TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Interaktivní úloha s robotem Pepper v prostorách děkanátu FM

Bakalářská práce

Studijní program: B2615 – Elektronika a informatika

Studijní obor: 2612R011-Elektronické, informační a řídící systémy

Autor práce: **Matyáš Horký**

Vedoucí práce: Ing. Miroslav Holada, Ph.D.

Ústav informačních technologií a elektroniky

Liberec 2021

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Interactive task with Pepper robot in FM Dean´s office

Bachelor thesis

Study programe: B2615 – Electrotechnology and informatics

Study branch: 2612R011-Electronic, information and control systems

Autor : **Matyáš Horký**

Supervisor: Ing. Miroslav Holada, Ph.D.

Ústav informačních technologií a elektroniky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Interaktivní úloha s robotem Pepper v prostorách děkanátu FM

Jméno a příjmení: **Matyáš Horký**
Osobní číslo: M18000030
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s humanoidním robotem Pepper na pracovišti školitele. Proveďte rešerši stávajícího stavu dostupných interaktivních úloh pro robota Pepper.
2. Navrhněte proprietární interaktivní program, pomocí kterého bude robot komunikovat s návštěvníky děkanátu FM.
3. Navržený program realizujte a ověřte jeho funkcionalitu.
4. V závěru diskutujte výhody návrhu a možná bezpečnostní rizika.

Rozsah grafických prací: Dle potřeby dokumentace
Rozsah pracovní zprávy: 30-40 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] NENCHEV, Dragomir N. a Atsushi KONNO. Humanoid Robots. 1. Berlin, SRN: Elsevier – Health Sciences Division, 2016. ISBN 9780128045602.
https://www.softbankrobotics.com
[2] VANER, Pavel: Spolupráce robotů NAO: NAO Robots collaboration. Liberec: Technická univerzita v Liberci, 2018. Bakalářské práce. Technická univerzita v Liberci.
[3] EICHLER, Miroslav: Využití dostupných senzorů robota Nao pro detekci objektů a mapování okolí: Use available Nao robots' sensors to detect objects and map of its surrounding. Liberec: Technická univerzita v Liberci, 2018. Bakalářské práce. Technická univerzita v Liberci.

Vedoucí práce: Ing. Miroslav Holada, Ph.D.
Ústav informačních technologií a elektroniky

Datum zadání práce: 19. října 2020
Předpokládaný termín odevzdání: 17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

17. května 2021 Matyáš Horký

Poděkování

Rád bych zde poděkoval svým rodičům za umožnění tohoto studia a Ing. Miroslavu Holadovi, Ph.D. za vstřícné vedení i za nepříznivých podmínek.

Abstrakt

Tato práce se zabývá humanoidním robotem Pepper od firmy Softbank robotics a možnostmi jeho využití. Soustředí se na jednotlivé funkce a popisuje jejich funkčnost, složení, použití a uvádí příklad, jak je na robotu vytvořit. V druhá část práce pojednává o konkrétní aplikaci navržené pro použití na TUL za použití robota na verzi systému 2.9. a Android studia v kombinaci s QiSDK. V aplikaci je snaha o využití jednotlivých funkcí uvedených v první části v co nejvyšším rozsahu a zároveň o dodržení logiky fungování a chování robota. Výsledná aplikace bude využívána na TUL.

Klíčová Slova

Pepper, Softbank robotics, QiSDK, Chatbot, rozpoznání obličeje, humanoidní robot

Abstract

This thesis deals with humanoid robot Pepper made by Softbank robotics industries and his usage options. The thesis is focused on particular robot functions and describes its functionality, structure, usage and presents an example, how to create those for this robot. Second part of the thesis deals with specific application designed to be of use at TUL using robot at systém version 2.9. and Android studio with QiSDK implemented. Purpose of application is to merge as many of functions mentioned in first part of the thesis together in one working robot life cycle and creating an application, that can be helpful at TUL.

Key words

Pepper, Softbank robotics, QiSDK, Chatbot, face recognition, humanoid robot

Obsah

Seznam obrázků:

Seznam zkratek

SDK – software development kit

QA – Orgán pro schválení vydání aplikace v Softbank robotics

CC – Command center

1 Robot Pepper

Pepper je na světě první veřejně dostupný humanoidní robot navržený pro sociální účely, který umí rozpoznávat lidské obličeje a emoce. Byl optimalizován pro komunikaci a interakci s lidmi za pomoci jeho chování a zabudovaného tabletu. Momentálně se používá ve více než 2000 institucích, jak pro komerční, tak i pro edukativní účely. [1]

1.1 Hardware robota Pepper

Robot Pepper byl poprvé vyroben roku 2014 a je 120 cm vysoký, 42,5 cm široký a 48,5 cm dlouhý. Váží 28 kg a obsahuje Li-Ion baterii o 30 Ah/ 795 Wh, což vydrží robotu přibližně na 12 hodin provozu.

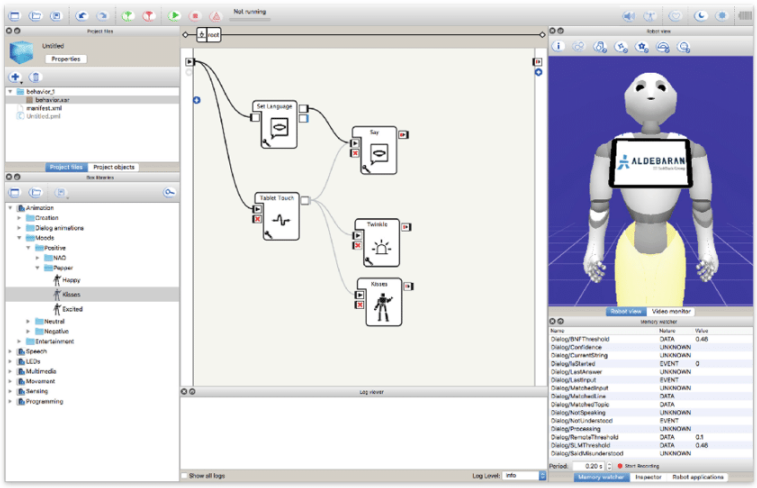
Pepper při svém pohybu disponuje 20 stupni volnosti, které jsou hlavně v oblasti paží a dlaní. To umožňuje plnohodnotné využívání gestikulace při komunikace a robot pak působí přátelštěji. Přesto, že paže a dlaně robota mohou vykonávat poměrně jemné a precizní pohyby, nejsou určeny k přenášení věcí. Jejich konstrukce k tomu není uzpůsobena a lze tedy přenášet předměty lehké (max 500 g) a nepříliš velké, jako například list papíru nebo sklenici vody. To je ovšem pro jeho sociální interakce naprosto dostačující. Na rozdíl od vrchní části těla, která je velmi pohyblivá, spodní část robota je složena ze spodní části trupu uloženého pomocí kloubu (pouze 1 stupeň volnosti) na nosné platformě, sloužící robotu pro přesouvání. To je zajištěno pomocí 3 volně otáčivých (360°) koleček na spodku platformy. Maximální rychlost robota je oficiálně uvedena 5 km/h, ale je v základu omezena na 3 km/h.

Robot je vybaven několika druhy senzorů pro měření vzdálenosti, intenzity doteku, snímání zvuku a několika kamerami. Senzory pro měření vzdálenosti se nacházejí na hlavě a pojízdné platformě. Robot nimi získává údaje o vzdálenosti překážek a lidí v jeho okolí. Senzorů vzdálenosti jsou na robotu dva druhy. Infračervené laserové, využívané k učení vzdálenosti od překážek a ultrazvukové, ke skenování širšího okolí, například místnosti. Senzory doteku jsou umístěny na hlavě a na rukou robota. Pojízdná platforma má senzory pro zjištění nárazu do překážky. Dále je robot vybaven 4 mikrofony, 2 HD kamerami (na puse a čele) a jednou hloubkovou kamerou v oblasti očí a gyrosenzorem.

Na hrudi robota je umístěn LG CNS tablet. Ten s ním nesdílí operační systém a pracuje samostatně a s robotem pouze komunikuje. Tablet měří 246 x 175 x 14.5 mm a ovládá se pomocí trojice tlačítek umístěných na jeho zadní straně. Tablet lze připojit k internetu pouze přes wifi, kdežto robot lze připojit jak přes ethernet, umístěný za odnímatelnou krytkou na hlavě, tak i přes wifi.[2][3][4]

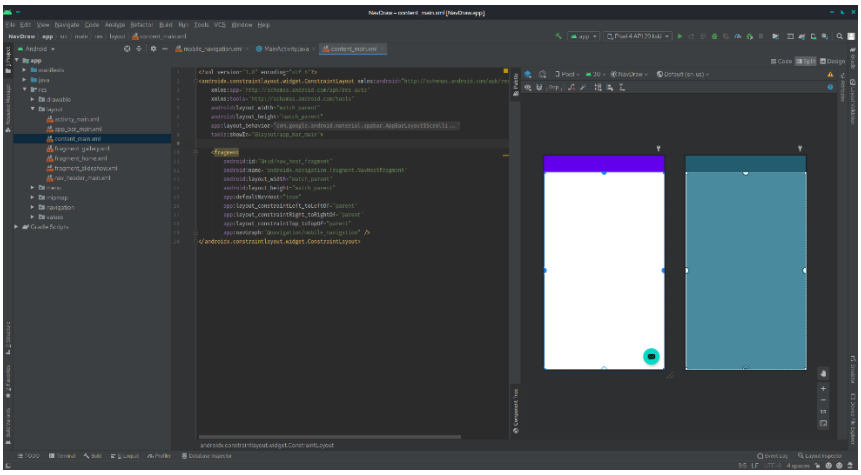
1.2 Software v robotu a využívaný k programování robota

Starší verze robota Pepper (2.1 až 2.8) se programují pomocí NAOqi v jazyce python a v prostředí softwaru Choreographe. Choreographe je blokový, převážně grafický editor, kde z předpřipravených nebo vámi naprogramovaných funkcí skládáte samotný program. Bloky funkcí propojíte časovými linkami v pořadí, v kterém chcete funkce vykonat a případně bloky zacyklíte. Tento způsob přes mnohé snahy o zjednodušení a převedení kostry programu do grafické podoby je na novějších verzích nahrazen standardnějším způsobem programování. Hlavním důvodem bylo přidání tabletu k robotu a Choreographe neumožňuje využití jeho veškerých možností.



Obrázek 1 Snímek obrazovky programu Choreographe [2]

Nynější verze robota se programuje pomocí QISDK, což je knihovna (rozšíření) do frameworku Android studio. Android studio se běžně využívá kromě jiné pro programování aplikací pro mobilní zařízení a tablety a jelikož je dostupné zdarma, je ideální volbou pro programování robota. Po instalaci SDK lze aplikaci tzv. robotizovat, což znamená převést do formy pro robota. Tvorba aplikace pro robota je tedy dosti podobná tvorbě aplikace pro mobilní telefon, pouze musí využívat primárních, předem definovaných metod robota.



