadavky**
  - Výsledná aplikace bude podporovat operační systémy iOS a iPadOS.
  - Aplikace bude psána v programovacím jazyce Swift.
  - Aplikace bude využívat dostupný datový stream Formule 1.
  - Aplikace využije pro tvorbu uživatelského rozhraní framework SwiftUI.
  - Aplikace umožní přehrát závod i při nedostupném datovém streamu.
  - Aplikace umožní zobrazit detaily vybraného vozu.
  - Aplikace umožní měnit ukotvení virtuální trati v reálném prostředí.
- **Nefunkční požadavky**
  - Aplikace bude zobrazovat relevantní informace.
  - Aplikace bude obsahovat přehledné uživatelské rozhraní.
  - Aplikace bude reagovat bez dlouhé odezvy.

5.3 Uživatelské rozhraní

Samotné uživatelské rozhraní je obecně pro mobilní aplikace stěžejním bodem návrhu. Pokud dojde k chybám v návrhu uživatelského rozhraní, nebo uživatelské zkušenosti, může dojít k situaci, kdy potenciální uživatel aplikaci otevře a po několika vteřinách opět zavře, v horším případě i odinstaluje. Zkrátka bude na uživatele aplikace působit nepříjemně a uživatel radši aplikaci používat nebude, i přes to, že samotná funkcionalita aplikace je bezchybná. Zde navržená aplikace se proto bude snažit o využití vhodného poměru mezi těmito kategoriemi tím, že zakomponuje odpovídající uživatelské rozhraní společně s intuitivní uživatelskou zkušeností.

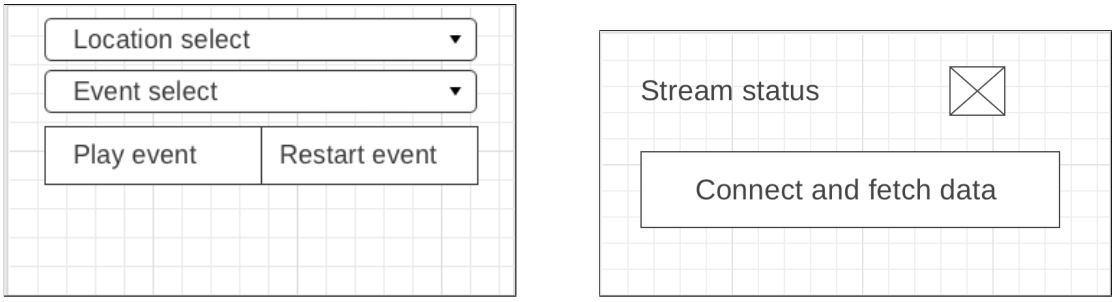
Aplikace nabídne v oblasti uživatelského rozhraní několik obrazovek, které zprostředkují uživateli aktuální dění na závodním okruhu. Zároveň zprostředkuje možnost sledovat detaily vybraného závodníka.



Obrázek 14: Wireframe zobrazující uspořádání hlavní obrazovky.

- **Hlavní obrazovka**

  Dominantním prvkem této obrazovky je zobrazení závodního okruhu ve 2D. Nad tímto zobrazením se nachází tlačítko, které bude zajišťovat přepnutí mezi režimy zobrazení 2D trati a trati v rozšířené realitě. Pod touto sekcí, kde je zobrazený aktuálně vybraný závodní okruh, je sekce, která bude zajišťovat přepínání mezi zdroji dat. Rozložení prvků této obrazovky je možné vidět na obrázku 14.



(a) Pro offline data. (b) Pro online data.

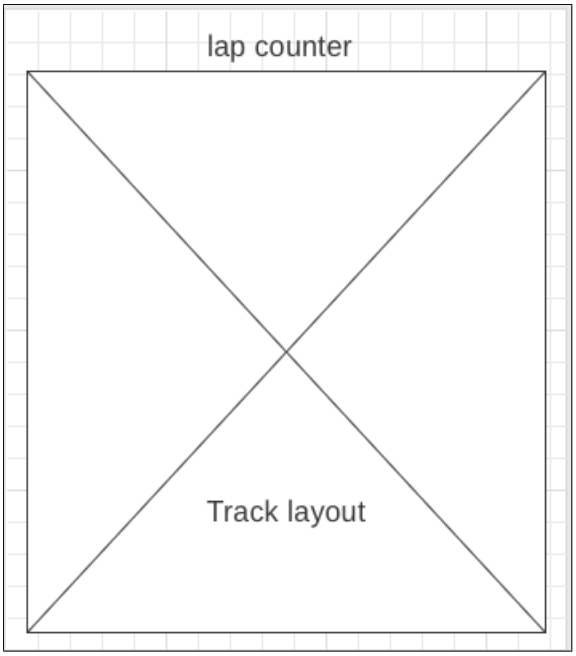
Obrázek 15: Wireframe zobrazující ovládací prvky, které budou zobrazeny po přepnutí zdroje dat.

V sekci s offline daty (obrázek 15a) bude na výběr z několika okruhů, u kterých byl pořízen záznam, kdy byly vozy na trati. Jako první vybere uživatel jeden ze závodních víkendů, například *FORMULA 1 GULF AIR BAHRAIN GRAND PRIX 2023* nebo *FORMULA 1 CRYPTO.COM MIAMI GRAND PRIX 2023*. Následně se zobrazí pole pro výběr konkrétní události z vybraného závodního víkendu. Může se jednat o jeden z volných tréninků, kvalifikaci, nebo hlavní závod. Výběrem jedné z těchto možností bude možné daný trénink, kvalifikaci či závod zpětně přehrát. Pro tyto ovládací akce jsou k dispozici ovládací tlačítka *Play event*, jenž umožní vybranou událost přehrát, a *Restart event*, které aktuální přehrávání zastaví a začne událost přehrávat od začátku.

Druhou sekcí, která bude sloužit pro ovládání aplikace, je sekce umožňující připojení k SignalR serveru Formule 1 (obrázek 15b). Ta bude zobrazena po změně výběru v ovládacím prvku *Offline data / Online feed* (obrázek 14). Tato sekce je složena z indikační ikony s popisem *Stream status*, která mění zvýraznění podle statusu připojení k serveru Formule 1. Pod touto indikací se nachází ovládací tlačítko *Connect and fetch data*, které spustí akci zajišťující připojení k serveru a spuštění příjmu zpráv. Výsledný stav připojení bude poté indikován již zmíněnou stavovou ikonou.

- **Zobrazení okruhu**

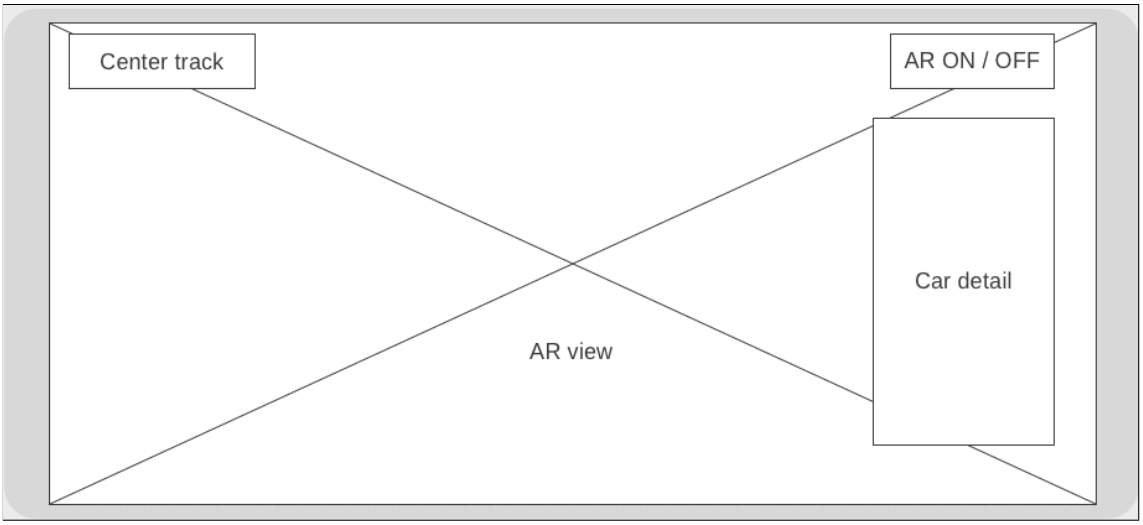
  Tato komponenta uživatelského rozhraní slouží primárně nejen k vyobrazení trati aktuálně vybraného okruhu, ale také pozic všech aktuálně jedoucích závodníků. Jednotlivá pozice závodníka na trati je unikátně rozlišena pomocí zobrazení závodníkovy zkratky, jenž je obarvena příslušnou barvou závodníkova týmu. Nad vizualizací trati je umístěn indikátor zobrazující číslo aktuálně probíhajícího kola závodu. Tento indikátor nahradí zástupný popis *lap counter*. Celé rozvržení této komponenty je zobrazeno na obrázku 16. Výše popsaná komponenta bude aktualizována se stejnou frekvencí, se kterou budou přijímána data ze zdroje.



Obrázek 16: Wireframe komponenty pro zobrazení trati vybraného okruhu.

- **Zobrazení rozšířené reality**

  Pro zobrazení rozšířené reality bude využitý režim zobrazení uživatelského rozhraní na šířku (aby byla zajištěna maximální plocha pro zobrazované předměty). Tato obrazovka bude nadále umožňovat přepnutí zpět do 2D zobrazení, které bude zajišťovat tlačítko v horní části obrazovky. Tlačítko bude umístěno ve stejné pozici jako na hlavní obrazovce (viz obrázek 14), aby byla zajištěna konzistence mezi obrazovkami a uživatelské rozhraní bylo jednotné. Dále tato obrazovka umožní zaznamenání požadované polohy kotvícího bodu pro zobrazení 3D trati, a to pomocí tlačítka umístěného v levé horní části obrazovky pojmenovaného zástupným textem *Center track*. Aktuální pozice kotvícího bodu bude také v prostředí rozšířené reality vizualizována. Celé prostředí popisované obrazovky je možné vidět na obrázku 17.



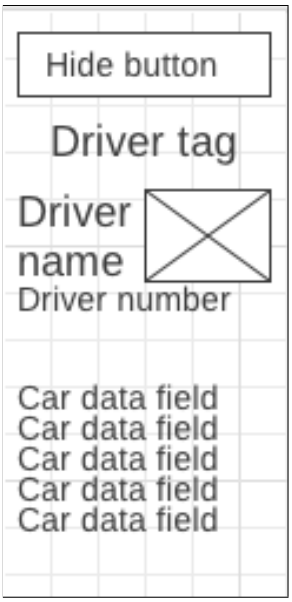
Obrázek 17: Wireframe zobrazující obrazovku, která bude sloužit pro zobrazení rozšířené reality.

- **Detail vozu**

  V případě, že bude vybrán konkrétní vůz, zobrazí se na této obrazovce formou takzvaného overlay (zobrazení komponenty na komponentě) detail vozu s aktuálními údaji z telemetrie. Tyto údaje budou obsahovat dostupné informace o voze (jako jsou například aktuálně zařazený rychlostní stupeň, otáčky motoru, rychlost vozu, pozice plynového pedálu, nebo pozice brzdového pedálu), společně s informacemi o závodníkovi (jako je závodníkova zkratka, celé jméno, závodní číslo a dostupná fotografie).

  Tento detail je popsán na obrázku 18. Samotný detail bude mít průhledné pozadí, aby nedocházelo k rušivému efektu při pozorování vozů Formule 1.

Všechny komponenty, které jsou navrženy tak, že pracují s proměnlivými daty, budou obnovovány s takovou frekvencí, kterou bude poskytovat zdroj dat. Výjimku tvoří prostředí rozšířené reality, kdy je frekvence vyobrazování objektů udávána buď zatížením zařízení nebo zařízením samotným. Zmíněnou frekvencí budou pouze obnovována data sloužící k zobrazení pozic virtuálních předmětů.