Obrázek 2 Snímek obrazovky programu Android studio

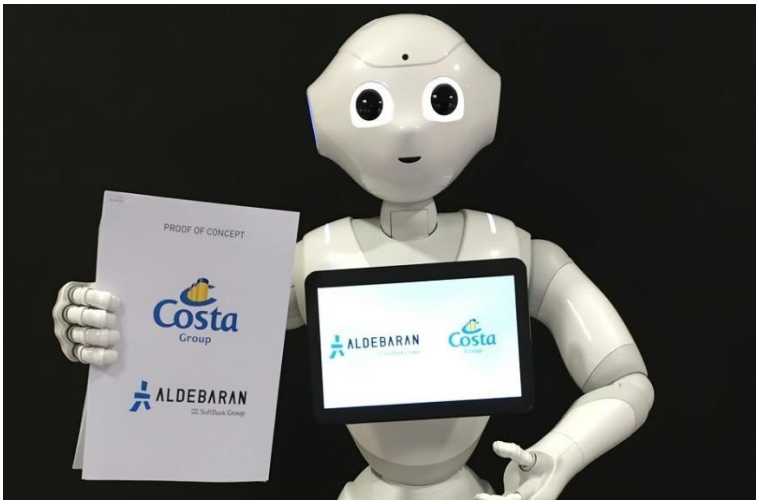
2 Využití robota Pepper

Využití robota Pepper najdeme spíše ve společnosti než v průmyslu. Robot je navržen pro asistenci lidem hlavně při vykonávání služeb, jako: prodej zboží, sekretářská činnost nebo pro kontrolu a podání informace. Při prodeji zboží lze ideálně využít možnosti dotykového tabletu ve spojení s řečí. Robot dokáže oslovit zákazníka a na základě jeho požadavků mu doporučit několik možností zboží, které zákazník pravděpodobně hledá. Možnosti zobrazí na tabletu a po schválení výběru ukáže na mapě cestu ke zboží nebo tam zákazníka přímo dovede. Dovede také zaslat informační a reklamní zprávy například pomocí email. Je možné, že v budoucnu budou prodejny rozložené jinak, aby umožňovaly pojmout co nevíce zboží a veškeré zboží budou vybíráno pomocí terminálu na robotu a rovnou také bezkontaktně placeno. K tomuto účelu by mohl sloužit podobný robot jako je Pepper.

Kromě prodeje zboží se už i nyní robot využívá pro sekretářskou činnost. Dá se vidět na výletních lodích a v lepších hotelích, kde zapisuje seznam hostů, kontroluje jejich totožnost nebo třeba objednává obědy, večeře. Tělo robota je opatřeno několika druhy senzorů se značně předimenzovanou rezervou. Je tedy vhodný i pro práci v nemocnicích, kde robot pomáhá při hlídání pacientů a mohl by také pomáhat v družinách při hlídání dětí.

Dalším běžným využitím robota Pepper je kontrola a podání informací. Velkou výhodou robota je možnost naučit jej mluvit 15 světovými jazyky. Toto se hodí pro předávání informací ve velkých podnicích s mezinárodním personálem. Robot tedy může sledovat například dodržování nošení přileb na rizikových místech a upozornit člověka při nevědomosti, popřípadě odeslat hlášení o porušení nařízení vedoucímu podniku. Pepper byl ještě využit například jako host v televizním pořadu nebo jako roztleskávačka na speciálním sportovním utkání

Běžně se všechny tyto vlastnosti robota kombinují, což je jeho největší výhodou. Díky tomuto chování je robot oproti jiným robotům hodně autonomní a dokáže spoustu jednodušších funkcí udělat kompletně bez přítomnosti člověka, ten je v této fázi ovšem nutný pro závěrečnou kontrolu, jelikož přesnost rozpoznávaní obrazu a řeči je dosti závislá na okolních podmínkách.



Obrázek 2 Pepper jako asistent na výletní lodi [18]

2.1 Konkurence robota Pepper

Softbank Robotics Romeo

Romeo je nástupce robota Pepper od stejné firmy Softbanks Robotics. Využívá velice podobný systém. Romeo byl navržen roku 2011 také pro komunikaci s limy, ale s větší specifikací oproti univerzálnímu robotu Pepper. Robot je zaměřen na pomoc v domácnosti lidem s omezenou možností pohybu (invalidé, senioři). Dokáže stejně jako Pepper rozpoznávat 15 světových jazyků a rozpoznávat lidi podle tváře. Na rozdíl od Pepper je Romeo větší (160 cm) a pohybuje se chůzí, nikoliv otáčením kol. To umožňuje zvýšenou schopnost překonávat překážky a podporuje manipulaci s objekty, takže mezi jeho reálné využití patří otevírání dveří, šuplíků a nošení předmětů. Úkolem Romea je také se chovat jako partner. To zajišťuje navazováním konverzace a podáváním návrhů na trávení volného času.

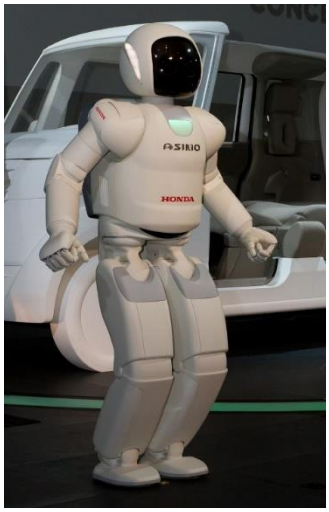
[5]

Asimo od společnosti Honda

Asimo je robot společnosti Honda, který byl zkonstruován s podobným účelem jako Romeo firmy Softbanks Robotics. Byl navržen roku 2000, takže je o 11 starší a měří 120 cm, stejně jako Pepper. Hlavní výhodou Asimo je na humanoidního robota dost plynulý a svižný pohyb (chůze). Dokáže vyvinout rychlost až 10 km/h a velmi spolehlivě se vyhýbá překážkám. To jej dělá ideálním pomocníkem v zalidněných prostorech jako jsou kancelářské prostory velkých firem. Umí se také naučit sekvence činností, jako například uvařit a donést kávu nebo roznést firemní noviny. Avšak oproti Romeu a Pepper strádá v možnostech komunikace s lidmi. Asimo dokáže mluvit pouze dvěma jazyky (Japonština, Angličtina) a nedokáže rozpoznávat jednotlivé obličeje a lidské emoce. Jedná se tedy o robota, který má pomáhat člověku spíše s fyzickou činností, než fungovat jako partner. Tento robot je však experiment a nikdy nebyl na prodej.

[6]

Obrázek 4 robot Romeo od Softbanks robotics [19]



Obrázek 3 Asimo od firmy Honda [20]

2.2 Další roboti (nepřímá konkurence)

Valkyrie Robonaut 5

Humanoidní robot asociace NASA, postaven za účelem budování základen, laboratoří a dalších staveb na planetě Mars. Robot byl postaven roku 2013 v Johnson Space Center. Na robotu byl kladen důraz hlavně na spolehlivost, odolnost a soběstačnost. Z důvodu špatné komunikace mezí Zemí a Marsem, bylo nutné vytvořit robota tak, aby dokázal pracovat samostatně i bez vnějšího řízení. Robot váží přibližně 136 kg a je vysoký 188 cm. Na rukou robota najdeme 3 prsty plus palec, každý prst má 3 články, pouze palec má jen 2 stejně jako u lidské ruky. Řízení robota zajišťují dva procesory intel 7. generace a kapacita baterie je 1,8kWh.

[7]

Sophia

Robot Sophia byl vyroben společností Hanson Robotics roku 2016. Udává se, že se jedná o světově nejinteligentnější robota. Řadí se do skupiny obecné umělé inteligence a za použití internetu dokáže vést konverzaci nebo napodobit osobnost člověka. Robot má pouze horní část lidského těla (od trupu k hlavě) a na hlavě má latexový obličej pro napodobování výrazů. Umělá inteligence tohoto robota je natolik pokročilá, že roku 2017 získala občanství v Saudské Arábii a stala se prvním robotem, který kdy získal státní občasnství.

[8]

Obrázek 6 Robonaut 5 [21]

Obrázek 5 robot Sophia od Hansom Robotics [22]

3 Command center

Jedná se o platformu umožňující do robota nahrávat aplikace a spravovat jeho vlastnosti. Byla navržena pro zjednodušení procesu nahrávání aplikace a jejich sdílení. Nově umožňuje přidávání licencí k aplikacím a ulehčuje hledání.

Command center má 3 vizuální rozhraní.

1) Uživatelské rozhraní umožňuje
   - Přidělování úkolů
   - Přidávání přístupu ostatním uživatelům k robotu
   - Změnu jména robota
   - Vytvořit skupinu spolupracujících robotů
   - Přidat uživatele jednotlivým skupinám
   - Změnit uživatelská práva a omezení
   - Upravovat uživatelské profily

2) Vývojářské rozhraní umožňuje
   - Nahrávat úkoly založené na androidu (.apk)
   - Nahrávat úkoly založené na Naoqi (.pkg)
   - Uložit a obnovit úkol
   - Upravit informace o úkolu
   - Sdílet Beta verzi úkolu
   - Pozvat spolupracovníky
   - Požadovat ověření či zrušení úkolu
   - Vydat úkol do databáze úkolů
   - Distribuovat licenci

3) Rozhraní databáze úkolů
   Databáze úkolů slouží jako portál pro sdílení aplikací a úkolů mezi vývojáři na robotu Pepper. Některé aplikace jsou zdarma, jiných dostanete omezený počet licencí ke koupi robota, ale většina je vždy zpoplatněna. Většina úkolů nacházejících se v databázi obsahuje demo pro určitou funkci robota Pepper, například prodej v obchodě nebo čtení textu atd. Získané úkoly se následně uloží ve vaší uživatelské knihovně. Tu je následně možné jednoduše nahrát a po spuštění aktualizační aplikace nebo restartu robota se úkol objeví v hlavním menu.

[9]

Obrázek 7 ikona aktualizační aplikace [9]

3.1 Vydání aplikace v Command centru

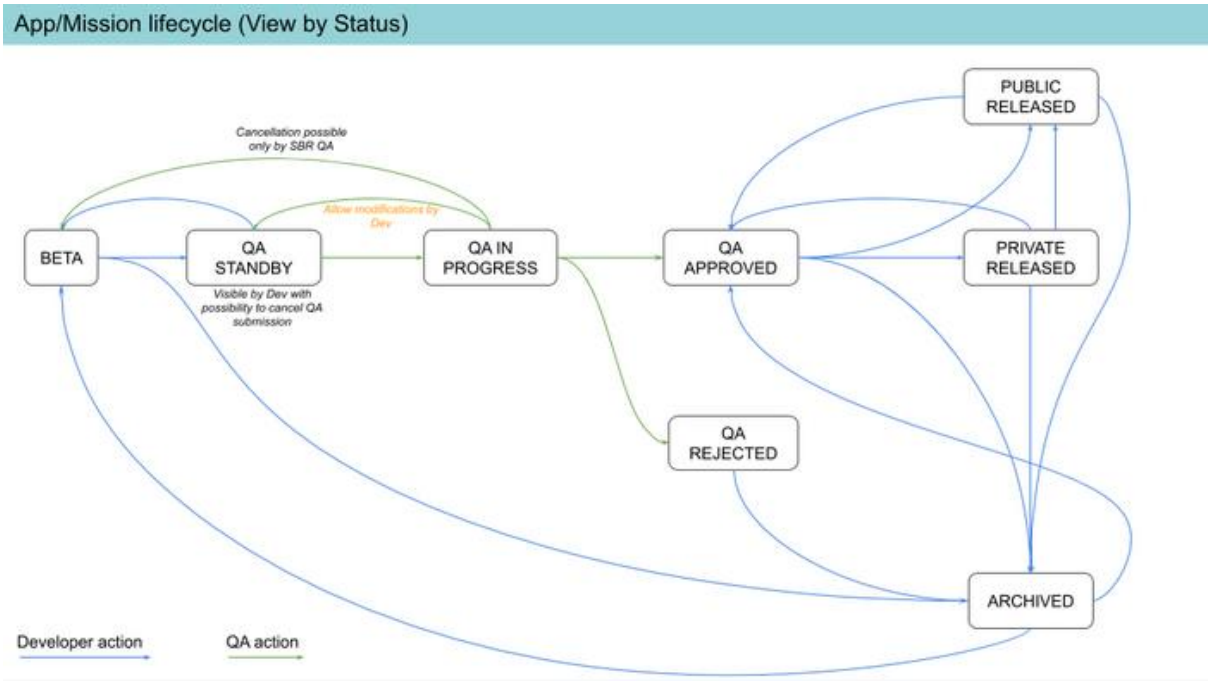
Pro vydání vlastní aplikace je nutné splnit několik kroků.

1) V základu po nahrání aplikace do databáze úkolů se objeví jako beta verze. Ta má následující vlastnosti:
   - Lze používat na vlastních robotech po nekonečně dlouhou dobu
   - Po dobu 30 dnů lze sdílet se zákazníkem anebo partnerem
   - Dá se archivovat (dojde k odinstalaci ze všech robotů)
   - Lze ji odeslat do na QA

2) Cyklus QA
   Cyklus QA se skládá z několika kroků
   - Stand-by, kde úkol čeká ve frontě na otestování
   - In Progress, probíhá testování
   - QA rejected/approved, vyhodnocení, jestli aplikace vyhovuje podmínkám pro přijetí

3) Vydání aplikace
   Command centrum umožňuje dva druhy vydání
   - Osobní, kde pouze vy můžete poslat licenci k aplikaci skrze vývojářský interface
   - Veřejná, aplikace je veřejně viditelná a každý si ji může v databázi úkolů vyhledat a stáhnout

[9]



Obrázek 8 diagram cyklu vydání aplikace na command centru [9]

3.2 Nastavení aplikace pro vydání

Před samotným vydáním aplikace do command centra je třeba, aby aplikace splňovala několik podmínek. Aplikace musí být určena pouze pro tablet robota Pepper. Pro vytvoření aplikace pouze pro tablet Peppera je třeba upravit soubor pro stavbu aplikace „build.gradle“. Tato změna změní aplikaci při sestavení a bude podporovat pouze ABI tabletu Peppera.

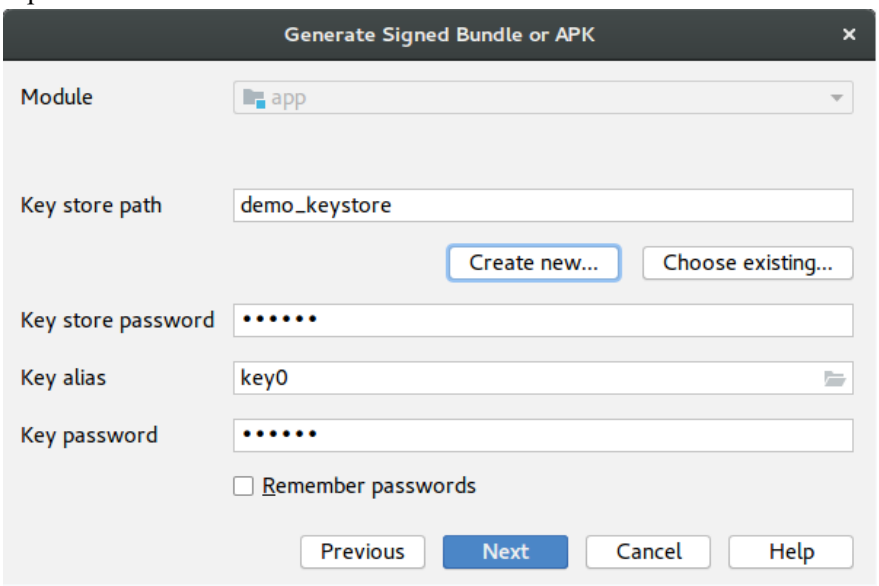
Do souboru pro vystavbu aplikace je třeba přidat následující kód:

```
splits {
   // Configures multiple APKs based on ABI.
   abi {
       enable true
       reset()
       include "x86", "armeabi-v7a"
       universalApk false
   }
}
```

Obrázek 9 kód pro upravení build souboru pro podporu tabletu Pepper [9]

Po upravení kódu je třeba použít znovu příkaz pro vytvoření projektu. To zapříčiní tvorbu dvou APKs, kde jedna bude vytvořena pouze pro Peppera. Do command centra je povoleno nahrát pouze verzi pro robot Pepper. APK pro Pepper tablet je „armeabi-v7a“ .

Druhou podmínkou pro vydání aplikace do command centra je podpis. Toto slouží k odstranění bezpečnostních rizik. Podpisu aplikace lze dosáhnout b menu výstavby „build“ kliknutím na „vytvořit podepsanou skupinu / APK“. Následně se vybere možnost „APK“. V následujícím okně se zadávají informace o podpisu a klíči.



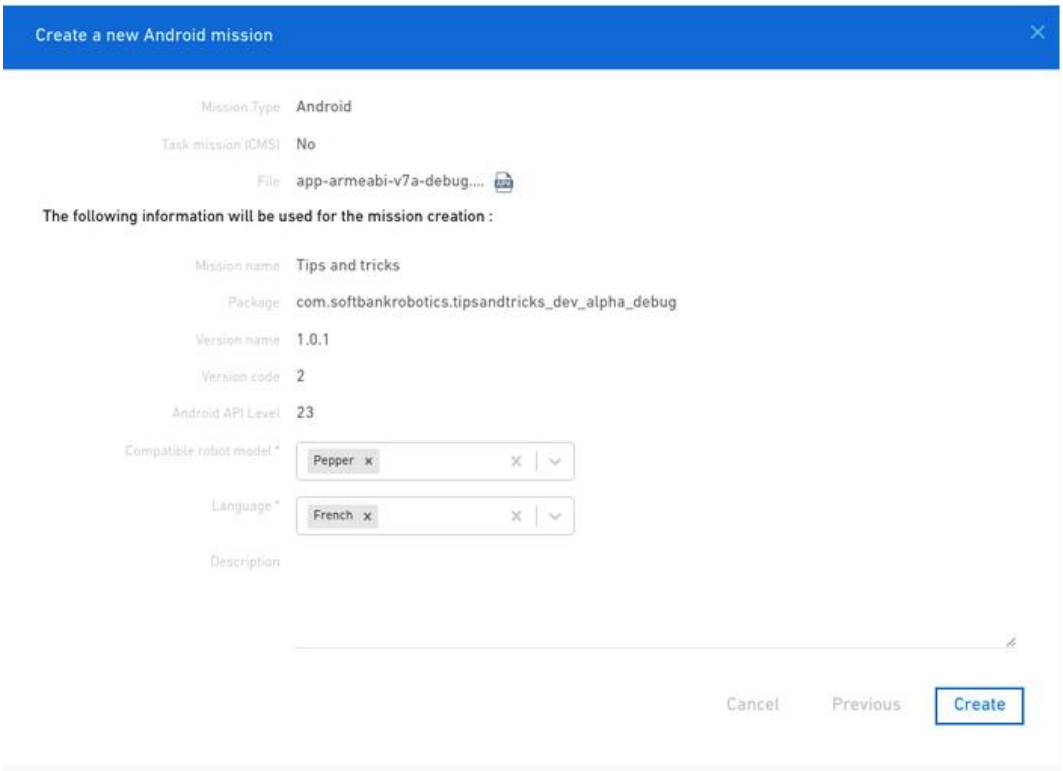
Obrázek 10 okno pro tvorbu klíče [9]

Další okno slouží pro zvolení verze pro vydání, kde zvolte „release“ a potvrďte tlačítkem dokončit. Následně by se mělo zobrazit upozornění oznamující úspěšné sestavení v pravém dolním rohu IDE.

[9]

3.3 Nahrání aplikace do command centra a následně do robota

Nahrání aplikace do command centra je jednoduché a intuitivní. V rozhraní pro vývojáře se zaklikne tvorba nového úkolu, kde se zadá platforma a typ úkolu (android a no task). Následně se otevře okno, kde se pomocí drag and drop přesune váš aplikační soubor. V následujícím okně pak vyjde výčet vlastností a parametrů úkolu.



Obrázek 11 parametry při vydání aplikace [9]

Nahrání aplikace do robota lze buď přímo, kde se pouze vybere úkol v lokální databázi nebo nepřímo na jiném účtu. Vyhledá se aplikace v databázi úkolů a pokud vám majitel daroval licenci, lze ji stáhnout. Následně se standardně přidělí robotu kliknutím na moji roboti, kde se zvolí přidat úkoly a úlohy, najde stažený úkol a potvrdí. Lze úkol označit jako hlavní, což zapříčiní jeho spuštění po zapnutí robota.

[9]

4 Tvorba Dostupných interaktivních úloh pro robot Pepper

4.1 Obecný návrh aplikace

Tato kapitola se zabývá doporučeným postupem při tvorbě projektu pro robot Pepper. Postup vychází z praktických zkušeností vývojového týmu Softbank Robotics a snaží se předejít častým chybám a neošetřeným stavům, do kterých se robot může dostat. Jedná se o postup doporučený a obecně preferovaný, ovšem nikoliv jediný.

Návrh projektu pro robot Pepper vychází z obecného návrhu projektu, tedy základní kroky jsou stejné.