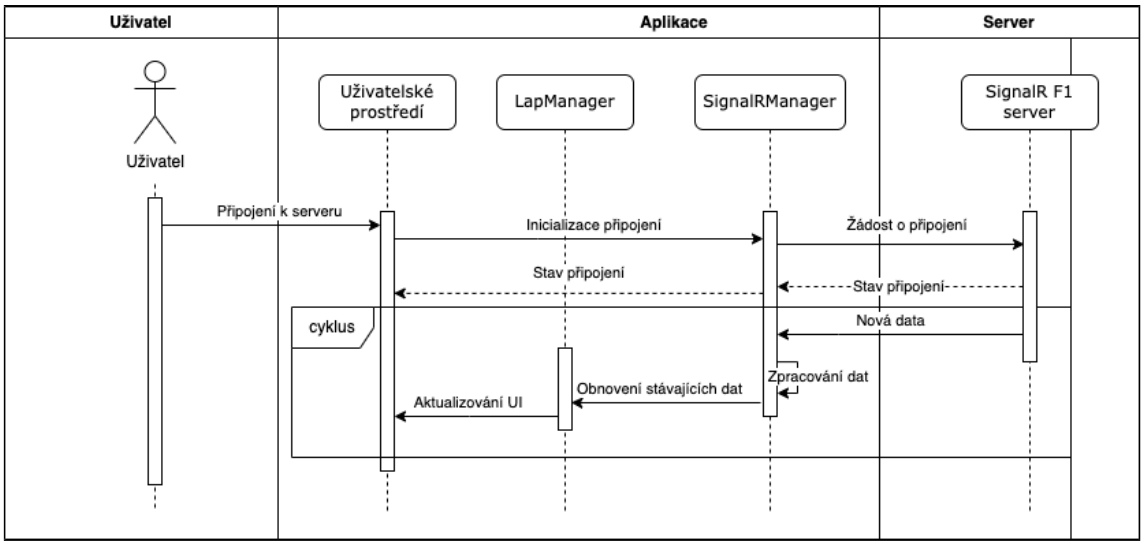
Obrázek 18: Wireframe zobrazující uspořádání detailu vozu v rozšířené realitě.

5.4 Architektura aplikace

Architektura aplikace je rozdělena do logických funkčních celků, které vykonávají své funkce. Pro vizualizaci těchto interakcí mezi uživatelem a celky aplikace spolu se serverem Formule 1, který slouží jako zdroj dat, jsou připraveny následující diagramy zobrazené na obrázcích 19 a 20.

- **Interakce uživatele s využitím reálných online dat**
  Tento diagram vizualizuje sekvenci akcí, které jsou vyvolány uživatelem a které jsou dále, již samostatně, vykonávány uvnitř aplikace.

  Uživatel iniciuje připojení k serveru stisknutím tlačítka v uživatelském prostředí, konkrétně tlačítka *Connect and fetch data* z obrázku 15b. Po stisknutí zmíněného tlačítka je volána akce třídy `SignalRManager`, která zajišťuje komunikaci se serverem, a ta odešle k serveru žádost o připojení. Jakmile třída `SignalRmanager` dostane odpověď na žádost o připojení, je výsledný stav propagován do uživatelského prostředí, kde je zobrazen uživateli. Zároveň třída `SignalRmanager` začíná přijímat data ze serveru. Tento příjem je opakován v cyklu do ukončení přenosu dat. Poté, co třída `SignalRmanager` přijme data, dojde k jejich prvotnímu zpracování a následnému předání dat třídě `LapManager`. `LapManager` aktualizuje svá data o pozicích vozů a data z telemetrie, která následně předá k dispozici uživatelskému rozhraní, jenž je aktualizuje pro zobrazení uživateli.

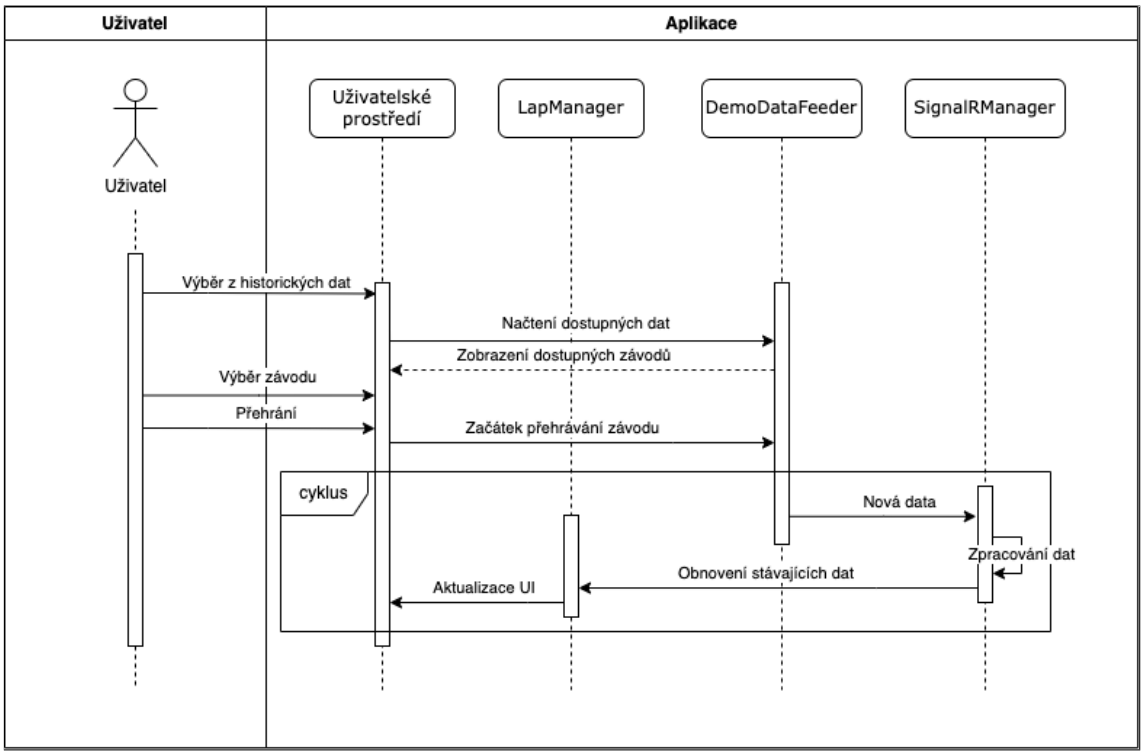


Obrázek 19: Sekvenční diagram s využitím online dat.

- **Interakce uživatele s využitím nahraných offline dat**
  Interakce uživatele s aplikací, kdy dojde k využití již dostupných zpracovaných offline dat, je vizualizována v diagramu na obrázku 20. Interakci stejně jako v případě online dat zahájí uživatel. Tentokrát ale požadavkem na výběr události z historických dat. Požadavek přijde do aplikace skrze uživatelské rozhraní, které zavolá třídu `DemoDataFeeder`, jež načte seznam dostupných Velkých cen a vrátí jej uživatelskému prostředí k zobrazení. Uživatel poté pokračuje konkrétním výběrem závodu a stiskem tlačítka *Play event* (v návrhu na obrázku 15a).

  Stisk tlačítka pro přehrání vyvolá ve třídě `DemoDataFeeder` opakovanou událost, kdy budou nová data — načtená z lokálního souboru, periodicky předávána třídě `SignalRmanager` ke zpracování. Následné zpracování dat třídou `SignalRmanager`, předání třídě `LapManager` a aktualizace dat v uživatelském rozhraní jsou identické jako v případě získávání dat online. Odlišnost je pouze ve zdroji, který je použit k zásobování třídy `SignalRmanager` daty.

Pro zjednodušení návrhu je část uživatelského rozhraní sjednocena jak pro 2D zobrazení, tak pro zobrazení v rozšířené realitě. Data, která jsou těmto zobrazení předávána, jsou identická a případná úprava je již v režii tříd, jež mají jednotlivá zobrazeními na starosti.



Obrázek 20: Sekvenční diagram s využitím offline dat.

6 Implementace návrhu

Část této kapitoly o implementaci návrhu představuje použité nástroje a metody, které byly při programování aplikace použity. Vývojovým prostředím byl zvolen XCode, který umožňuje vývoj nativních aplikací pro zařízení Apple. Vývojový jazyk byl zvolen Swift v kombinaci s knihovnou SwiftUI použitou pro návrh a kódování uživatelského prostředí.

6.1 Připojení ke zdroji dat

V předchozí kapitole, která popisovala návrh řešení, byly uvedeny různé metody získávání aktuálních i historických dat z prostředí Formule 1. Pro splnění požadavků na aplikaci je nutné vybrat takový zdroj dat, který zajistí konzistentní spojení a datový proud. Tyto požadavky splňuje použití datového streamu Formule 1, který využívá protokolu SignalR a který lze použít bez nutnosti registrace či dalšího ověření. Komunikaci s protokolem SignalR je možné navázat na adrese: `https://livetiming.formula1.com/signalr`.

Jelikož je platforma zařízení Apple zařazena mezi nepodporované, není dostupný oficiální framework pro příjem zpráv protokolem SignalR. To ale neznamená, že není možné vyvinout nativní aplikaci v jazyce Swift, která bude zprávy z tohoto protokolu přijímat. Existuje neoficiální knihovna SwiftSignalRClient, která zprostředkovává komunikaci s protokolem SignalR a také připojení k serveru, kde je protokol využíván. Další cestou, kterou by bylo možné vyvinout aplikaci pro zařízení Apple, je využít multiplatformního vývoje aplikací v jazyce ASP.Net Core a nástroje Xamarin. Tím by ale nebyly splněny funkční požadavky a vývoj samotné aplikace by byl složitější.

Během vývoje aplikace bylo zjištěno, že je zdrojový server naprogramován v jazyce ASP.NET, nikoliv ASP.NET Core, a používá starší implementaci SignalR protokolu. Nelze proto použít knihovnu, jež je referencovaná v dokumentaci protokolu SignalRCore (tj. SwiftSignalRClient), ale je nutné použít starší dostupnou knihovnu SignalRClient, která komunikaci se starší verzí protokolu podporuje.

Další podstatnou překážkou se při vývoji ukázala absence oficiální dokumentace metod protokolu SignalR na serverech Formule 1. Tato překážka byla odstraněna rešerší komunitních stránek (Reddit, StackOverflow) a dokumentace knihovny Fast F1, které se touto tematikou zabývají a kde byly nalezeny názvy metod pro připojení a obecné rady pro implementaci.

Výsledkem rešerše bylo zjištění názvu SignalR hubu *Streaming* a následujících hlavních metod pro příjem dat:

- `SPFeed`
- `Feed`
- `Subscribe`

K odběru těchto metod by se měla mobilní aplikace přihlásit – pomocí registrace `.on("název metody")` metod – a server by měl pomocí vzdáleného volání metod protokolu SignalR tyto metody vyvolávat. Vyvolání vzdálené metody by mělo v parametru zprávy obsahovat nová data, která se budou dále zpracovávat. Identický proces přijímání zpráv využívá `Live Timing Client`, který je součástí knihovny Fast F1 a jenž je možné využít k nahrávání streamovaných dat během akce na trati.

Zmíněný `Live Timing Client` je v této práci využit k nahrávání dat za účely offline zpětného přehrání. Pro nahrání a uložení záznamu je nutné nainstalovat knihovnu Fast F1 podle instrukcí v dokumentaci a následně pomocí příkazu v příkazovém řádku spustit nahrávání. Tento příkaz je zobrazen v ukázce kódu 8. Příkaz zajistí spuštění funkce `livetiming` knihovny Fast F1 s parametrem `save`, jenž ve své hodnotě obsahuje požadovaný název souboru, do kterého se mají data ukládat. V případě úspěšného spuštění nahrávání je uživateli v příkazovém řádku zobrazeno potvrzení.

```
1 python -m fastf1.livetiming save saved_data.txt
```

Ukázka kódu 8: Příkaz pro spuštění nahrávání události s využitím knihovny Fast F1 v příkazovém řádku.

Po ukončení přenosu dat dojde k detekci absence zpráv a nahrávání je automaticky ukončeno.

6.2 Zpracování dat

Hlavní třídou, která má na starosti komunikaci s frameworkem SignalRSwift, je třída `SignalRManager`. Jedná se o komplexní třídu, která zajišťuje komunikaci se serverem pomocí SignalR protokolu a následné prvotní zpracování přijatých dat.

V rámci inicializace instance této třídy dojde k přípravě připojení k serveru Formule 1. Kód na ukázce 9 obsahuje hlavní kroky potřebné k navázání komunikace se serverem. V první části kódu je definice konstant obsahujících parametry přidané do hlavičky požadavku o připojení. Lze si povšimnout hlavičky `"Connection": "keep-alive, Upgrade"`, která je specifická pro komunikaci s využitím protokolu SignalR. Po vzniku hlaviček následuje vytvoření referencí na připojení k hubu `HubConnection` s parametrem obsahujícím adresu serveru. Poslední vytvořenou proměnnou je `stream` typu `HubProxy`, která je inicializována později.

```
1     private let headers = ["User-agent": "BestHTTP",
2                            "Accept-Encoding": "gzip, identity",
3                            "Connection": "keep-alive, Upgrade"]
4     var hubConnection = HubConnection(
5                                withUrl: "https://livetiming.formula1.com/signalr")
6     var stream : HubProxy
7     public init(autostart: Bool = false) {
8         hubConnection.headers = self.headers
9
10        stream = hubConnection.createHubProxy(hubName: "Streaming")!
11        registerOnEvents()
12        hubConnection.started = {
13            print("Connected")
14            self.streamingStatus = .available
15        }
16
17        hubConnection.reconnecting = { print("Reconnecting...") }
18
19        hubConnection.reconnected = { print("Reconnected.") }
20
21        hubConnection.closed = {
22            print("Disconnected")
23            self.streamingStatus = .offline
24        }
25
26        hubConnection.connectionSlow = { print("Connection slow...") }
27
28        hubConnection.error = { error in print("Error") }
29
30        hubConnection.received = { data in print(data) }
31
32        if autostart {startConnection()}
33    }
```

Ukázka kódu 9: Inicializační metoda třídy `SignalRManager`.