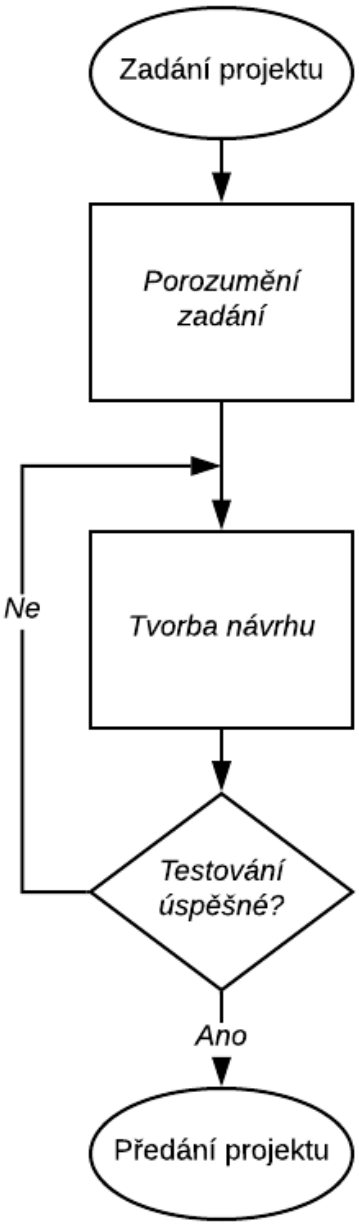
1) Porozumění zadání klienta
2) Tvorba prvního designu
3) Testování funkčnosti

Při návrhu modelu je důležité být pragmatický a objektivní. Vycházet z podložených a nejlépe vyzkoušených fakt a průběžně model testovat.

Kromě obecných problémů je třeba dávat pozor i na specifické problémy spojené s robotem. Tyto jsou základní pro každý projekt robota Pepper:

1) Fyzické možnosti robota
   Pepper se nedokáže pohybovat stejně rychle jako člověk, navíc jeho pohyb nabývá omezení při pohybu mezi překážkami kvůli pohybu pomocí kol nezvládne překonat strmé překážky.
2) Interakce mezi člověkem a robotem
   Ne všichni lidé se budou chovat k robotu podle očekávání. Může se například stát, že se lidé budou stydět interagovat s robotem před ostatními lidmi. Také se může stát, že si s robotem začne hrát dítě, což většinou vyústí v nemálo doteku fyzických interakcí (ohýbání prstu, doteků hlavy atd.). Dalším častým případem je výskyt robota v zalidněném místě například prodejní hale. Obzvláště při rušných hodinách může docházet ke kolizím mezi lidmi a robotem.

[10]



Obrázek 12 Vývojový diagram obecného postupu při řešení projektu

3) Umístění robota
   Nutností je zajistit robotu přístup k nabíjení. Pepper vydrží pracovat bez nabíjení přibližně 11 hodin, pak se musí připojit opět na nabíjecí stanici. Je třeba vhodně umístit nabíjecí stanici a zajistit do ní přívod eklektické energie. Zkontrolujte přívod energie jak v denních, tak nočních hodinách. Vyvarujte se použití skládacího nábytku v přítomnosti robota, jelikož se dá zvednout a senzory robota jej nemusí zaregistrovat. Stal by se pak pro robota neviditelnou překážkou. Doporučuje se omezit možnosti vzniku hlučného prostředí, které by mělo za následek znemožnění použití hlasového ovládání robota a vyvarujte se také kolizím robota s pohyblivými objekty například dveřmi.
4) Způsob tvorby aplikace pro robota
   Při ukládání uživatelských dat je třeba dodržovat právní nařízení a legislativu, zajistit ochranu osobních údajů (GDPR). Při tvorbě aplikace je několik možností postupu a nejsou všechny stejně hardwarově účinné. Použití Naoqi nebo Qisdk? Předělat již hotovou aplikaci nebo začít od nuly? Dále je dobré si vytvořit mapu, kde se robot bude pohybovat a přidat do ní možné překážky. Toto vám pomůže předejít kolizím a dalším problémům s pohybem a ovládáním robota.

[10]

Modelování

Modelování je proces, kde se programátor snaží vytvořit prototyp aplikace zadané zákazníkem tak, aby neobsahovala žádné programové chyby (bugy). Většinou dochází v projektu k několikanásobnému modelování, jelikož obvykle bezchybná (zákazníkem odsouhlasená) aplikace nevzniká z prvního modelu.

V aplikaci pro robota Pepper se doporučuje při modelaci zaměřit na následující body.

1) Zvolení účinné metody
   Jak již bylo zmíněno, aplikace se dá vytvořit několika způsoby, ale je důležité zvolit efektivní způsob. Robot má omezený hardware, a proto se tedy softwarové nedostatky nedají nahrazovat upgradem hardwaru. Proto se volí iterační postup. Jedná se o postupné nabalování funkcí s okamžitým testováním. Začíná se takzvanými core funkcemi, které slouží jako základ a jsou jádrem celé aplikace, proto core. Testování probíhá vždy při přidání samostatného funkčního bloku do aplikace, kterým může být jedna funkce nebo několik na sobě závislých funkcí. Tato metoda, ač se může zdát pomalá, pomáhá tím, že programátor ví, kde se chyba nachází a snadno ji může opravit. Zároveň umožnuje v případě komplikované chyby zpětně zjednodušovat program a najít poslední bezchybný stav. Dalším doporučením je komunikovat s ostatními programátory a odborníky. Velice často se stává, že programátor vynaloží spoustu úsilí a vytvoří aplikaci, která je volně dostupná. Při programování je také důležitá disciplína, tvorba rozvrhu a dodržení jej zajistí stabilní progres v modelování.

[10]

2) Zvážit okolnosti zákazníka
   Zákazník většinou nebývá tak zkušený ve vytváření aplikace pro robota jako vy a některých problémů si nemusí všimnout. Promyslete okolnosti, za kterých bude robot použit a možné problémy konzultujte se zákazníkem a navrhněte možná řešení. Například robot může mít za úkol zkontrolovat zákazníka v obchodě a přiřadit jemu určené zboží ze skříně ve skladu. Pokud zákazník zvolí levné skříně, které budou plně manuální, je třeba vyřešit, jak se k nim robot dostane, což může na konec stát více než koupě inteligentních skříní, které by mohly komunikovat s robotem přes wifi. Toto je dobré zákazníkovi objasnit a poradit se o dalším přístupu. Výhodou je také vědět předem rozpočet, množství prostoru a například dostupnost sítě wifi.

[10]

Prototypizace

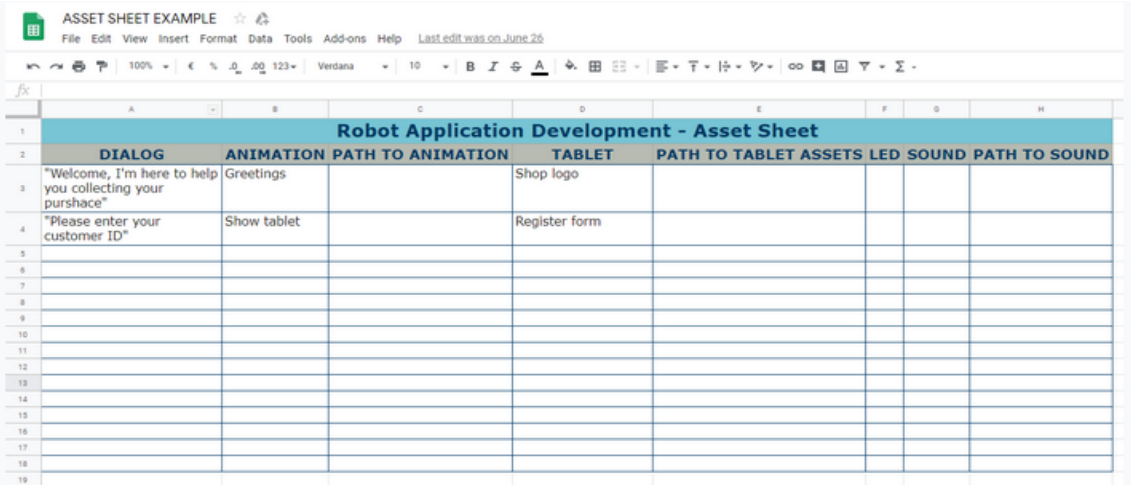
Prototyp slouží jako základ pro následující model. Prototypizace dává návrháři příležitost zkoumat a testovat nové funkce ve zjednodušených podmínkách. Prototypem může být například zjednodušený model, model robota v simulátoru nebo třeba pouze jednoduchá náhrada jako papírová deska s nákresem. Prototyp pomáhá v adaptaci teoretického modelu do praxe. Může zvýraznit problémy s přepravou, překážkami nebo jinými nepříjemnostmi. Například v prodejně při kontrole věrnostních karet robotu nastavíte čas na zákazníka. Tento časovač se bude ale zapínat několikrát během pouhého jednoho zákazníka, protože doba předložení dárkové karty od zákazníka je odlišná. Někteří ji nosí v batohu, peněžence, kabelce a jiní ji mají již připravenou. Tuto chybu ovšem nezjistíte, dokud aplikaci nevyzkoušíte, a proto se tedy testuje nejdříve prototyp ve zjednodušených podmínkách. Dobrým zvykem je zaznamenávat historii jednotlivých prototypů a při vydání aplikace záznamy porovnat pro další aplikace.

[10]

Organizace aplikace

Může být složité zkoordinovat pohyb, mimiku a přístup robota k člověku. Doporučuje je se použití asset sheet. Jedná se o tabulku obsahující jednotlivé funkce robota, jejich programové složení, vlastnosti a charakteristiku. V případě, že se při modelaci něco změní, tak při zápisu do tabulky jsou viďet další věci, které je třeba také změnit, protože jsou závisle nebo se mají chovat podobně jako změněná funkce.

[10]



ASSET SHEET EXAMPLE

File Edit View Insert Format Data Tools Add-ons Help Last edit was on June 26

Robot Application Development - Asset Sheet							
DIALOG	ANIMATION	PATH TO ANIMATION	TABLET	PATH TO TABLET ASSETS	LED	SOUND	PATH TO SOUND
"Welcome, I'm here to help you collecting your purshace"	Greetings		Shop logo				
"Please enter your customer ID"	Show tablet		Register form				

Obrázek 13 Příklad tabulky assetů [10]

Testování aplikace

Pro docílení jednoduchého testování je dobré aplikaci rozdělit na robotickou a ne robotickou část. Pro pohyb v kódu využívejte ladících příkazů a zlomových bodů (break points). Tímto testováním jednotlivých částí kódu můžete ušetřit spoustu času, ovšem před vydáním konečné aplikace celkový test proveďte.

Mějte na paměti, že robot pracuje ve statických podmínkách. Běžné prostředí se neustále mění a vyvíjí, proto je potřeba robota v průběhu změn aktualizovat, jinak vzniknou nechtěné chyby, které v původních podmínkách nastat nemohly.

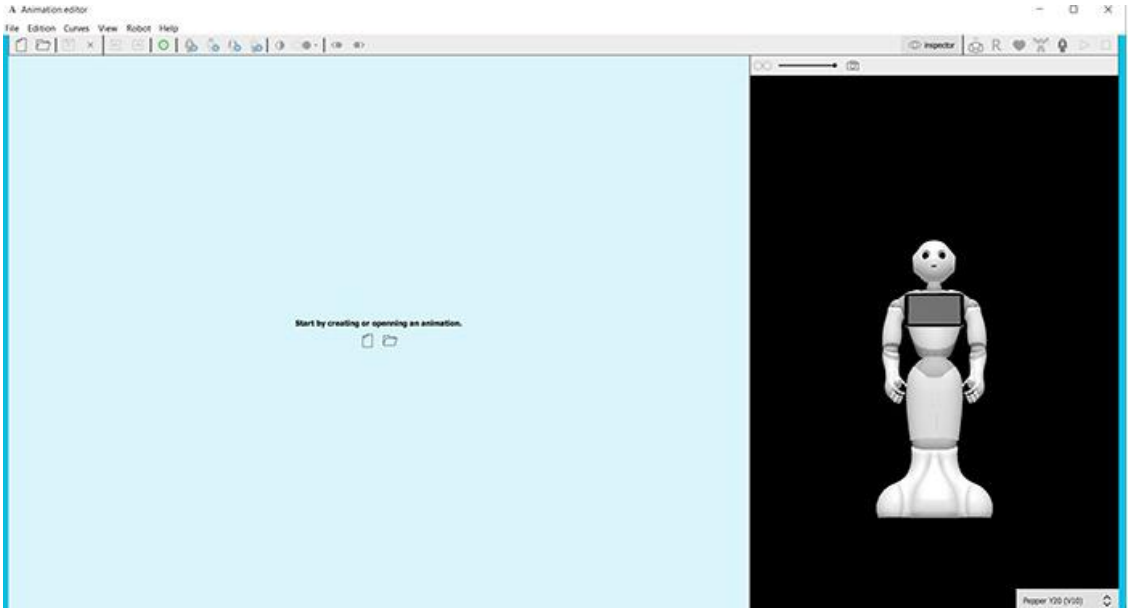
[10]

4.2 Tvorba animace

Pro tvorbu animací robota Pepper je dostupný editor, který se nainstaluje spolu s SDK přímo do Android studia. Využití animací při aplikaci robota je opravdu hodně například ukázání okolí, článků, ilustrací, promovaného výrobku nebo reakcí na lidi, takže animace rozhodně využije robot Pepper, který je určený k promování, prodávání, bavení, napovídání, průzkumu dat a poskytování služeb.

Hlavní okno editoru se skládá ze tří hlavních částí

1) Hlavní lišta
2) Velký světle modrý čtverec umožňující práci se soubory
3) 3D náhled na simulovaného/fyzického robota

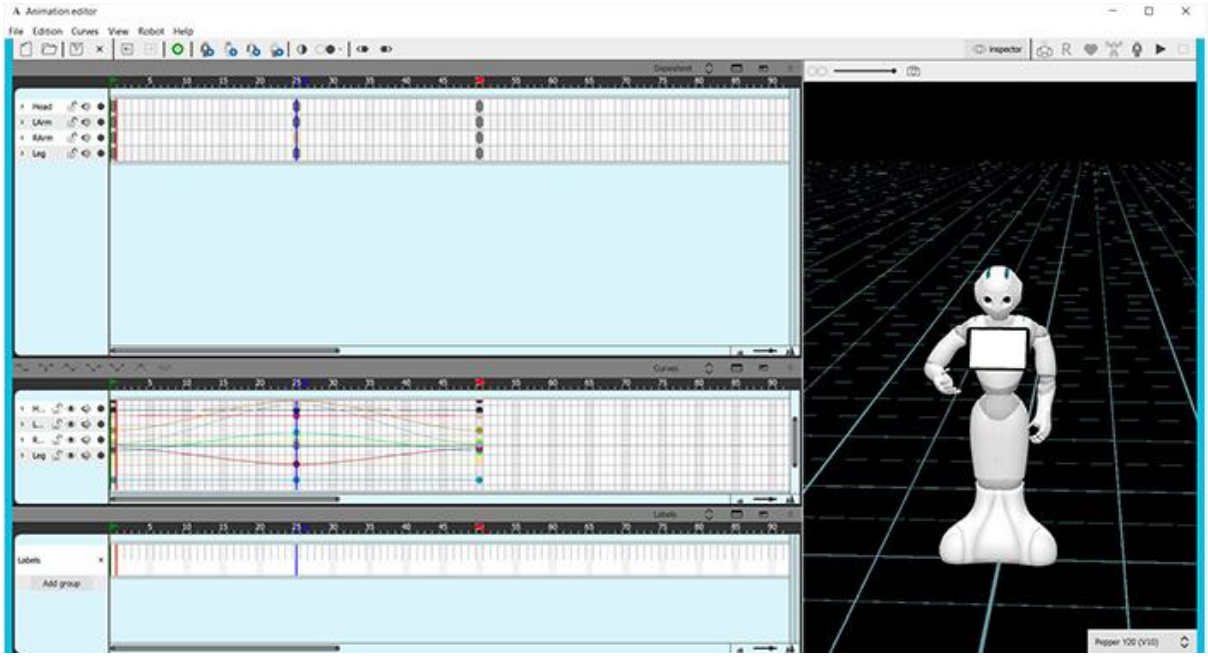


Obrázek 14 Animační editor pro robot Pepper [11]

Kliknutím na ikonu vlevo ve středu modrého čtverce vytvoříte prázdný soubor. Tento soubor je typu „qianim“, který se při nahrání do robotu přepočítá do pohybových příkazů.

Po otevření nového souboru se v modrém čtverci zobraz časová osa pro jednotlivé části robota. Na této ose se dají plánovat jednotlivé pohyby animace. V horním pásku je funkce označená jako rozdělit a přidat oblouky. Ta rozdělí základní části robota na jednotlivé dílčí pohyblivé části a odemyká plné možnosti animace robota. Kliknutím do časové osy přidáte pozici, do které se daná část robota musí dostat. Přechod mezi jednotlivými pozicemi je automaticky dopočítáván podle algoritmu, ten se dá zvolit z několika možností. Pro detailní manipulaci s robotem lze animaci tvořit přímo na 3D modelu a přídavné liště nad ním.

[11]



Obrázek 15 tvorba animace v editoru pro robot Pepper [11]

Při tvorbě animace je dobré pamatovat na tři věci.

1) Metodika
   Systematický postup podle určitého plánu dodá animaci přirozenost a předchází chybám, kdy animace vypadá „kostrbatě“.
2) Rozložení
   Tvorba animace se skládá z modelování několika póz a později se tyto pózy spojí. Není dobré řešit přechod mezi prvními dvěma pózami, když jej může další póza ovlivnit.
3) Časování
   Dobré načasování je základ. Při podání ruky robot nemá třepat rukou při jejím natahování, ale až po.

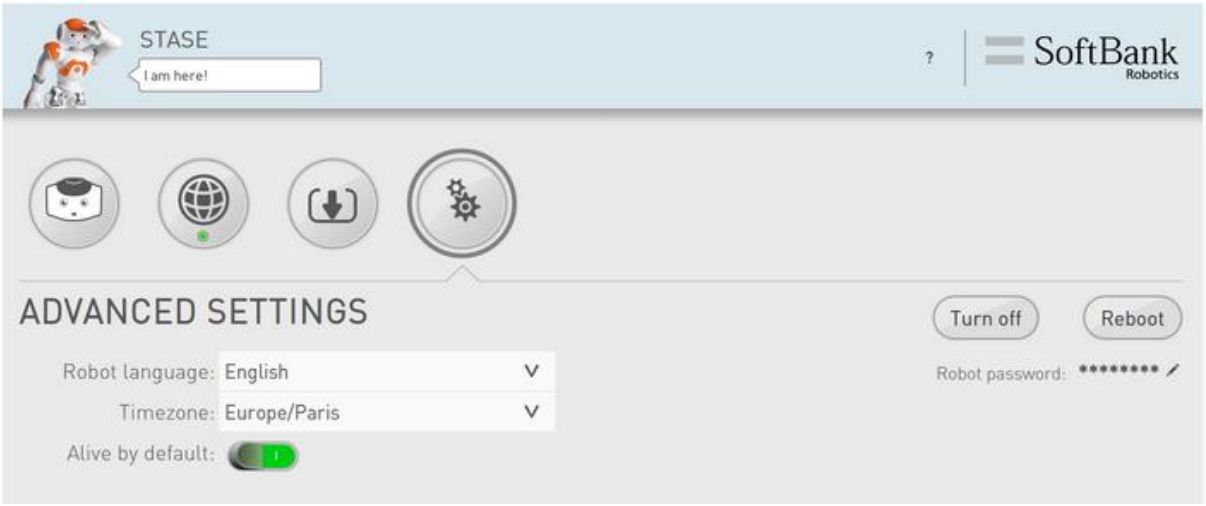
[11]

4.3 Řeč

Pepperův hlas je automaticky generovaný ze psaného textu za použití text to speech enginu založeného na staženém jazykovém balíčku. Tento balíček obsahuje databázi nahrané řeči a pravidel, podle kterých určuje výslovnost. Těmto útržkům řeči se říká phonemeny a slouží k vytvoření jednotlivých a veškerých slov daného jazyka. Nahrávky byly následně filtrovány, aby robotův hlas zněl více robotičtěji a nedošlo k záměně s člověkem.

Webová stránka robota

Nejjednodušším způsobem, jak nechat robota promluvit je přes jeho webovou stránku. Na tu se lze dostat zadáním IP adresy robota do vyhledavače, kde adresu můžete zjistit stisknutím hlavního tlačítka na hrudi robota. Po zadání přihlašovacích údajů se zobrazí stránka poskytující základní informace, jako například nabití baterie, nastavenou hlasitost, jazyk a nastavené časové pásmo. V levém horním rohu této stránky je okno pro příkaz řeči robota.



Obrázek 16 webová stránka robota

Řeč v android studiu

Běžně se řeč robota programuje v Android studiu, nikoliv přes webovou stránku, ta slouží pouze pro testování a jednoduchou ukázku. Android studio představuje více možností v oblasti řeči a základ je poměrně jednoduše programovatelný.

Příklad kódu pro řeč robota:

```
////////////////////////////
//        helpers         //
////////////////////////////

private fun makeSay(text : String) : Say {
   return SayBuilder.with(qiContext)
           .withText(text)
           .build()
}

//////////////////////////
// Presentation logic //
//////////////////////////


private fun runPresentation() {

   // Part 1: "Making me talk..."

   makeSay("Okay, so ... making me talk is a first step a bit like ...").run()
   Thread.sleep(200)
   makeSay("a rolling rock ...").run()

}
```

Obrázek 17 Příklad kódu pro řeč robota [12]

Jak je na obrázku vidět, základní programování řeči je poměrně jednoduché. Ukázka začíná tvorbou funkce pro mluvení, kterou následně volá a vkládá do ní řeč ve formě stringů. Příkaz Thread.sleep(200) dodává pauzu mezi vlákny, takže fráze řeči nezačínají hned po sobě.