Následuje funkce `init` třídy `SignalRManager`, kde dochází k nastavení hlaviček do `hubConnection` a inicializace proměnné `stream`. Tato proměnná je dále používána pro registraci `.on()` metod ve vnitřní funkci `registerOnEvents()` třídy `SignalRManager`, vyobrazené v kódu 10. V této funkci je nastaven příjem událostí

ze serveru pro metodu `feed`, která v parametru obsahuje přijatá data ze serveru, která jsou následně v těle metody zpracována.

```
func registerOnEvents(){
    let _ = stream.on(eventName: "feed") { (args) in
        let message = args[0]
        print("Message: ON Stream")
        // pokračující zpracování přijaté zprávy...
}
```

Ukázka kódu 10: Funkce, která registruje odběr událostí hubu `Streaming`.

Poslední část inicializační funkce obsahuje definování akcí, jež se mají provést v případě změny připojení. Tyto funkce jsou definované ve frameworku SignalRSwift a díky nim lze v aplikaci reagovat na změnu stavu připojení. Jedná se o následující funkce instance třídy `HubConnection`:

- `started` – vyvoláno v případě, že dojde k úspěšnému navázání spojení se serverem.
- `reconnecting` – stav značící opětovný pokus o připojení.
- `reconnected` – stav, kdy bylo spojení úspěšně opětovně připojeno.
- `closed` – označení pro ukončení spojení.
- `connectionSlow` – stav značící připojení k serveru pomalou rychlostí.
- `error` – chybový stav připojení.
- `received` – vyvoláno v případě obdržení zprávy ze serveru.

Jak je možné v kódu 9 vidět, tak většina těchto funkcí ve svém těle definuje pouze výpis značící stav připojení pro programátora. Výjimkou jsou případy `hubConnection.started` a `hubConnection.closed`, jež mění aktuální stav připojení uložený v interní proměnné, která je používána dále v aplikaci pro vizualizaci stavu připojení uživateli.

Posledním krokem inicializace je možnost automatického spuštění navázání spojení. Funkce `startConnection()` obsahuje pouze příkaz `.start()` pro instanci `hubConnection`.

Během testování připojení bylo zjištěno, že nedochází ke korektnímu vyvolávání `.on()` metod serverem. Vzhledem k tomuto poměrně signifikantnímu zjištění bylo nutné najít vhodnou alternativu k získávání dat ze serveru prostřednictvím protokolu SignalR. Je nutné zmínit, že daná chyba, jež se projevuje nevoláním metod aplikace ze serveru, může být zapříčiněna vnitřní chybou použitého frameworku. Jak již bylo zmíněno, vývojová platforma pro nativní vývoj aplikací pro zařízení Apple není oficiálně podporována.

Tato překážka byla odstraněna poměrně snadným způsobem. Bylo zjištěno, že po vyvolání metody `Subscribe` s parametry obsahujícími požadované názvy dat na serveru, vrátí server jako odpověď slovník[21] naplněný požadovanými daty. Toho bylo dále pro potřeby vyvíjené aplikace využito.

Aplikace obsahuje třídu `DemoDataFeeder` obsahující funkce, které se starají o periodické vyvolání serverové metody `Subscribe`. Tyto funkce jsou uvedeny v kódu 11. `startFeeding(atRateOf: Double)` je funkce, jež přijímá parametr datového typu `Double` značící interval, s kterým budou funkce volány. V této funkci je inicializována proměnná třídy `DemoDataFeeder`, `feedTimer`. Časovač `feedTimer` opětovně spouští funkci `feedOnlineData()`, která volá privátní funkci `getFreshCarPositionAndData()` třídy `SignalRManagerV2`. Tímto opakujícím se procesem je zajištěno periodické načítání dat ze serveru. Frekvenci načítání dat lze v kódu změnit dle požadavků. Z testování aplikace byla vysledována ideální frekvence 0,5 sekundy.

```
1  var feedTimer: Timer?
2  func startFeeding(atRateOf: Double){
3      feedTimer = Timer.scheduledTimer(
4                      withTimeInterval: atRateOf,
5                      repeats: true,
6                      block: { [self] timer in
7          feedOnlineData()
8      })
9  }
10
11 private func feedOnlineData(){
12     SignalRManagerV2.shared.getFreshCarPositionAndData()
13 }
```

Ukázka kódu 11: Funkce zajišťující periodické načítání dat ze serveru.

[21] Datová struktura, kde jsou data ve formátu [klíč: hodnota].

```
func startFeedingDemoData(forTrack: String, event: String){
    if let eventData = (demoDataTracks[forTrack] as! Dictionary<String, AnyObject>)
    [event] as? [Dictionary<String, AnyObject>]{
        self.demoDataSource = eventData.filter { item in
            if item.keys.contains(where: { keyString in
                (keyString == "Position.z") ||
                (keyString == "CarData.z") || (keyString == "LapCount")
            }){
                return true
            }else{return false}
        }
        let delay = 0.1
        print("Local Feeder started with delay: \(delay)")
        feedTimer = Timer.scheduledTimer(
                        withTimeInterval: delay,
                        repeats: true,
                        block: { [self] timer in //1s
            if lastFedIndex < demoDataSource.count{
                feedLocalData()
                lastFedIndex += 1
            }else{
                stopFeeding()
                lastFedIndex = 0
            }
        })
    }
}
func feedLocalData(){
    SignalRManagerV2.shared.parsePosition(from: demoDataSource[lastFedIndex])
}
```

Ukázka kódu 12: Funkce zajišťující přehrání offline dat.

Pro zajištění možnosti použití již nahraných dat disponuje třída `DemoDataFeeder` funkcí `startFeedingDemoData` (zobrazené na ukázce kódu 12). Tato funkce přijímá dva vstupní parametry datového typu `String`. První, `forTrack`, reprezentuje název závodního víkendu a druhý, `event`, reprezentuje požadovanou událost, která se v rámci závodního víkendu odehrála. Funkce provede načtení požadovaných dat z globální proměnné `demoDataTracks`, zobrazené v ukázce kódu 13. Tato data poté projdou filtrem zpráv a jsou vybrána pouze ta, která jsou stěžejní pro přehrání události v aplikaci. V tomto případě se jedná o následující:

- `Position.z`
- `CarData.z`
- `LapCount`

Zprávy těchto druhů jsou poté umístěny do pole, jenž je použito v těle časovače. Tento časovač následně s frekvencí 0,1 sekundy předává prvky pole funkci `parsePosition(from: _)` třídy `SignalRManagerV2`, která je zpracuje stejným způsobem jako zprávy přijaté ze serveru.

```
var demoDataTracks: OrderedDictionary = ["Singapore 2022": ["Race": demoData],
"Saudi Arabia 2023": ["Qualification": demoData2, "Race": []]]
as OrderedDictionary<String, Any>

var demoData: [Dictionary<String, AnyObject>] = load("Singapore_21.json")
var demoData2: [Dictionary<String, AnyObject>] = load("saudiquali.json")
```

Ukázka kódu 13: Způsob uložení a načtení offline dat pro následné přehrání aplikací.

Data přijata třídou `SignalRManagerV2` musí být zpracována, aby mohla být dále použita v aplikaci. Jakmile je obdržena nová zpráva, dochází k rozlišení druhu zprávy a příslušnému zpracování. Pro možnost použití určitých dat (především `CarData.z` a `CarPosition.z`) je nutné provést kroky popsané v následujících bodech:

- **Dekomprese**
  Název dat s příponou typu *.z* logicky implikuje, že jsou daná data komprimována. Jediným faktorem, který práci s dekompresí dat komplikuje, je neexistující dokumentace. Při rešerši zpracování dat bylo zjištěno, že tento formát zpráv používá kompresní algoritmus Deflate a Inflate. Aby bylo možné tato komprimovaná data dále použít, byl využit framework DataCompression, jenž usnadňuje práci s komprimovanými daty. Je tedy možné na komprimovaná data využít funkci `inflate()` pro získání dat dekomprimovaných. Příklad je na ukázce kódu 14, kde jsou data načtena z textového formátu a poté pomocí `inflate()` dekomprimována do výsledné podoby.

```
    guard let carPositionZData = Data(base64Encoded:
                                    (carPositionZ as! String)) else {return}
    if let decompressed = carPositionZData.inflate(){ pokračování zpracování...}
```

Ukázka kódu 14: Příklad dekomprese dat pomocí funkce `inflate()`.

- **Objektové mapování**

  Jakmile jsou potřebná data dekomprimována, je stále nutný jejich převod do objektové podoby. Tento proces lze v obecném měřítku popsat následovně. Vytvoří se objektová předloha – objekt, který obsahuje atributy s odpovídajícími datovými typy – a ta je následně naplněna reálnými daty. Anglicky se tento proces nazývá *object mapping* a do češtiny je doslovně přeložen jako objektové mapování. Vyvíjená aplikace pracuje s objektovým mapováním podle následujících modelových předloh. U všech modelových předloh byla použita struktura namísto třídy. Struktura nabízí menší paměťovou náročnost a má omezenou dědičnost, což je v tomto případě vítané.

  - `Car`

    `Car` je struktura obsahující informace o závodním voze, konkrétně jeho číslo, které je totožné s číslem závodníka, a atribut `channels` datového typu `CarChannels`, jenž je v konstruktoru vytvořen pomocí typového slovníku `Dictionary<String, Any>`. Tato struktura je vyobrazena na ukázce kódu 15. Výsledný objekt je vytvořen z přijatých dat `CarData.z`, které nejdříve projdou dekompresí. `CarData.z` obsahují také data mapována z předpisu `CarChannels`. Z toho důvodu na sebe oba datové předpisy navazují.

```
1   struct Car{
2       let racingNumber: Int
3       let channels: CarChannels
4
5       init(racingNumber: Int, channelsDict: Dictionary<String, Any>) {
6           self.racingNumber = racingNumber
7           channels = CarChannels(
8                           fromDict: channelsDict["Channels"]
9                           as! Dictionary<String, Any>)
10      }
11  }
```

Ukázka kódu 15: Datová struktura `Car` použita k objektovému mapování.

  - `CarChannels`

    Tato struktura, jak již bylo zmíněno výše, je inicializována z typového slovníku `Dictionary<String, Any>`. Atributy této struktury jsou naplněny ze získaných hodnot pomocí odpovídajících klíčů slovníku. Dané klíče jsou definovány již v přenesených datech a konkrétní popis byl získán z dokumentace knihovny Fast F1. Tato struktura je vyobrazena v ukázce kódu 16.

```
struct CarChannels{
    let rpm, speed, gear, throttle, c45, brake: Int

    init(fromDict: Dictionary<String, Any>) {
        rpm = fromDict["0"] as! Int //RPM
        speed = fromDict["2"] as! Int //Speed
        gear = fromDict["3"] as! Int //Gear
        throttle = fromDict["4"] as! Int //Throttle %
        c45 = fromDict["45"] as! Int
        brake = fromDict["5"] as! Int //Brake 0/1
    }
}
```

Ukázka kódu 16: Datová struktura `CarChannels` použita k objektovému mapování.

– **`CarPosition`**

`CarPosition` je datovou strukturou, která je z celé aplikace nejdůležitější. Obsahuje totiž souřadnice x, y, z, jež reprezentují pozici vozu na trati. Dalšími atributy jsou `carNumber` – číslo závodního vozu, `status` – textový popis, zda je vůz na trati, nebo mimo trať („OnTrack“, „OffTrack“), `timestamp` – datum, kdy byla konkrétní pozice zaznamenána, a `id` – interní unikátní identifikátor přidaný z důvodu použití struktury pro zobrazení uživatelského rozhraní. Tato struktura je vyobrazena v ukázce kódu 17. `CarPosition` je druhý předpis, jenž je vytvářen z dat, které je nutné nejdříve dekomprimovat. Původní název dat je `Position.z`.

```
struct CarPosition{
    let id = UUID()
    let timestamp: Date
    let x, y, z: Int
    let status: String
    let carNumber: Int

    init(carNumber: Int, timestamp: Date, fromDict: Dictionary<String, Any>) {
        self.timestamp = timestamp
        x = fromDict["X"] as! Int
        y = fromDict["Y"] as! Int
        z = fromDict["Z"] as! Int
        status = fromDict["Status"] as! String
        self.carNumber = carNumber
    }
}
```

Ukázka kódu 17: Datová struktura `CarPosition` použita k objektovému mapování.