[12]

Úprava výslovnosti

Jak již bylo řečeno, databáze fenomonů je od vývojářů uzamčena, proto jsou jediné dostupné úpravy pomocí speciálních znaků jako jsou například „ . “ nebo „ , “.

Pauza

Použitím pauzy se mezi slovy vytvoří časové zpoždění, je to velice podobné použití oddělení čárkou v běžném jazyce a proto speciální znak pro pauzu je „ , “. Použití pauz je velmi důležité pro vytvoření plynulé a srozumitelné mluvy. Nedoporučuje se při potřebě prodloužit dobu pauzy napsat za sebe více znaků pauzy, ale přenastavit jejich dobu. Toho lze docílit pomocí následujícího příkazu.

```
"\pau=value\"
```

Kde maximální hodnoty pauzy je 30 sekund.

Hláskování a výslovnost

Pokud dochází ke špatné výslovnosti, nejčastěji u cizích slov, základním trikem je použití takzvaného nuceného hláskování. Jedná se o nahrazení zápisu slova přepisem jeho výslovnosti, například místo „apple“ by se psalo „epl“. Nevýhoda použití tohoto nuceného hláskování je, že se nesmí zobrazovat na tabletu pomocí textu.

Pozn. Při použití japonských znaků (hiragana, katakana a kanji) může z důvodů několikanásobné výslovnosti znaků docházet při použití nuceného hláskování k chybám a může být třeba kombinovat znaky z jednotlivých písem.

Pomocí příkazu, lze přenastavit výslovnost jednotlivých frází.

```
"s:(Vystup) ^replace(Vyraz1, Vyraz2, Frekvence)"
```

Kde výraz jedna je psaná forma a výraz dvě je jeho alternativní výslovnost pomocí nuceného hláskování. Frekvence určuje zlomek záměn, 0 … žádná, 1 … všechny výskyty se nahradí.

Další příkazy pro úpravu výslovnosti

Nastavení hlasitosti mluvy:

```
"\vol=hodnota\"
```

0 – 100, v základním nastavení je nastaveno na 80.

[12][13]

Nastavení rychlosti mluvy:

`"\rspd=hodnota\"`

`50 – 400, kde v základním nastavení je nastaveno na 100.`

Nastavení výšky hlasu

`"\vct=hodnota\"`
50 – 200, kde v základu je nastaveno 100.

Reset parametrů úpravy mluvy

`"\rst\"`

Změna důrazu

`"\emph=hodnota\"`

0: snížený důraz

1: naléhavý

2: s přízvukem

Nastavení zvukových hranic

`"\bound=hodnota\"`

W: slabý přechod

S: výrazný přechod

N: bez přechodu

Zapnutí přestávek mezi větami

`\eos=hodnota\`

1: zapnuto

0: vypnuto

[12][13]

Výrazové příkazy (Angličtina, Francouzština, Čínština)

`\style=neutral\` základní hlas

`\style=didactic\` vhodnější pro vyprávění

`\style=joyful\` robot zní více natčený

Fonetické psaní

Jedná se o alternativu k nucenému hláskování. Využití najde ve slovech, která se stejně píšou a jinak vyslovují (hlavně v angličtině). Příkladem může být anglické slovíčko „resign“, které běžně znamená odstoupit nebo odejít, můžeme ale chtít „resign“ použít jako spojení „re“ + „sign“, což znamená znovu zapsat nebo opětovně nastoupit. Zápis těchto významů je totožný, a proto dochází k jedinému rozdílu ve výslovnosti (rɪˈzaɪn/riːˈsaɪn). Fonetické psaní používá 2 tagy pro oddělení od normálního textu. \toi=lhp\ pro začátek a \toi=orth\ jako ukončení. Mezi tyto tagy se napíše nová výslovnost slova.

[12][13]

4.4 QiChat, chatbot pro robot Pepper

QiChatbot přidává robotu Pepper možnost odpovídat na otázky lidí v jeho okolí. Jedná se o poměrně složitou aplikaci a proto se doporučuje dodržovat základní strukturu robotí aplikace. V této struktuře hlavní třída MainActivity obsahuje minimálně tyto funkce:

```
class MainActivity : RobotActivity(), RobotLifecycleCallbacks {
    override fun onRobotFocusGained(qiContext: QiContext) {
        // TODO
    }

    override fun onRobotFocusLost() {
        Log.i(BOOKMARK, "Focus lost")
    }

    override fun onRobotFocusRefused(reason: String?) {
        Log.i(BOOKMARK, "Focus refused because $reason")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        QiSDK.register(this, this)
    }

    override fun onDestroy() {
        super.onDestroy()
        QiSDK.unregister(this, this)
    }
}
```

Obrázek 18 základní struktura robotí aplikace [14]

Toto jsou základní funkce pro většinu aplikací pro robota, které reagují na jeho základní chování. První tři funkce se zabývají soustředěním robota, to nastane například pokud robot rozpozná obličej, dojde k soustředění a robot se bude snažit na obličej dívat. Focus gained se tedy spustí, pokud robot obličej najde, focus lost pokud obličej ztratí a focus refused, když obličej najde, ale nemůže na něm držet pozornost, protože má přidělenou akci s vyšší prioritou. Funkce on create se zavolá při vytvoření objektu a on destroy při jeho smazání.

[14]

Tvorba téma

Pro téma je dobré vytvořit samostaný soubor v záložce soubor, nový a téma konverzace. Toto téma je třeba aktivovat vytvořením tří objektů téma (jako objekt kódu), qiChatbot a chat.

```
class MainActivity : RobotActivity(), RobotLifecycleCallbacks {
    override fun onRobotFocusGained(qiContext: QiContext) {
        val topic = TopicBuilder.with(qiContext).withResource(R.raw.basics).build()
        val qiChatbot = QiChatbotBuilder.with(qiContext).withTopic(topic).build()
        val chat = ChatBuilder.with(qiContext).withChatbot(qiChatbot).build()
        // Start the dialogue
        chat.run()
    }
}
```

Obrázek 19 kód pro aktivaci souboru s tématem [14]

Dialogy

Nyní je možné začít psát dialogy. Ty se píší v souboru s koncovkou „.top“. V tomto souboru je nutno zachovat hlavičku, jinak nedojde ke spojení se souborem tématu. Základní dialog se píše stylem, kdy za „u:“ se do kulatých závorek napíše fráze, kterou má robot rozpoznat a za ně odpověď robota.

V dialozích existují syntaxe pro zvýšení variability a pokrytí konverzace. Prvním příkladem jsou synonyma, která se udávají v hranatých závorkách „[ ]“. Pokud robot rozpozná alespoň jedno slovo ze všech synonym v hranatých závorkách, vyhodnotí dialog a odpoví. Toto se tedy používá, pokud robot může reagovat na více vstupních frází jednou dialogovou podmínkou, takže například pozdrav. Tady ovšem vzniká problém při frázích o více slovech, například: u:[Ahoj dobrý den]. Při tomto zápisu by robot reagoval na tři jednotlivá vstupní slova „ahoj“, „dobrý“ a „den“. To je ale v tomto případě nechtěné, jelikož chceme, aby robot reagoval na frázi „dobrý den“ nikoliv na jednotlivá slova. Tento problém se řeší syntaxí fráze. Ta je uvedena v anglických uvozovkách “ “ a tedy správná syntaxe fráze by byla: “dobrý den“.

Dalším je takzvaný doplněk. To je slovo, které ve větě být může nebo nemusí. Většinou doplňkem bývá výplňové slovo (jako, prostě, atd…) . Doplněk tedy slouží jako ochrana proti těmto výplňovým slovům, které na význam věty nemají vliv. Píše se do složených závorek „{}“.

Mezi základní syntaxe patří také koncept. Funguje jako zástupce na ploše počítače. Jedná se o předem definovanou frázi, které je přiděleno jméno. To lze následně používat v dialozích místo psaní této fráze znovu. Koncept dokáže ušetřit spoustu času hlavně v aplikacích se složitými dialogy.

[14]

Pamatování frází

K tomuto účelu v chatbotu slouží proměnné. Proměnná se zapisuje znakem dolaru a číslem určujícím konkrétní proměnnou, takže například „$3“. Pamatování frází se může hodit, pokud chceme referovat v konverzaci na informaci, kterou jsme získali v předchozích dialozích.

Příklad:

```
concept:(nazev)
["první nazev" "druhy nazev"]

u:(Jmenuji se _~nazev)

Rád tě poznávám $1 !
```

V příkladu je vytvořen seznam, který volá funkce „nazev“. Před voláním té funkce je použit znak „_“, který slouží pro zapamatování hodnoty do první dostupné dočastné proměnné. Robot následně odpovídá dialogem, ve kterém volá proměnnou, do které je uložen první výskyt jména uvedeného současně v dialogu i v seznamu. Pokud jméno nebude nalezeno v seznamu, robot odpoví pouze „Rád tě poznávám!“.

[14]

Instrukce divoká karta

V použití s catch („_“) slouží k zapamatování jakéhokoliv slova. Divoká karta se označuje znakem „*“ a zachytí jakoukoliv frázi v dialogu a zapíše jej v podobě, jak mu robot rozuměl. Jednoduchý příklad může nahradit, zjednodušit a případně vylepšit předchozí kód.

```
u:(Jmenuji se _*)
Rád tě poznávám $1 !
```

Následující kód zachytí libovolnou frázi, která následuje po „Jmenuji se“. Následně frázi vrátí v dalším dialogu pomocí dočasné proměnné. Takto zadaný dialog, má ovšem nevýhodu v tom, že nedochází ke kontrole vstupu a může se stát, že robot bude špatně rozumět nebo někdo bude hlásit špatné jméno.

[14]

Dialogová funkce random

V určitých případech můžeme chtít, aby robot vybral odpověď z nějakého seznamu. K tomu lze použít algoritmus nebo třeba funkci random, která pseudonáhodně vybere odpověď ze seznamu. Značí se „^rand[]“, kde je v hranatých závorkách zapsán seznam výstupů.

[14]

Funkce first

Jedním z výběrových algoritmů je funkce first. Ta má zá úkol vybrat první odpověď splňující podmínky. Podmínka se zapisuje stylem: $proměnná==hodnota a může použít operátory „==“, „<“, „>“ a „<>“ pro operace rovnosti, větší, menší a nerovnosti.

```
u:(Co o mě víš?)
^first[
        "$asked_joke==1 Jmenuješ se $name a rád vtipkuješ!"
        "Jmenuješ se $name !"
        "Promiň, vůbec tě neznám."
]
```

V tomto případě vložení proměnné do výstupu funguje jako podmínka. Uvedený příklad tedy odpovídá prvním dialogem pokud proměnná „asked_joke“ je rovna jedné a zároveň proměnná „name“ má hodnotu. Pokud „asked_joke“ není rovna jedné, tak se použije druhý dialog a pouze pokud ani jeden z předchozích případů nelze vyhodnotit (není určená proměnná „name“) tak odpovědí bude dialog třetí. Třetí dialog zde funguje jako takzvaný základní výstup.