– **DriverInfo**

Objektový předpis `DriverInfo` reprezentuje data o konkrétním závodníkovi. Tento seznam byl získán v rámci vývoje a testování aplikace je načítán ze statického umístění v aplikaci. Atributy jsou použity pro zobrazení závodníkova detailu při vizualizaci na okruhu. Tento objektový předpis je vyobrazen v ukázce kódu 18.

```
struct DriverInfo{
    let broadcastName, countryCode, firstName, fullName, headshotURL,
    lastName, reference, teamColour, teamName, tla: String
    let racingNumber, line: Int
}
```

Ukázka kódu 18: Datová struktura `DriverInfo` použita k objektovému mapování.

– **LapCount**

`LapCount`, jak již název napovídá, reprezentuje typ zprávy, jež obsahuje informace o aktuálním závodním kole. Tato struktura obsahuje pouze dva atributy, a to `currentLap` datového typu `Int`, který je naplněn pořadovou číslicí kola, a `date` reprezentující začátek daného kola. Tato struktura je vyobrazena v ukázce kódu 19.

```
struct LapCount {
    let currentLap: Int
    let date: Date
}
```

Ukázka kódu 19: Datová struktura `LapCount` použita k objektovému mapování.

6.3 Vykreslení závodního okruhu ve 2D

Aby bylo možné zobrazit uživateli podobu závodního okruhu, je nutné přijatá a namapovaná data předat zobrazovací vrstvě. Všechna data, která byla přijata, spojuje dohromady třída `LapManager`. Tato třída přijímá data od třídy `SignalRManager`, jež má na starosti jejich příjem a prvotní zpracování.

Přijatá data poskytují pouze aktuální polohu vozů na trati, jak bylo popsáno ve struktuře `CarPosition` v ukázce kódu 17. Aplikaci jsou tedy dostupné pouze souřadnice x, y, z, které nemají předem daný rozsah. Je tedy poměrně obtížné zobrazit odpovídající okruh pouze na základě těchto dat.

Během vývoje bylo vysledováno, že zmíněné souřadnice x, y, z jsou v rozmezí přibližně $[-20000, 20000]$. Tento údaj dává alespoň předběžnou představu o jejich rozmezí. Bohužel je tato hranice proměnlivá podle daného okruhu, kde se aktuální závod Formule 1 pořádá. Je tedy nutné zvolit odpovídající komponentu, která dokáže zobrazit dynamická data, prozatím v osách x a y. Odpověď na otázku hledané komponenty je ve výsledku poměrně jednoduchá.

Funkcionalita zobrazování dat v grafu je ideální pro dynamická data, u nichž předem nevíme, v jakém jsou rozsahu. Díky dynamickým osám lze vykreslit jakýkoliv tvar závodního okruhu.

```
extension LapManager: CarPositionsProtocol{
    ...
    func receivedNewData(carPositions: [[CarPosition]]) {
        if !carPositions.isEmpty {
            switch lapMappingStatus{
            case .idle:
                let positions = carPositions.first!
                let trackCar = positions.filter { pos in
                    pos.status == "OnTrack"
                }.randomElement()!
                mappingCar = trackCar
                mappingCarPositions.append(trackCar)
                updateLapModel()
                lapMappingStatus = .mapping
                driverInfo = SignalRManagerV2.shared.getDriverList(from:
                                                    driverList)
            case .mapping:
                if let targetCar = carPositions.first!.first(where: { car in
                    car.carNumber == mappingCar!.carNumber
                }){
                    mappingCarPositions.append(targetCar)
                    updateLapModel()
                }
            case .mapped:
                allCarsPositions = carPositions
                updateLapModel()
            case .error:
                print("Error mapping")
            }
        }
    }
    ...
}
```

Ukázka kódu 20: Rozšíření třídy `LapManager`, implementující protokol `CarPositionsProtocol` pro příjem dat.

Poté, co je vybrána vyhovující zobrazovací komponenta, je nutné zjistit, jaké budou další logické postupy pro zobrazení. Při každé aktualizaci dat přijme aplikace informace o polohách všech závodníků spolu s informací, zda je daný závodník na trati, nebo mimo trať. Aby bylo zajištěno, že vždy bude vykreslen odpovídající okruh, je zaveden proces mapování okruhu. Ukázka kódu 20 zobrazuje implementaci protokolu třídy `SignalRManager`, který pomocí definovaných funkcí předává data pro následné zpracování. Funkce `receivedNewData(carPositions: [[CarPosition]])` zajišťuje zmíněný proces mapování. `LapManager` disponuje atributem, který indikuje proces mapování, jedná se o `enum`[22] `LapStatus` s následujícími stavy:

- `idle` – výchozí stav přiřazený po spuštění aplikace, kdy ještě nebylo spuštěno mapování okruhu.
- `mapped` – stav, kdy je okruh zmapovaný a připravený pro následující použití.
- `mapping` – stav indikující probíhající proces mapování okruhu.
- `error` – chybový stav.

Ve zmíněné funkci `receivedNewData(carPositions: [[CarPosition]])` je poté na základě těchto stavů rozlišováno, jaké akce budou spuštěny po příjmu nových dat.

Pokud ještě nezapočal proces mapování okruhu, je stav `.idle` proměnné `lapMappingStatus`. V tomto případě dojde k náhodnému výběru vozu, který má stav „OnTrack“. Tento vůz bude sloužit jako referenční vůz `mappingCar`, jehož pozice bude zaznamenávána pro účely získání podoby trati. Tato podoba je ukládána do pole všech zaznamenaných pozic sledovaného vozu `mappingCarPositions`. Posledním krokem v tomto stavu je inicializace detailů závodníků.

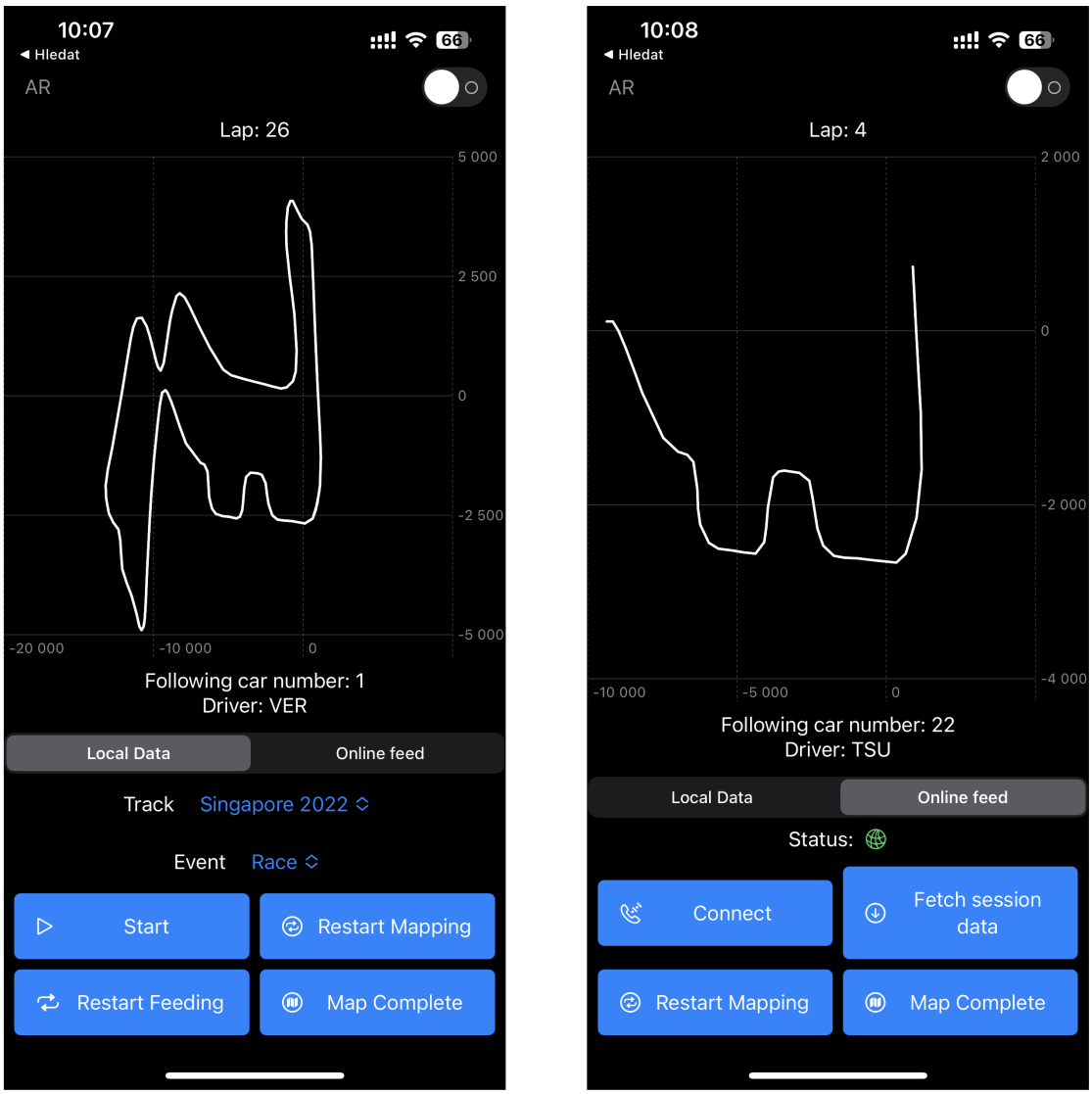
Po první fázi `.idle` přechází mapování okruhu do fáze `.mapping`. Každá nově přijatá zpráva stále obsahuje údaje všech závodních vozů. Je tedy z tohoto seznamu vybrán ten, který byl určen jako `mappingCar`, a jeho pozice je přidána do vznikajícího pole pozic závodního okruhu.

Poslední fáze mapování okruhu je stav `.mapped`, kdy dochází pouze k předání všech pozic vozů. Tento stav už nepracuje s `mappingCar`, protože byly zaznamenány všechny pozice, které dohromady tvoří závodní okruh.

[22] Návrhový vzor výčtu, kde je datový typ tvořen konečnou omezenou množinou pojmenovaných hodnot.

Ve všech stavech, kterými projde proces mapování závodního okruhu, se nachází volání funkce `updateLapModel()`. Tato funkce zajišťuje aktualizaci dat pro uživatelské rozhraní, a tím je uživateli umožněno sledovat postup mapování okruhu.

Na skupině obrázků 21 je možné vidět podobu výsledných obrazovek uživatelského prostředí.



(a) Podoba závodního okruhu a ovládací tlačítka pro offline data.

(b) Podoba závodního okruhu a ovládací tlačítka pro interakci se serverem.

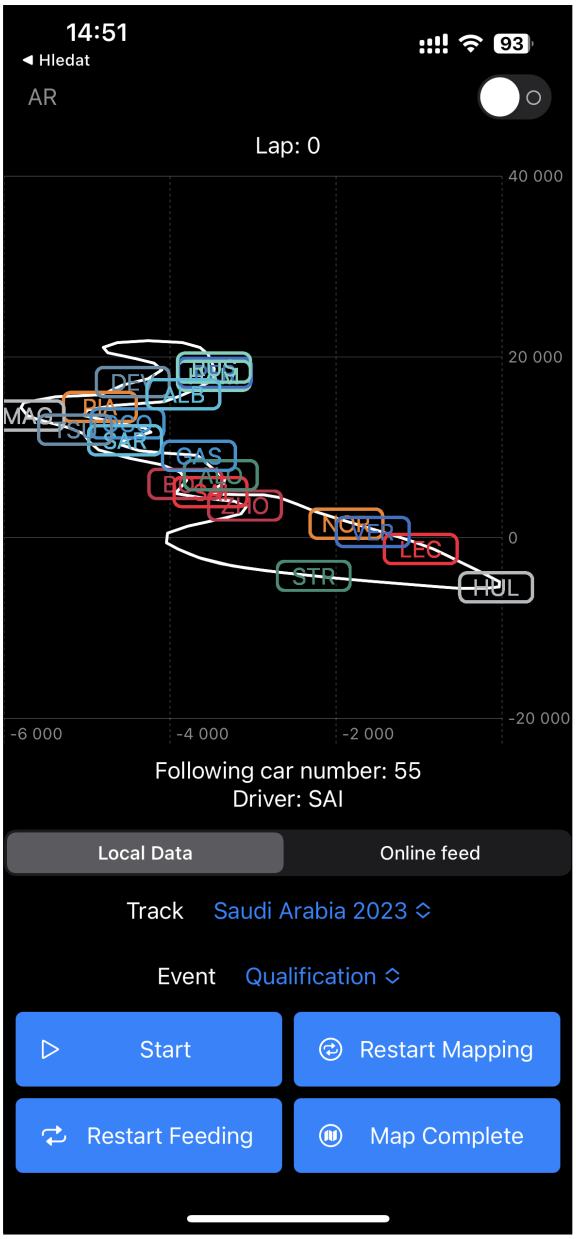
Obrázek 21: Podoba hlavních obrazovek pro zobrazení 2D závodního okruhu.