[14]

Dialogový strom

Uvedené příkazy jsou dostatečné k vytvoření jednoduchých dialogů. Pro komplikovanější konverzaci se využívá dialogový strom. Jde o mapu, kde se podle definovaných podmínek a priorit posouvá ukazatel a spouští následující dialog.

V dialogovém stromu nezáleží na pořadí zápisu dialogů, který dialog se spustí je čistě řízeno pomoci pravidel a podpravidel.

```
u:(Nudím se)
Je libo jazykolam?
        u1:(ano)
        super!
        u1:(ne)
        Nevadí!
```

Víše uvedená část dialogu, je správně řešená otázka o potvrzení. Nejdříve robot čeká na vstup dialogu u a až potom může reagovat na u1. Takže robot nebude odpovídat na vstup „ano“, dokud nezaznamená vstup, „Nudím se“. Toto by platilo i v případě, že by dialog u1 byl napsán v kódu nad dialogem u. Podpravidla takto definují rámec dědičnosti dialogů. Jednoduše, dokud nebude proveden rodičovský dialog (např. u je rodič u1 a u3 je rodič u4), tak je potomek dialogu zakázán a nemůže být proveden.

[14]

Pravidla, podpravidla dialogů a jejich priority

Pravidlem se myslí dialog na obecné úrovni, většinou značen pouze písmenem bez čísla. Podpravidlo je jeho potomek a každý další potomek. Podpravidla mají přednost při vykonávání z dodržení tématu konverzace a zamezení náhodných nesmyslných odpovědí. Takže pokud se splní podmínka pravidla a další dialog bude mít vyhodnocenou podmínku pravidla a podpravidla dědičného z dříve aktivovaného dialogu, robot bude reagovat pomocí dědičného podpravidla. Tato priorita podpravidla zůstává pouze na jeden následující dialog, pokud je tedy následující dialog nějakým způsobem náročný, například jazykolam, a uživateli se nepovede jej říct napoprvé, je třeba spustit znovu počátečního předka a postupně se dostat opět do dialogu s jazykolamem. Tento problém může řešit funkce „stayInScope“, umožňující po dokončení dialogu zastavit ukazatel na stejné rámcové úrovni.

```
u:(Nudím se)
Je libo jazykolam?
        u1:(ano)
        Skvěle! Opakuj po mě: text jazykolamu.
                u2:(text jazykolamu)
        Wow, jde ti to skvěle!
        u2:(Vzdávám to)
                Škoda, příště to půjde lépe!
        u2:(*)
       ["Nějak se mi to nezdá, zkus to znovu! ^stayInScope" "Pořád to není
      ono, ještě jeden pokus! ^stayInScope" "Příště to zvládneš!"]
        u1:(ne)
        Nevadí!
```

Výše uvedený kód umožňuje 2 opravné pokusy. Uživatel také může jazykolam vzdát a rovnou se přesunout znovu na obecnou úroveň rámce.

[14]

Návrhy, záložky a funkce „goto“

Návrh slouží jako pravidlo bez vstupních dat.

```
u:(Nudím se)
Je libo jazykolam?
        u1:(ano)
        Skvěle! ^enableThenGoto(TONGUE_TWISTER)
        u1:(ne)
        Nevadí!


proposal: %TONGUE_TWISTER
Opakuj po mě: text jazykolamu.
        u1:(text jazykolamu)
        Wow, vedeš si skvěle!
        u1:(Vzdávám to)
        Příště to půjde!
        u1:(*)
        ["Nějak se mi to nezdá, zkus to znovu! ^stayInScope" "Pořád to není
      ono, ještě jeden pokus! ^stayInScope" "Příště to zvládneš!"]
```

Pomocí návrhu došlo k rozdělení do 2 dialogových bloků. Funkčnost je pořád stejná, začíná globální pravidlo s jeho dědičnými potomky. Pokud dojde k potvrzení jazykolamu, dojde ke spuštění funkce „goto“, která přeskočí na její záložku (návrh s jazykolamem) . Tento kód je ekvivalentní k předchozímu příkladu s jazykolamem, pouze je čitelnější a lépe se s ním pracuje, jelikož záložku lze volat i z jiné části kódu.

[14]

Proaktivní dialog

Dialogy, které jsou zmíněné víše, jsou reaktivního typu. Vždy robot reaguje na nějaký podmět (většinou ve formě hlasového vstupu). Proaktivní dialog je ten, který se spouští, bez reakce na vnější podmět. Jako příklad lze uvést, že robot začíná konverzaci při dlouhém tichu. Tento typ dialogu lze sestavit z návrhu a záložek.

```
proposal: %nazev

//proaktivní dialog
```

Na první pohled dialog vypadá dosti podobně jako ostatní, ovšem je zřejmé že mu chybí vstupní podmínka. Tento dialog by se v tomto stavu nikdy nespustil, a proto je jej třeba volat přímo z kódu.

[15]

Pro volaní takového dialogu je dobré vytvořit pomocnou metodu.

```
private fun goToBookmark(bookmarkName : String) {
    qiChatbot.goToBookmark (
        topic.bookmarks[bookmarkName],
        AutonomousReactionImportance.HIGH,
        AutonomousReactionValidity.IMMEDIATE)
}
```

Obrázek 20 metoda volání dialogu [15]

Tato metoda kromě názvu záložky pracuje s dalšími dvěma argumenty. Důležitostí a validitou. Tyto parametry určují chování robotu, pokud robot již mluví. Jeli důležitost vyšší než důležitost právě probíhajícího dialogu, pak se dialog přeruší a robot začne mluvit ten s vyšší prioritou. Validita určuje, jestli má cenu dialog říct. Pokud je validita nastavena na hodnotu „delayable“, robot přehraje dialog po dokončení právě probíhajícího dialogu. Ovšem pokud je nastavena na hodnotu „immediate“, tak se dialog spustí pouze pokud jej lze spustit ihned.

Tuto metodu následně voláme, když chceme dialog spustit. Pro jednoduchost je na příkladu metoda volána po spuštění akce

[15]

```
override fun onRobotFocusGained(qiContext: QiContext) {
    // ...
    chat.addOnStartedListener {  goToBookmark("MULTIPLY_CHALLENGE") }
    // Start the dialogue
    chat.run()
}
```

Obrázek 21 volání metody po startu akce [15]

Použití proměnných v dialogu

Proměnné i pro dialog se definují v kódu aplikace. Nejprve se definuje obecně, stejně jako každá jiná a přiřadí se jí hodnota. Následně je však třeba vložit hodnotu do obrazu proměnné v dialogu (má před sebou znak dolaru „$“) .

[15]

```
private fun randomize() {
    val numberA = Random.nextInt(3, 9)
    val numberB = Random.nextInt(3, 9)
    val multiplicationResult = numberA * numberB

    qiChatbot.variable("numberA").value = "$numberA"
    qiChatbot.variable("numberB").value = "$numberB"
    qiChatbot.variable("result").value = "$multiplicationResult"
}
```

Obrázek 22 tvorba proměnné pro dialog [15]

Předchozí příklad přiřazuje proměnným „numberA“ a „numberB“ náhodnou hodnotu od 3 do 9 a proměnné „multiplicationResult“ výsledek po jejich vynásobení. Potom jejich hodnoty vkládá do jejich obrazů v dialogu pojmenovaných jako „$numberA, $numberB, $Result“. Pojmenování obrazů a proměnných nemusí být nutně schodné, ovšem je to přehlednější.

Obrazy v dialogu využíváme jako náhradu proměnných.

```
proposal: %MULTIPLY_CHALLENGE
Okay, what's, $numberA times, $numberB ?
    u1:($result)
    Right!
    u1:(*)
    Wrong!
```

Obrázek 23 dialog využívající obraz proměnné [15]

Může se stát, že chceme odlišit špatnou odpověď od jasně nesmyslné odpovědi. Nejjednodušší způsob jak v tomto případě odstranit všechny možné nesmyslné odpovědí je udělat pole možných špatných odpovědí. V případě násobení dvou čísel malé násobilky je to jednoduché. Je třeba vytvořit pole s horní a dolní mezí podle možných výsledků násobení a přidat rezervu, která určuje toleranci špatné odpovědi od nesmyslu.

[15]

```
private fun randomize() {
    val numberA = Random.nextInt(3, 9)
    val numberB = Random.nextInt(3, 9)
    val multiplicationResult = numberA * numberB

    qiChatbot.variable("numberA").value = "$numberA"
    qiChatbot.variable("numberB").value = "$numberB"
    qiChatbot.variable("result").value = "$multiplicationResult"
    val numberPhrases = ArrayList<Phrase>()
    for (i in 0..100) {
        if (i != multiplicationResult) {
            numberPhrases.add(Phrase("$i"))
        }
    }
    wrongNumbersSet.removePhrases(wrongNumbersSet.phrases)
    wrongNumbersSet.addPhrases(numberPhrases)
}
```

Obrázek 24 pole špatných odpovědí [15]

Tento příklad ukazuje tvorbu pole hodnot od 0 do 100, které se berou jako špatná odpověď. Pole se generuje v cyklu a následně se z něj odebírá správná odpověď. Následně se také uloží do svého obrazu, který se použije v dialogu.

```
dynamic: wrongNumbers

proposal: %MULTIPLY_CHALLENGE
Okay, what's, $numberA times, $numberB ?
   u1:($result)
   Right!
   u1:(~wrongNumbers)
   Wrong!
```

Obrázek 25 dialog s polem špatných odpovědí [15]

V tomto případě se programátor rozhodl nereagovat na nesmyslnou odpověď. V tomto případě se z tohoto dialogu nelze dostat dále bez odpovědi branné alespoň jako špatná. Takto může vzniknout problém například při práci s malými dětmi nebo pacienty v nemocnicích a psychiatrických léčebnách. Pro tento případ by bylo vhodné vytvořit externí spouštěč, který by automaticky resetoval dialog při neaktivitě (například formou časovače). Řešením také může být použití divoké karty pro možnost nesmyslu s hláškou a přepnutím do dalšího dialogu.

[15]

Návrat na odkaz

Pro cyklení v dialozích lze použít proměnnou, příkaz pro provedení nebo odkaz s párovou záložkou, což je asi nejjednodušeji pochopitelný způsob.

```
randomize()
qiChatbot.addOnBookmarkReachedListener {
   when (it.name) {
       "RERANDOMIZE" -> randomize()
   }
}
// Start the dialogue
chat.run()
```

Obrázek 26 dialogový odkaz [15]

Tento odkaz znovu odkazuje metodu náhody, a tedy umožňuje znovu vylosovat dialog z pole.