Obrázek 21a znázorňuje aktuálně dokončený proces mapování závodního okruhu. Uživatel nejdříve vybral požadovaný závodní víkend a k němu podřazenou událost. Uživatelské rozhraní nabídlo možnosti z lokálních demo dat. Dále stiskl tlačítko *Start*, které zajistí spuštění časovače ve třídě `DemoDataFeeder` pře-

dávající data třídě *SignalRManager*, jež je následně předá třídě `LapManager` spravující proces mapování okruhu. Jakmile jsou data zpracována, jsou předána skrze `LapViewModel` uživatelskému rozhraní, jenž vykreslí pozice vozu v podobě liniového grafu. Uživatel má dále možnost toto přehrávání spustit od začátku tlačítkem *Restart Feeding* nebo, pokud se vyskytne chyba, stiskem tlačítka *Restart Mapping* spustit proces mapování okruhu odznovu. Poslední tlačítko umístěné ve spodní části obrazovky slouží k označení stavu, kdy dojde k návratu sledovaného závodního vozu na výchozí pozici. Během vývoje aplikace nebyl bohužel nalezen optimální způsob detekce dokončeného kola určitého závodníka. Dostupná data neposkytují informace, které by bylo možné k sobě logicky přiřadit, a tím získat vazbu například mezi začátkem kola a daným závodníkem. Také nebylo dosaženo detekce dokončení okruhu při sledování pozic vozu a jeho návratu na začátek sledování.

Na obrázku 21b je k vidění uživatelské prostředí při výběru načítání dat z online zdroje. Zobrazení 2D závodního kola formou grafu je identické se zobrazením při použití offline dat. Horní část obrazovky vypisuje aktuálně sledované kolo závodu a pod prostorem grafu je vizualizováno, jaký závodník je aktuálně sledován spolu s jeho číslem vozu. Prostor s ovládacími tlačítky je přizpůsoben konečné funkcionalitě oproti původnímu návrhu, kde je tlačítko *Connect and fetch data* (dovolující uskutečnit připojení k serveru a zahájit příjem dat) rozděleno podle odpovídající funkcionality na tlačítka *Connect* a *Fetch session data*. Toto rozdělení je z důvodu nutnosti použít umělé vyvolávání příjmu dat ze serveru. Jakmile je spojení navázáno, změní indikační ikona připojení s popisem *Status:* barvu popředí na zelenou. Ve výchozím případě je ikona obarvena barvou červenou. Poslední řada tlačítek nabízí identickou funkcionalitu jako u offline dat, a to restartování mapování trati tlačítkem *Restart Mapping* a zaznamenání dokončené trati tlačítkem *Map Complete*.

Možnost vizualizovat pozice závodníků na zmapované trati ve 2D je na obrázku 22. Každý závodník je reprezentován zkratkou *tla*, která je obarvena v barvě závodníkova týmu. Obě informace jsou získané z objektu `DriverInfo`. Při vizualizaci je stále použita komponenta grafu (knihovna Charts vývojáře Apple). Frekvence, se kterou jsou data v grafu aktualizována, v kombinaci s množstvím údajů způsobuje trhané zobrazení. Z tohoto důvodu je ve výsledné aplikaci tato funkcionalita nezveřejněna.



Obrázek 22: Zobrazení závodního okruhu s vizualizací pozic závodníků pomocí značek.

6.4 Převod závodního okruhu do 3D

Aby bylo možné zobrazit závodní okruh v rozšířené realitě, je zapotřebí několika kroků. Jak je popsáno v sekci 3.4, jež pojednává o rozšířené realitě na zařízeních Apple, osy 3D prostoru nekorespondují s osami 2D prostoru. Je tudíž nutné určité osy zaměnit, aby bylo dosaženo požadovaného výsledku. Proto byla učiněna následující obměna:

$$x \longrightarrow y, \quad y \longrightarrow z, \quad z \longrightarrow x$$

Dalším krokem je převod rozsahu souřadnic. Ve 2D zobrazení jsou souřadnice v rozmezí přibližně $[-20000, 20000]$, a jelikož rozšířená realita používá jako jednotky metry, je nutná normalizace souřadnic na požadovaný interval. Ukázka kódu 21 obsahuje funkci, která zajistí převod souřadnic na rozmezí $[-0,5, 0,5]$. Specifická je ve 3D prostoru osa y značící v případě závodního okruhu převýšení na trati. V této ose dochází k normalizaci na interval $[0, 0, 05]$ z důvodu omezení chyb při vykreslení převýšení.

```
func normalize(min: Self, max: Self,
            from a: Self = -0.5, to b: Self = 0.5) -> Self {
    (b - a) * ((self - min) / (max - min)) + a
}
```

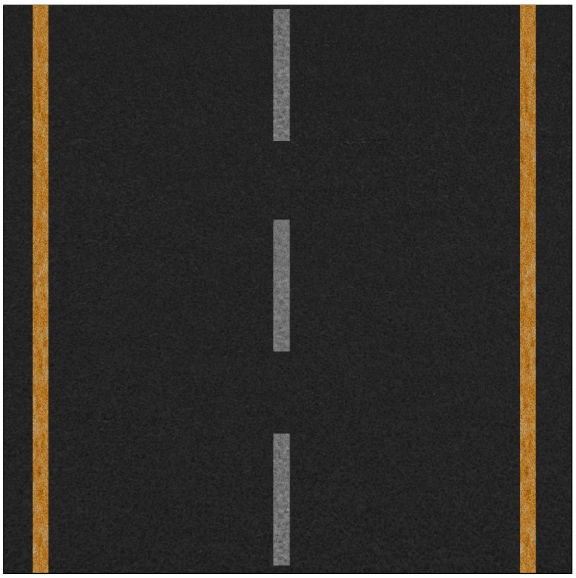
Ukázka kódu 21: Normalizace souřadnic na požadovaný interval.

```
func transformCarPositionsToVectors() -> [SCNVector3]{
    let carPositions = LapManager.shared.lapModel.carPos
    var resVectorArray = [SCNVector3]()
    var index = 0
    for carPos in carPositions{
        if index % 2 == 0{
            let normalizedX = Float(carPos.x).normalize(
                                min: Float(minX), max: Float(maxX))
            let normalizedY = Float(carPos.y).normalize(
                                min: Float(minY), max: Float(maxY))
            let zLow = Float(carPos.z).normalize(
                                min: Float(minZ), max: Float(maxZ),
                                from: 0, to: 0.05)
            resVectorArray.append(SCNVector3(
                                    x: normalizedY,
                                    y: zLow,
                                    z: normalizedX))
        }
        index += 1
    }
    return resVectorArray
}
```

Ukázka kódu 22: Vytvoření souřadnic typu `SCNVector3` pro reprezentaci bodů trati.

3D prostor umožňuje zobrazovat libovolné virtuální objekty. Aby mohla být vizualizována trať, je nutné najít její vhodnou reprezentaci v prostoru. Za tímto účelem byla vybrána knihovna SCNPath, jež umožňuje vizualizovat spojitou čáru s určitou texturou. SCNPath přijme pole vektorů `SCNVector3` vytvořené funkcí `transformCarPositionsToVectors()`, zobrazené v ukázce kódu 22, a mezi kaž-

dým bodem vykreslí část trati, jež má požadovanou texturu. Jako textura byl zvolen obrázek vizualizující vozovku – viz obrázek 23.



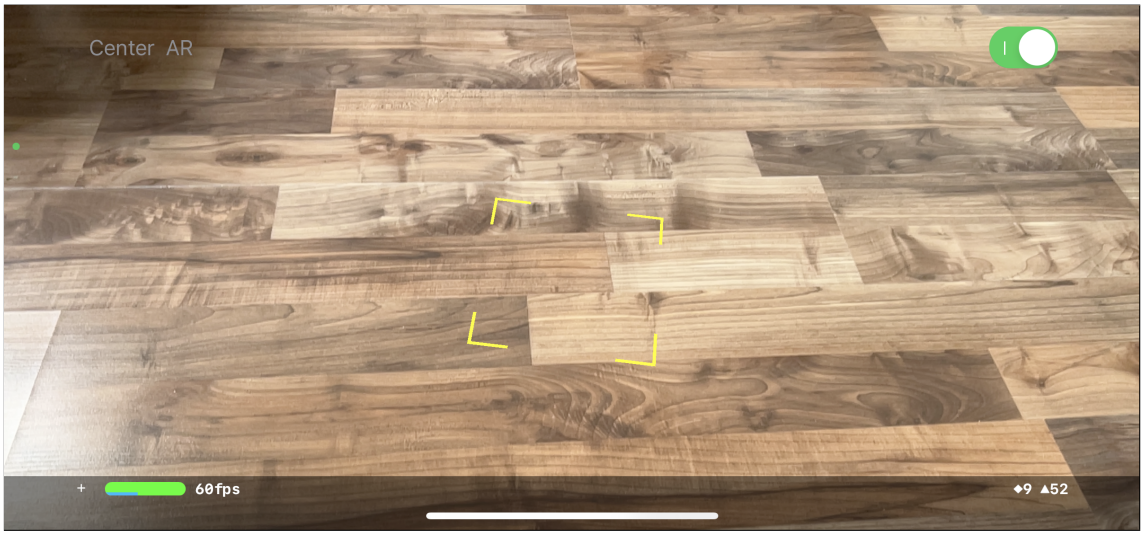
Obrázek 23: Textura použitá pro vizualizaci trati [39].

Výsledek zobrazování trati, kde byly souřadnice normalizovány a pro vykreslení byla použita textura, je vidět na obrázku 24. Během testování byl rozpoznán limit knihovny SCNPath, jež je zodpovědná za vykreslení tratě v počtu bodů, které může trať obsahovat. Tímto limitním počtem je 500+ bodů. Při překročení této hranice může docházet k neidentifikované chybě, jenž způsobí nevykreslení trati ve scéně. Aby bylo zajištěno vykreslení trati, je pro její reprezentaci v rozšířené realitě použit každý druhý prvek zdrojových pozic. Tento proces je možné vidět v ukázce kódu 22.

Obrázek 24: Zobrazená trať s texturou v rozšířené realitě.

6.5 Detekce plochy pro umístění objektů

Rozšířenou realitu na zařízeních Apple je možné vytvořit několika způsoby. Pro účely této vyvíjené aplikace byl zvolen způsob využívající ARKit a SceneKit. Jejich výsledná kombinace přináší možnosti rozšířené reality a vývoje 3D obsahu. V aplikaci je využito `ARSCNView`, jenž lze nastavit tak, aby detekovalo plochy, na které lze umístit kotevní body. Zároveň byla využita knihovna FocusNode, jež do scény přidává interaktivní indikaci nalezených ploch. Uživatel má tak přehled o nalezené ploše pro umístění středu tratě. Proces hledání plochy je možné vidět na obrázcích 25 a 26.



Obrázek 25: Začátek hledání plochy pomocí knihovny FocusNode.

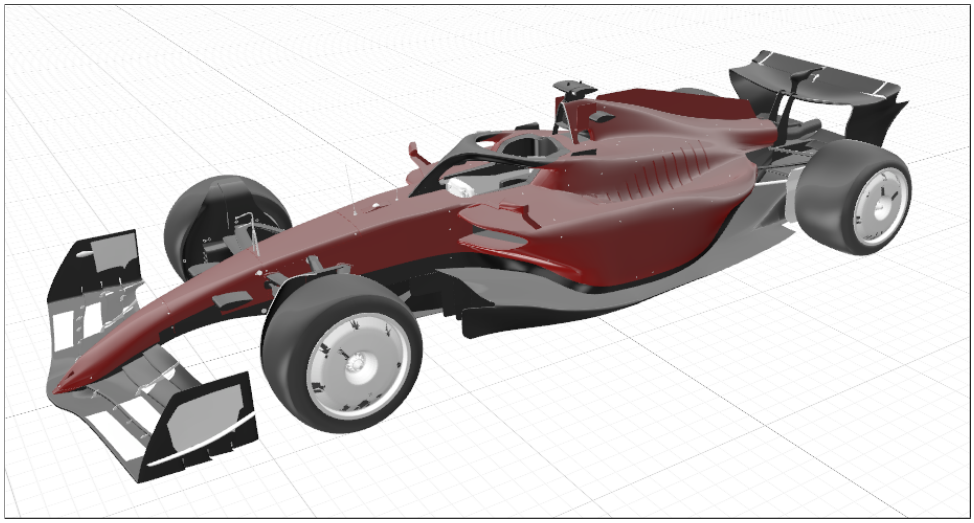
Obrázek 26: Detekována plocha s využitím knihovny FocusNode, čeká se na interakce uživatele.

Poté, co dojde k úspěšné detekci plochy, má uživatel možnost pomocí tlačítka *Center* v levém horním rohu označit aktuální pozici žlutého čtverce jako střed virtuálního závodního okruhu.

6.6 Umístění vozů na trať

Uživatel označil požadovaný bod pro umístění středu závodního okruhu a trať byla přidána do scény. Následuje přidání modelu vozů Formule 1.

Jako model reprezentující vozy Formule 1 byl vybrán volně dostupný 3D model uživatele D3DP – Samuel Mazuy dostupný na webových stránkách GrabCad. Tento model je zobrazený na obrázku 27.



Obrázek 27: Použitý model Formule 1, 2022 Scuderia Ferrari [40].

Pro použití daného modelu v aplikaci byl vytvořen USDZ soubor obsahující zdrojový model. Tento soubor je součástí zkompilované aplikace. V ukázce kódu 23 jsou postupně vidět kroky potřebné k načtení a následnému vložení do scény. Nejdříve je vytvořena URL adresa daného souboru, kterou poté ve funkci `viewWillAppear(_ animated: Bool)` třídy `ViewController` (jež má na starosti zobrazení scény) použije `MDLAsset` k načtení modelu do paměti. Dále jsou modelu načteny textury pomocí funkce `loadTextures()`. Následuje cyklus, který pro každého závodníka připraví závodní vůz. Načtený model je poté zmenšen na rozměry úměrné velikosti trati. Posledním krokem pro přidání modelu vozu do scény je přidání vozu jako potomka modelu trati. Tímto krokem je zajištěno, že závodní vůz má stejnou výchozí pozici jako je střed trati. Pokud by tento krok nebyl proveden tímto způsobem a vůz by byl umístěn jako potomek scény, byl by umístěn do počátku, který je v pozici kamery mobilního zařízení.

```
let f1CarUsdz = Bundle.main.url(forResource: "F1_2022_1", withExtension: "usdz")!
override func viewWillAppear(_ animated: Bool) {
    ...
    let asset = MDLAsset(url: f1CarUsdz)
    asset.loadTextures()
    ...
    for driver in LapManager.shared.driverInfo{
        ...
        let driverCar = SCNNode(mdlObject: asset.object(at: 0))
        driverCar.scale = SCNVector3(x: 0.01, y: 0.01, z: 0.01)
        ...
    }
    trackPathNode.addChildNode(driverCar)
    ...
}
```

Ukázka kódu 23: Hlavní kroky potřebné k načtení modelu Formule 1.

6.7 Úpravy modelů pro rozlišení závodníků

Tyto úpravy probíhají ve stejné části třídy `ViewController` jako načítání modelu vozu popsané v předchozí podkapitole. Kroky, zobrazené v ukázce kódu 24, jsou provedeny pro všechny závodníky individuálně. Každému vozu je přiřazen název, jenž je později použit pro identifikaci vozu na trati. Dále je vytvořen materiál `SCNMaterial`, který definuje barvu závodníkova týmu. Tato barva je poté přiřazena částem modelu vozu, které pokrývají největší plochu – konkrétně to jsou části `MeshBody152`, `MeshBody327` a `MeshBody331`. Nyní jsou hlavní části vozů obarveny barvou reprezentující jejich příslušnost do závodních týmů. Tato barva je přebírána z atributu `teamColour` objektu `DriverInfo`. Pokud by závodní tým neměl definovanou svoji barvu, bude použita barva fialová.

Pro jednoznačnou identifikaci závodníků je přidán ke každému vozu popisek obsahující jezdcovu zkratku. Tato zkratka je také převzata z atributu `tla` objektu `DriverInfo`. Vytvoření jezdcovy jmenovky za použití 3D objektu `SCNText` je možno vidět ve spodní části ukázky 24. Jmenovka je poté přiřazena jako potomek každého vozu a je obarvena stejnou barvou jako závodní vůz.

Obarvené vozy s přiřazenými jmenovkami jsou na obrázku 28.

```
override func viewWillAppear(_ animated: Bool) {
    ...
    for driver in LapManager.shared.driverInfo{
        ...
        driverCar.name = "\(driver.racingNumber)"
        let nodeMaterial = SCNMaterial()
        nodeMaterial.diffuse.contents = UIColor(
                                        Color(hex: driver.teamColour) ?? .purple)

        for partName in ["MeshBody152", "MeshBody327", "MeshBody331"]{
            driverCar.childNode(withName: partName,
            recursively: true)!.geometry!.firstMaterial = nodeMaterial
        }
        ...
        let label = SCNText(string: "\(driver.tla)", extrusionDepth: 2)
        let labelNode = SCNNode(geometry: label)
        labelNode.name = "Label"
        labelNode.scale = SCNVector3(x: 0.25, y: 0.25, z: 0.25)

        labelNode.position.x -= 0.1
        labelNode.position.y += 0.1

        labelNode.geometry?.firstMaterial = nodeMaterial
        driverCar.addChildNode(labelNode)
    }
    ...
}
```

Ukázka kódu 24: Hlavní kroky potřebné k rozlišení závodníků na okruhu.

Obrázek 28: Zobrazení rozeznatelných vozů na závodním okruhu během *FORMULA 1 ROLEX AUSTRALIAN GRAND PRIX 2023*.

6.8 Animace 3D objektů

Samotný pohyb modelů vozů je svázán s funkcí `renderer(_)` protokolu `ARSCNView-Delegate` zobrazené v ukázce kódu 25. Tato funkce zajišťuje překreslení všech objektů ve scéně. Při použití spuštění bloku kódu na hlavním vlákně asynchronně je zajištěn plynulý proces vykreslení bez hrozby „zamrznutí“ uživatelského rozhraní.

```
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    DispatchQueue.main.async { [self] in
        for carPositions in LapManager.shared.allCarsPositions{
            for car in carPositions{
            if let trackCar = trackPathNode.childNodes.first(where: { node in
                node.name == "\(car.carNumber)"
            }), let trackCarLeadingNode = trackPathNode.childNodes.first(
                where: { node in
                node.name == "\(car.carNumber)-heading"
            }){
                let normalizedX = Float(car.x).normalize(
                            min: Float(minX), max: Float(maxX))
                let normalizedY = Float(car.y).normalize(
                            min: Float(minY), max: Float(maxY))
                let zLow = Float(car.z).normalize(
                        min: Float(minZ), max: Float(maxZ), from: 0, to: 0.05)
                let nextTrackCarPosition = SCNVector3(
                            x: normalizedY, y: zLow, z: normalizedX)

                trackCarLeadingNode.position = nextTrackCarPosition
                let trackCarMoveAction = SCNAction.move(to:
                nextTrackCarPosition, duration: 0.5/Double(LapManager.shared
                                                .allCarsPositions.count))
                trackCar.runAction(trackCarMoveAction)
                }
            }
        }
    }
}
```

Ukázka kódu 25: Funkce používaná k překreslení obsahu zobrazované scény.

Vnitřní cyklus projde všechny aktuální pozice vozů a pro každý vůz provede následující akce:

- Výběr konkrétního vozu ve scéně za pomoci identifikátoru obsahujícího závodníkovu zkratku.
- Vytvoření konstanty `nextTrackCarPosition` datového typu `SCNVector3`, která obsahuje další pozici pro konkrétní vůz.

- Vytvoření akce `SCNAction`, která zajistí animaci vozu z aktuální pozice na trati na novou.

Samotný posun závodního vozu po trati ovšem nestačí. Vůz se prozatím pouze pohybuje po trati, ale nedochází k jeho rotaci v závislosti na zatáčkách závodního okruhu. Tento problém řeší použití „omezení“ (anglicky *constraint*).

Toto omezení je definováno v ukázce kódu 26 a spočívá v definici skrytého modelu, jenž je umístěn před každým závodním vozem. Tento model nemá žádnou geometrii, je pouze unikátně identifikován podle příslušného závodního vodu s přídavkem `-heading`. Každému závodnímu vozu je poté definováno omezení spočívající v přiřazení `SCNLookAtConstraint(target: trackCarLeadingNode)` s tím, že `trackCarLeadingNode` je onen zmíněný skrytý model. Toto omezení zajišťuje, že model vozu bude v každém okamžiku zobrazení ve scéně směřovat přední částí vozu směrem k `trackCarLeadingNode`.

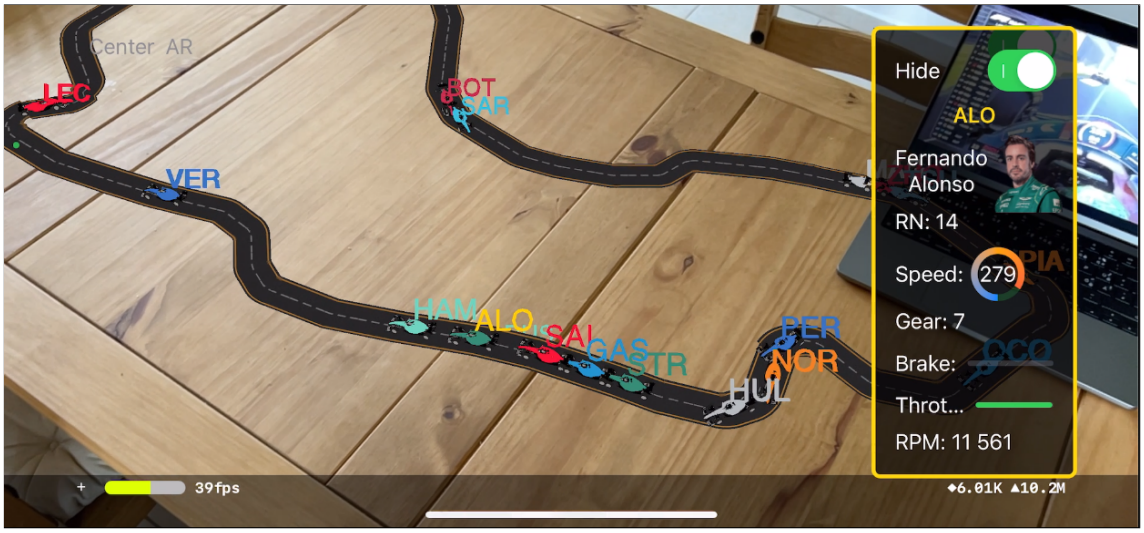
```
override func viewWillAppear(_ animated: Bool) {
    ...
    for driver in LapManager.shared.driverInfo{
        ...
        let trackCarLeadingNode = SCNNode()
        trackCarLeadingNode.name = "\(driver.racingNumber)-heading"
        ...
        let bill = SCNBillboardConstraint()
        labelNode.constraints = [bill]

        let carLook = SCNLookAtConstraint(target: trackCarLeadingNode)
        carLook.localFront = SCNVector3(0, 0, 1)
        carLook.worldUp = SCNVector3(0, 1, 0)
        carLook.isGimbalLockEnabled = true
        driverCar.constraints = [carLook]
    }
    ...
    trackPathNode.addChildNode(trackCarLeadingNode)
}
```

Ukázka kódu 26: Hlavní kroky potřebné k zajištění pohybu a správné orientace vozů ve scéně.

Pokud se vrátíme k ukázce kódu 25, můžeme si všimnout, že dochází nejen k vytvoření akce animace závodního vozu, ale také k posunování `trackCarLeadingNode`. Ten je vždy „o krok napřed“, a tím je zajištěno správné natočení modelu Formule 1 vůči jeho následující pozici.

Posledním modelem, který vyžaduje aktualizaci během překreslení scény, je jezdcův popisek. Tento popisek byl přiřazen jako potomek daného závodního vozu. Přidáním omezení typu `SCNBillboardConstraint` je zajištěno neustálé korektní natočení textu směrem ke kameře. K této funkcionalitě, která podstatně přidá na přehlednosti, stačí pouhé dva řádky kódu.

6.9 Zobrazení detailů vozu a závodníka



Obrázek 29: Zobrazení detailu závodníka v rozšířené realitě.

Posledním bodem pro splnění definovaných požadavků na aplikaci je uživatelem vyvolané zobrazení informací o závodníkovi v rozšířené realitě. Zobrazení těchto detailů je možné vidět na obrázku 29. Zobrazená data jsou načítána z již zmiňovaného objektu `DriverInfo`, jenž obsahuje tyto zobrazované atributy:

- Závodníkova zkratka – `tla`
- Závodníkovo plné jméno – `firstName` a `lastName`
- Závodníkovo číslo – `racingNumber`
- Webová adresa pro získání fotografie – `headshotURL`

Dále jsou data doplněna informacemi o voze z atributu `CarChannels` objektu `carData`. Jedná se o následující data:

- Rychlost vozu
- Aktuálně zařazený rychlostní stupeň
- Procentuální vyjádření aplikovaného brzdného pedálu
- Procentuální vyjádření aplikovaného plynového pedálu
- Otáčky motoru

Tyto detaily o voze a závodníkovi jsou zobrazovány v případě, že dojde k zachycení výběru konkrétního vozu uživatelskou interakcí stisknutí. Na znamení výběru je závodníkova zkratka přebarvena na žlutou barvu a dojde k zobrazení detailu. Samotný rámeček detailu je také označen žlutou barvou, aby byla zajištěno dostatečné logické spojení mezi zobrazeným detailem a vybraným vozem.

```
@objc func handleTap(_ gestureRecognizer: UITapGestureRecognizer) {
    guard gestureRecognizer.state == .ended else {
        return
    }
    let hitTestOptions = [SCNHitTestOption: Any]()
    let hitResults: [SCNHitTestResult] = arView.hitTest(
    gestureRecognizer.location(in: arView), options: hitTestOptions)
    if !hitResults.isEmpty{
        for res in hitResults{
            if res.node.name != nil{
                -- nalezení konkrétního vozu podle názvu --
                -- obarvení popisku vozu --
            }
        }
        contentView!.showDetails = true
    } else {
        if lastTappedCarNumber != nil{
            let driver = LapManager.shared.driverInfo.first { driver in
                driver.racingNumber == lastTappedCarNumber
            }
            -- navrácení barvy popisku  --
        }
    }
}
```

Ukázka kódu 27: Zachycení uživatelské interakce pro zobrazení detailu vozu.

Proces zachycení uživatelské interakce je zobrazen v ukázce kódu 27. Místo, kde se uživatel dotkl displeje, je převedeno na souřadnice vztažené k zobrazované scéně komponentou `ARSCNView`. Následně je proveden tzv. `HitTest`[23], jehož výsledek určí, zda se v tomto místě nachází nějaký model. Pokud má tento model nastavený název, jenž se shoduje s názvem modelu závodního vozu, je provedeno jeho obarvení. V případě, že se na daném místě žádný model vozu nenachází, je popisek posledního vybraného vozu obarven zpět na svoji původní barvu dle příslušného týmu.

[23]Označení pro proces, kdy dojde k vyhodnocení, zda se na určitém bodě nachází daný objekt.

7 Použité technologie a metody

Tato kapitola popisuje technologie a metody, které byly využity k vývoji výsledné aplikace. Jedná se o použité návrhové vzory, jež usnadňují komunikaci mezi jednotlivými objekty a přináší do návrhu aplikace určitý řád a pořádek. Po popisu návrhových vzorů následuje popis použitých frameworků, jež usnadňují řešení určitých problémů.

- **Návrhový vzor Singleton**
  Jedná se o návrhový vzor, který je v aplikaci použitý u několika tříd – konkrétně se jedná o `SignalRManager`, `DemoDataFeeder` a `LapManager`. Aplikace návrhového vzoru je u těchto tříd potřebná k zajištění existence pouze jedné instance za běhu aplikace. Tyto třídy mají vlastní vnitřní stavy, jež jsou pro chod aplikace velmi důležité, a je nutné, aby byla možnost v kódu interagovat právě s jednou instancí těchto tříd.

- **Návrhový vzor MVVM**
  Tento návrhový vzor (*Model–View–ViewModel*) rozděluje logiku návrhu na tři kategorie. *Model* reprezentuje samotný předpis pro vytváření objektů. *View*, neboli uživatelské rozhraní, obsahuje například definici podoby tlačítek a rozvržení prvků na obrazovce. Poslední komponentou tohoto návrhového vzoru je *ViewModel*. Jedná se o objekt, jenž obsahuje data z modelu a obohacuje je o aplikační logiku potřebnou pro zobrazení ve *View*. Aplikace jej využívá u struktury `LapViewModel`, jež obsahuje přídavné funkce používané především pro účely *View*. Například funkce `func colorForDriver(number: Int) -> Color`, která podle závodníkova čísla vrátí příslušnou barvu týmu, respektive vozu.

- **Frameworky ARKit + SceneKit**
  Tyto dva frameworky jsou nativně vyvinuté společností Apple pro tvorbu 3D obsahu. ARKit je knihovna používaná k zobrazení objektů v rozšířené realitě. Knihovnu SceneKit je možné využít především pro vývoj 3D her bez využití rozšířené reality. Aplikace využívá jejich nativní kombinaci v podobě `ARSCNView`, které je použito pro zobrazení 3D objektů v rozšířené realitě. Část SceneKit je využita k programovému vytváření objektů, pro jejichž vizualizaci se pak používá ARKit.

- **Framework SignalRSwift**
  Jak již bylo zmíněno, knihovna SignalRSwift je využita pro komunikaci se servery Formule 1 pomocí komunikačního protokolu SignalR. Tato knihovna je vyvinuta uživatelem Jordan Camara a je dostupná skrze balíčkového manažera CocoaPods. Knihovna je napsána v jazyce Swift ve verzi 3.0 s licencí MIT.

- **Framework SCNPath**
  Pro vizualizaci trati je využita knihovna SCNPath uživatele Max Cobb dostupná v balíčkovém manažeru CocoaPods nebo Swift Package Manager. Knihovna je psána v jazyce Swift ve verzi 5.0.

- **Framework FocusNode**
  Knihovna FocusNode je od stejného vývojáře Max Cobb jako předchozí knihovna SCNPath. Díky použití této knihovny je možné uživateli vizualizovat detekované plochy, na které je možné umístit virtuální objekty. Knihovna je psána v jazyce *Swift* ve verzi 5.0 a je dostupná skrze použití CocoaPods nebo Swift Package Manager.

- **Framework SwiftCollections**
  Knihovna vyvinutá vývojovým týmem společnosti Apple byla pro snadnější integraci do projektů upravena uživatelem haifengkao k použití s CocoaPods. Tato knihovna dovoluje snadnější práci s datovým typem `Dictionary` (slovník). Je využita v části aplikace, jež zajišťuje načítání offline dat ze souborů.

- **Framework DataCompression**
  Jedná se o stěžejní knihovnu, bez které by nebylo možné použít zdrojová data. Tato knihovna je použita k dekompresi přijatých dat pomocí algoritmu Inflate. Knihovna je dostupná při použití správce knihoven CocoaPods. Jedná se o knihovnu, která podporuje prozatím nejvyšší verzi jazyku Swift, 5.7. Knihovnu vyvinul uživatel Markus Wanke.

- **Framework Fast F1**
  Díky této knihovně je zajištěno nahrávání závodů. Její dokumentace zároveň sloužila jako zdroj informací k API serveru Formule 1. Kromě možnosti nahrávání dat během probíhajícího závodu nabízí nástroje pro datovou analýzu s využitím programovacího jazyku Python. Knihovna je dostupná v repozitáři GitHub na adrese: `https://github.com/theOehrly/Fast-F1`.

8 Shrnutí výsledků

Během vývoje aplikace byly překonány určité překážky, které nebyly ve fázi návrhu zpočátku viditelné. Jednalo se především o komunikaci se serverem, kde nedocházelo ke korektnímu volání metod v mobilním zařízení. Tato překážka byla posílena absencí veřejné dokumentace ke konkrétní implementaci protokolu SignalR serveru Formule 1. V návaznosti na tyto překážky bylo nutné upravit uživatelské rozhraní aplikace tak, aby byla zajištěna požadovaná funkcionalita.

Další omezení přineslo vykreslení závodního okruhu ve 2D. Kvůli nenalezení spojitosti v datech nebylo například možné detekovat pozici startu daného kola. Proto je nutný zásah uživatele k zaznamenání konce procesu mapování okruhu.

Zmíněné překážky byly odstraněny použitím alternativních metod, které vedly ke stejnému cíli. Absence oficiální dokumentace byla z části vyřešena dokumentací nástroje Fast F1, jenž určité zprávy zasílané serverem popisuje. Jeden z důvodů vzniku těchto překážek je fakt, že použití protokolu SignalR není pro nativní vývoj na platformě Apple oficiálně podporováno. Během zobrazování trati bylo také vysledováno, že při použití velkého množství bodů k vykreslení 3D trati nedochází k jejímu zobrazení. Z důvodu, že se jedná o vykreslení trati s použitím knihovny, bylo nutné omezit počet bodů, jenž bude k vykreslení použit. Toto omezení může v určitých případech znamenat nenavazování úseků trati.

Po překonání těchto překážek bylo dosaženo požadovaného výsledku – vizualizace pozic závodníků Formule 1 v rozšířené realitě s využitím dat vysílaných v reálném čase. Je nutno podotknout, že podobná implementace těchto „živých“ dat není v současné chvíli k dispozici. Ostatní dostupné nástroje zobrazují dostupná data pouze formou tabulek či 2D zobrazení. Proto lze tuto aplikaci označit jako „*Proof Of Concept*“ – důkaz použitelnosti návrhu.

9 Závěr

V první části této práce byl čtenář seznámen s historií Formule 1. Toto téma je stěžejní k pochopení rozsahu a významu, jaký má Formule 1 ve světě motorsportu. Po představení technologií používaných pro rozšířenou realitu a problematiky streamování dat v reálném čase je čtenáři prezentována praktická část práce.

Praktická část pojednává o návrhu a ideální implementaci technologií pro zajištění požadované funkcionality. Jsou představeny návrhy uživatelského prostředí formou takzvaných wireframů a plánované interakce v rámci aplikace pomocí sekvenčních diagramů.

Část kapitoly o implementaci návrhu podrobně představuje práci s přijatými daty a jejich následné transformace. Dále je představena konečná podoba uživatelského prostředí v průběhu zpracování dat. Výsledná aplikace dokáže zpracovávat dva zdroje dat. Prvním je práce s historickými daty, které je nutné nejdříve upravit pro zajištění kompatibility s formátem JSON. Druhým typem dat jsou data přijímaná ze serverů Formule 1. Data z obou zdrojů jsou následně zpracována a vizualizována uživateli v podobě 2D zobrazení nebo zobrazení v rozšířené realitě. Tím jsou splněny stanovené cíle diplomové práce. Zmíněná zobrazení ve 2D a v rozšířené realitě jsou demonstrována na snímcích obrazovky aplikace v části, která pojednává o implementaci řešení.

Vytvořená aplikace nabízí prostor k dalším budoucím vylepšením, jež by zlepšila jak uživatelskou zkušenost, tak práci s daty uvnitř aplikace. V oblasti zobrazování dat lze například implementovat zobrazení aktuálního pořadí jezdců, nebo vizualizovat infografiku o použitých typech pneumatik a jejich stáří.

Literatura

[1] Formula One Art & Genius. Aug. 2018. Dostupné z: https://www.f1-grandprix.com/

[2] Martin Williamson. A timeline of Formula One. Dostupné z: http://en.espnf1.com/f1/motorsport/story/3836.html

[3] 1950 Formula One season. Oct. 2022, page Version ID: 1117919127. Dostupné z: https://en.wikipedia.org/w/index.php?title=1950_Formula_One_season&oldid=1117919127

[4] The Physics of Racing. Dostupné z: https://www.nas.nasa.gov/About/Education/Racecar/physics.html

[5] Venturiho efekt - frwiki.wiki. Dostupné z: https://cs.frwiki.wiki/wiki/Effet_Venturi

[6] Remi Humbert. BMW Turbo F1 Engine. Dostupné z: http://www.gurneyflap.com/bmwturbof1engine.html

[7] 1988 Formula One World Championship. Oct. 2022, page Version ID: 1113703431. Dostupné z: https://en.wikipedia.org/w/index.php?title=1988_Formula_One_World_Championship&oldid=1113703431

[8] Cóilín Higgins. The 10 biggest innovations in Formula 1 history: active suspension, halo, fan car & more. Dostupné z: https://www.autosport.com/f1/news/the-10-biggest-innovations-in-formula-1-history-active-suspension-halo-fan-car-more/10125979/

[9] Groundbreaking F1 Gearbox: 1989 Ferrari 640. Dostupné z: https://rossoautomobili.com/blogs/magazine/groundbreaking-f1-gearbox-1989-ferrari-640

[10] Matthew Neill. The slippery history of traction control in F1. Feb. 2021. Dostupné z: https://racingnews365.com/traction-control-in-f1-a-short-history

[11] Paul Evans. Formula One KERS explained. Mar. 2009, section: Automotive. Dostupné z: https://newatlas.com/formula-one-kers/11324/

[12] NextgenAutoVideos. F1 2014 - Renault Sport F1 - V6 turbo hybrid in 3D. 2014. Dostupné z: https://www.youtube.com/watch?v=a-zUgVcFCb4

[13] Fédération Internationale de l'Automobile. fia_2026_formula_1_technical_regulations_pu_-_issue_1_-_2022-08-16.pdf. Aug. 2022. Dostupné z: https://www.fia.com/sites/default/files/fia_2026_formula_1_technical_regulations_pu_-_issue_1_-_2022-08-16.pdf

[14] Speciál k 90. narozeninám: Jak Bernie Ecclestone postavil celý byznys kolem formule 1 | GPF1. Oct. 2020. Dostupné z: https://gpf1.cz/special-jak-bernie-ecclestone-postavil-cely-byznys-kolem-formule-1/

[15] F1 World Champions - Every World Champion in Formula 1 History. Mar. 2023. Dostupné z: https://racingnews365.com/every-world-champion-in-formula-1-history

[16] Fédération Internationale de l'Automobile. fia_2023_formula_1_sporting_regulations_-_issue_4_-_2023-02-22. Aug. 2022. Dostupné z: https://www.fia.com/sites/default/files/fia_2023_formula_1_sporting_regulations_-_issue_4_-_2023-02-22.pdf

[17] How Much Data Does An F1 Car Generate? | F1 Data Explained. Nov. 2022, running Time: 661 Section: F1 News. Dostupné z: https://f1chronicle.com/how-much-data-does-an-f1-car-generate/

[18] Koďousková, B. Rozšířená realita: využití AR ve firmách a startupech. July 2020. Dostupné z: https://www.rascasone.com/cs/blog/rozsirena-realita-ar-vyuziti-firmy-aplikace

[19] Working with Image Tracking in ARKit. Mar. 2022, section: iOS. Dostupné z: https://www.appcoda.com/arkit-image-tracking/

[20] Nextech. What Are The Different Types of Augmented Reality? Dostupné z: https://www.nextechar.com/blog/what-are-the-different-types-of-augmented-reality

[21] Iqbal, K. Další informace o 3D formátech souborů a rozhraních API, která mohou otevírat a vytvářet 3D soubory. Dostupné z: https://docs.fileformat.com/cs/3d/

[22] Kraakman, N. The Best 3D Model File Format for your AR Applications. Mar. 2018. Dostupné z: https://headjack.io/blog/best-3d-model-file-format-ar-applications/

[23] USDZ schemas for AR. Dostupné z: https://developer.apple.com/documentation/arkit/usdz_schemas_for_ar

[24] d937c9e9-05fd-46e3-b72d-eb353b79bfbd.png (770×884). Dostupné z: https://docs-assets.developer.apple.com/published/ffb3831f78/d937c9e9-05fd-46e3-b72d-eb353b79bfbd.png

[25] f76d63a3-7620-40d1-9e52-0d9ad6329678.png (1360×633). Dostupné z: https://docs-assets.developer.apple.com/published/b99f86dcfb/f76d63a3-7620-40d1-9e52-0d9ad6329678.png

[26] Inc, A. Introducing Reality Converter - Latest News - Apple Developer. Dostupné z: https://developer.apple.com/news/?id=01132020a

[27] Inc, A. AR Creation Tools - Augmented Reality. Dostupné z: https://developer.apple.com/augmented-reality/reality-composer/

[28] Managing Session Life Cycle and Tracking Quality. Dostupné z: https://developer.apple.com/documentation/arkit/managing_session_life_cycle_and_tracking_quality

[29] Krčmář, P. WebSocket jako cesta k úniku z příliš restriktivní sítě. ISSN: 1212-8309. Dostupné z: https://www.root.cz/clanky/websocket-jako-cesta-k-uniku-z-prilis-restriktivni-site/

[30] Melnikov, A.; Fette, I. The WebSocket Protocol. Request for Comments RFC 6455, Internet Engineering Task Force, Dec. 2011, doi:10.17487/RFC6455, num Pages: 71. Dostupné z: https://datatracker.ietf.org/doc/rfc6455

[31] ws: a Node.js WebSocket library. Apr. 2023, original-date: 2011-11-09T22:32:45Z. Dostupné z: https://github.com/websockets/ws

[32] Introduction | Socket.IO. Mar. 2023. Dostupné z: https://socket.io/docs/v4/

[33] Pusher vs. Socket.IO: Features, pros and cons, and use cases. Dostupné z: https://ably.com/topic/pusher-vs-socketio

[34] Introduction | Building Real-Time Laravel Apps with Pusher. Dostupné z: https://pusher-community.github.io/real-time-laravel/

[35] bradygaster. Overview of ASP.NET Core SignalR. Feb. 2023. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction

[36] bradygaster. Introduction to SignalR. Sept. 2020. Dostupné z: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

[37] bradygaster. ASP.NET Core SignalR supported platforms. Feb. 2023. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/supported-platforms

[38] Schaefer, P. Fast F1. Apr. 2023, original-date: 2020-04-18T10:00:31Z. Dostupné z: https://github.com/theOehrly/Fast-F1

[39] Road texture seamless , jordan Holbrook. Nov. 2018. Dostupné z: https://www.artstation.com/artwork/Bm1N2D

[40] Ferrari-F1-75-2022 | 3D CAD Model Library | GrabCAD. Dostupné z: https://grabcad.com/library/ferrari-f1-75-2022-1

A Ukázka streamovaných dat

```
{'C': 'd-CDF51A6A-B,0|Ee,0|Ef,11|e,149|v,12FC|u,12FD|a,2|Y,29F|Eg,0|
i,27D|W,D1|d,52|j,1|X,35|s,16|h,1|l,F|Z,D|m,3A40',
'M': [{'H': 'Streaming', 'M': 'feed', 'A': ['Position.z',
'lZc/TxwxEMW/i+tb5Plrz/apEykUCVEKFFGcIiCCS4Xuu8e7nt2YxmuaEyfxNOPnNz/PvYUvz6/ny
/n5Kcw/3sLt+fHh9XL/+CfMASPSFGkCu4U0I82oN0ZsinQXTuHT0+Xl/PAa5rcAy8fXy/3lb/kaPj/
dvtz/+l3+5VuYJ1JMp/A9zDlrPIW7MAOk6ylgR2Oqq2RKyx+rxIqEOxKOJKuG1VyiRQKx1xqlWkczk
It4EfXPI7UQAqA13UGvPeLl8EUFGVFchYtKe4dChlVlnHIVoSyi3BORUC2FFLcG8+J4zwvAWmkSEXA
RLaKeF4DZSxm3hXp3i4bVCU5KbSHqichwFREjtpV6nkvm7Ujp3ZFST7QY7SJpRNTzoYSvqpbx8HuKS
2C7kZBY+0s5UhNZkW4iPOZliFJs+tOefRKj3xNHS1VFS/pSzwqO4JlI7/zL3Uiw1QYnzva/wev1dAS
XdIOJDaJ+CC5ufLZkg3CBiFsuMgzTRasVuJeRY7qY08V2++SYLh5aBOTcgqxPFzGnCwMM04V8Ek1kS
206pAt7akt/G88Rj+ii5o6z2TBcyoDUmU9pHC5ZHS55HC7MNbKkIE0g+nDBjRMaaRguUTYRcPPg9OE
CqA4Xk8bwA7i4KFnCcbiIOlwMeRQuZSHwRBQ4+XNNcAQXslSRVB57GYZLmSk3MMb0MbhI4qzyIbZof
XMsGg6zxbzB/SFYQ9tni28hlBsf+mhhf60TKA+jJdG2uJC9o9jB4uKZyMo6jBaO1QfTjbFoh2gpl+p
oUWjXnS5a9r2lpG8YLexWaBxGC8W07SO5NrPbR4vE7He7m3e4tzDrRgnCtlL3sYYdLXuOjtGybRNlv
6TG8AO0SJ33DFmH0UL73QK0ke2jZV0hVrTQtmGuOTpAC9RfHGU/jTrMFvDnZhJga9jy8/oP',
'2023-03-19T17:23:27.574865Z']}]}
{'C': 'd-CDF51A6A-B,0|Ee,0|Ef,11|e,149|v,12FC|u,12FD|a,2|Y,29F|Eg,0|i,27D
|W,D1|d,52|j,1|X,35|s,16|h,1|l,F|Z,D|m,3A41', 'M':
[{'H': 'Streaming', 'M': 'feed', 'A': ['TimingData',
{'Lines': {'4': {'Sectors': {'2': {'Segments': {'2': {'Status': 2048}}}}}}}
, '2023-03-19T17:23:28.133Z']}]}
```

Ukázka kódu 28: Vzorek streamovaných dat před zpracováním.

```
{"Position":
[{"Timestamp":"2023-03-19T17:23:26.9349623Z", "Entries":{
"1":{"Status":"OnTrack","X":-3627,"Y":8860,"Z":117},
"2":{"Status":"OnTrack","X":-966,"Y":-766,"Z":119},
"4":{"Status":"OnTrack","X":-4035,"Y":469,"Z":116},
"10":{"Status":"OnTrack","X":-3376,"Y":6813,"Z":114},
"11":{"Status":"OnTrack","X":-3655,"Y":21129,"Z":119},
"14":{"Status":"OnTrack","X":-3460,"Y":18225,"Z":112},
"16":{"Status":"OnTrack","X":-4241,"Y":9478,"Z":125},
"18":{"Status":"OnTrack","X":-4353,"Y":12309,"Z":118},
"20":{"Status":"OnTrack","X":-121,"Y":-5551,"Z":113},
"21":{"Status":"OnTrack","X":-1283,"Y":194,"Z":118},
"22":{"Status":"OnTrack","X":-2920,"Y":4763,"Z":113},
"23":{"Status":"OnTrack","X":-2392,"Y":3422,"Z":118},
"24":{"Status":"OnTrack","X":-584,"Y":-5571,"Z":113},
"27":{"Status":"OnTrack","X":-578,"Y":-5575,"Z":113},
"31":{"Status":"OnTrack","X":-4038,"Y":9349,"Z":120},
"44":{"Status":"OnTrack","X":-3504,"Y":7803,"Z":116},
"55":{"Status":"OnTrack","X":-4235,"Y":11770,"Z":113},
"63":{"Status":"OnTrack","X":-5003,"Y":14097,"Z":132},
"77":{"Status":"OnTrack","X":-4011,"Y":-771,"Z":113},
"81":{"Status":"OnTrack","X":-1495,"Y":-4890,"Z":113}}},
```

Ukázka kódu 29: Dekomprimovaná data `Position.Z` ve formátu JSON.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2022/2023

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení: **Bc. Roman Auersvald**
Osobní číslo: **I1900799**
Adresa: Žižkova 353, Jilemnice, 51401 Jilemnice, Česká republika

Téma práce: Zpracování dat z vozů F1 v reálném čase s využitím rozšířené reality
Téma práce anglicky: Real-time processing of data from F1 cars using augmented reality
Jazyk práce: Čeština

Vedoucí práce: Ing. Karel Mls, Ph.D.
Katedra informačních technologií

Zásady pro vypracování:

Cíl:
Cílem práce je navrhnout a následně implementovat řešení pro vizualizaci dostupných dat z vozů F1 v prostředí rozšířené reality.
Osnova:
Úvod do řešené problematiky
Představení softwarových nástrojů
Zpracování dat a implementace
Vyhodnocení
Závěr

Seznam doporučené literatury:

PLINI, Federico. *Augmented reality and live racing: the future of the F1 Broadcasting Industry*. 2021. PhD Thesis.
Nedal Sawan, Ahmed Eltweri, Caterina De Lucia, Luigi Pio Leonardo Cavaliere, Alessio Faccia, and Narcisa Roxana Moşteanu. 2021. Mixed and Augmented Reality Applications in the Sport Industry. In 2020 2nd International Conference on E-Business and E-commerce Engineering (EBEE 2020). Association for Computing Machinery, New York, NY, USA, 55–59. https://doi.org/10.1145/3446922.3446932
Chad Goebert, Gregory P. Greenhalgh,
A new reality: Fan perceptions of augmented reality readiness in sport marketing,
Computers in Human Behavior,
Volume 106, 2020, 106231, ISSN 0747-5632,
https://doi.org/10.1016/j.chb.2019.106231.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

© IS/STAG, Portál – Podklad kvalifikační práce , auersro1, 13. dubna 2023 21:36