Pro návrat na proměnnou lze využít následující kód, který se odkazuje na proměnnou „show“.

```
qiChatbot.variable("show").addOnValueChangedListener {
   runOnUiThread {
       tabletLabel.text = it
   }
}
```

Obrázek 27 návrat na proměnnou [15]

Příkaz pro provedení se na rozdíl od ostatních uvedených variant provádí synchronně. Tento příkaz se v dialogu uvádí následující syntaxí: ^execute(„název“). Jelikož tato varianta pracuje na principu blokování, operační kód je znatelně více komplikovaný. Skládá se z prováděcího objektu:

```
val executors : Map<String, QiChatExecutor> = hashMapOf(
   "drumRoll" to DrumRollExecutor(qiContext)
)
qiChatbot.executors = executors
```

Obrázek 28 executor objekt [15]

[15]

A z přiřazené metody:

```
private inner class DrumRollExecutor(qiContext: QiContext) :
        BaseQiChatExecutor(qiContext) {
    override fun runWith(params: List<String>) {
        Thread.sleep(1_000)
    }
    override fun stop() {}
}
```

Obrázek 29 dedicated executor [15]

Sumarizace variant návratu:
Záložky – nepřerušují, nepřenášejí parametry
Proměnné – nepřerušují, pro jednoduchá použití, nedojde k volání, pokud se nezmění hodnota proměnné
Příkaz pro provedení – přerušuje, přenáší libovolné množství parametrů

[15]

4.5 Dialogflow agent

Dialogflow agent je framework chatbota od společnosti Google, který dokáže odpovídat na vstup v textové formě. Tato služba je zpoplatněna a cena se liší podle účelu použití. Dá se dobře využít s robotem Pepper, jelikož robot umí převádět hlasový vstup na text, na který následně může chatbot reagovat.

Pro vytvoření vlastního chatbota pomocí frameworku Dialogflow, je potřeba splnit následující dva kroky.

1) Integrovat obecného předpřipraveného agenta – dá se využít pro bežné hovory, vtipy a další běžné služby.
2) Definovat vlastní úmysly – tvorba vlastních dialogových vzorů pro specifické použití.

Po otevření, dialogflow automaticky vytvoří nového agenta. Je třeba se ujistit, aby pracoval v jazyce angličtina, jelikož tento jazyk je dostupný na každého robota Pepper. Do obecného agenta se následně přidávají vlastní dialogy pomocí textboxů.

Propojení android studia s agentem vyžaduje speciální povolení. Pro získání těchto povolení je nutné si založit služební účet a povolit agenta v souboru API.

Dialogflow používá následující knihovny, které je třeba importovat do projektu:

```
implementation 'com.google.cloud:google-cloud-dialogflow:0.92.0-alpha'
implementation 'com.google.http-client:google-http-client:1.29.1'
implementation 'junit:junit:4.12'
```

Dalším krokem, je automatická generace kódu agenta do zvoleného programovacího jazyku a vložení jej do aplikace. Tento kód je třeba upravit, aby pracoval správně s chováním robota.

Tvorba reakce

Reakce má za úkol propojit vstupní informaci do robota s chatbotem agentu. Pro jednoduchou reakci, kde robot začne mluvit lze použít následující kód:

[16]

```
class SimpleSayReaction internal constructor(context: QiContext, private val answer: String) :
   BaseChatbotReaction(context) {
   private var sayFuture: Future<Void>? = null

   override fun runWith(speechEngine: SpeechEngine) {
       val say = SayBuilder.with(speechEngine).withText(answer).build()
       sayFuture = say.async().run()
       try {
           sayFuture?.get() // Block until action is done
       } catch (e: ExecutionException) {
           Log.e("SimpleSayReaction", "Error during say: %e")
       }
   }

   override fun stop() {
       sayFuture?.requestCancellation()
   }
}
```

Obrázek 30: příklad jednoduché reakce [16]

Tvorba chatbota s dialog flow

Do chatbota se importuje dialogflow knihovna a výsledný kód může vypadat například takto:

```
class DialogflowChatbot internal constructor(context: QiContext,
                                             credentialsStream : InputStream
)
    : BaseChatbot(context) {
    companion object {
        private val TAG = "DialogflowChatbot"
    }
    private var dialogflowSessionId = "chatbot-" + UUID.randomUUID().toString()
    private val dataSource = DialogflowDataSource(credentialsStream)

    override fun replyTo(phrase: Phrase, locale: Locale): StandardReplyReaction {
        val input = phrase.text.toString()
        val language = locale.language.toString()
        var answer : String? = null
        try {
            answer = dataSource.detectIntentTexts(input, dialogflowSessionId, language)
            Log.i(TAG, "Got answer: '$answer'")
        } catch (e: Exception) {
            Log.e(TAG, "error", e)
        }
        return if (answer != null) {
            StandardReplyReaction(
                SimpleSayReaction(qiContext, answer), ReplyPriority.NORMAL
            )
        } else {
            StandardReplyReaction(
                EmptyChatbotReaction(qiContext), ReplyPriority.FALLBACK
            )
        }
    }
}
```

Obrázek 31 chatbot s dialogflow [16]

Pro toto použití je potřeba zajistit volání QISDK životních cyklů ze souboru hlavní aktivity a přidat do metody „onRobotFocusGained“ zpětnou vazbu.

[16]

```
private lateinit var chat : Chat

override fun onRobotFocusGained(qiContext: QiContext) {
    val credentials = applicationContext.resources.openRawResource(R.raw.credentials)
    val dialogflowChatbot = DialogflowChatbot(qiContext, credentials)
    chat = ChatBuilder.with(qiContext).withChatbot(dialogflowChatbot).build()
    chat.async().run()
}
```

Obrázek 32 Zpětná vazba pro chatbot s dialogflow [16]

4.6 Rozpoznávání obličeje

Pro tvorbu aplikace s rozpoznáním obličeje je třeba do aplikace přidat knihovnu. Tuto knihovnu lze do aplikace přidat dvěma způsoby.

1) Přidání knihovny jako zdrojový soubor

Pro tento způsob lze použít balíček Jitpack. Ten se vloží tak, že do souboru pro výstavbu aplikace se přidá uložiště balíčku a přidáním knihovny pro rozpoznání obličeje do souboru zdrojů.

```
allprojects {
                repositories {
                        ...
                        maven { url 'https://jitpack.io' }
                }
        }
dependencies {
        implementation 'com.github.softbankrobotics-labs:pepper-mask-recognition:master-SNAPSHOT'
}
```

Obrázek 33 přidání knihovny přímo do aplikace [17]

2) Přidáním knihoven OpenCV

Knihovny OpenCV lze také přidat dvěma způsoby.

a) Zahrnutím OpenCV native knihoven

Tato metoda přidává knihovny přímo do vaší aplikace, takže zvětšuje její datový objem. Knihovnu je třeba pouze nakopírovat do hlavního souboru a Android studio si již automaticky dohledá direktivu pro zahrnutí souborů do kódu.

b) Použitím externí aplikace s OpenCV

Tato metoda nainstaluje knihovnu přímo do robota a aplikace se na pouze odkazuje. Nevýhodou je, že aplikace bude fungovat pouze na robotech s nainstalovanou knihovnou. Pro instalaci knihovny na robota je třeba se pomocí manažéra aplikace připojit k robotu a nainstalovat balíček opencv.

```
$ adb connect 10.0.204.180:5555
$ adb install opencv-apk/OpenCV_3.4.7-dev_Manager_3.47_armeabi-v7a.apk
```

Obrázek 34 Příkazy pro instalaci OpenCV na robot Pepper [17]

[17]

Použití knihovny

Pro správné použití této knihovny je třeba ji načíst ve své aktivitě, nejlépe v metodě „onCreate“. Pro načtení je možné zavolat „OpenCVUtils.loadOpenCV(this)“:

```
class MyActivity : AppCompatActivity(), RobotLifecycleCallbacks() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        OpenCVUtils.loadOpenCV(this)
        //...
    }
```

Obrázek 35 načtení knihovny OpenCV [17]

Samotná detekce obličeje se skládá ze dvou částí:

- Záznamníku kamery, pro získávání snímku pro rozpoznávání
- Detektor, pro rozpoznávání kritérií

[17]

Kód pro detektor může vypadat následovně:

```
val detector = AizooFaceMaskDetector(this)
val capturer = BottomCameraCapturer(this, this)
val detection = FaceMaskDetection(detector, capturer)
detection.start { faces ->
    // Handle faces here
}
```

Obrázek 36 inicializace [17]

```
faces.forEach {
    when {
        (it.confidence < 0.5) && it.hasMask -> {
            // Process "someone has a mask"
        }
        (it.confidence < 0.5) && !it.hasMask -> {
            // Process "someone Doesn't have a mask"
        }
    }
}
```

Obrázek 37 detektor obličeje [17]

Zmíněný kód detektoru umí rozpoznat, jestli člověk nosí roušku nebo ne. Co ten detektor vlastně detekuje je definování metodou definované v knihovně OpenCV, takže detektor sám o sobě nic nerozlišuje, pouze volá předdefinovanou metodu.

Do kódu detektoru je také přidat vlastnost „boundingBox“ obličeje, která umožňuje vytvořit čtvercový rám kolem něj.

Tato akce se bude automaticky spouštět pokaždé, když bude naleze obličej člověka a životní cyklus robota dojede na tuto metodu v kódu (například 100krát/s). Pokud by měl robot například zdravit kolemjdoucí, je vhodné nastavit časovač a omezit tak spouštění této metody.

Výběr kamery pro záznam

Většinou je lepší použít kameru tabletu, protože má větší snímací frekvenci než kamera na hlavě robota. Před jejím použitím je však nutné nastavit povolení pro její použití. Po získání povolení stačí kameru inicializovat, což lze vidět na obrázku číslo 41.

Kamera umístěná na hlavě robota má výhodu, že může sledovat vidění robota. Pro její použití je třeba použít „TopCameraCapturer“, který vyžaduje „qiContext“ v metodě „onRobotFocusGained“:

```
override fun onRobotFocusGained(qiContext: QiContext) {
    val detector = AizooFaceMaskDetector(this)
    val capturer = TopCameraCapturer(qiContext)
    val detection = FaceMaskDetection(detector, capturer)
    detection.start { faces ->
        // Handle faces here
    }
}
```

Obrázek 38 Použití kamery na hlavě robota [17]

Pokročilé možnosti

Knihovna OpenCV má několik verzí s několika druhy modelů pro detekci. Zvolením správného modelu lze získat nové možnosti při konstrukci aplikace.

Pokud nevyhovuje žádný dostupný model v knihovnách, lze vytvořit model vlastní. Je dobré založit nový model na základě již fungujícího modelu.

[17]

5 Praktické využití robota Pepper na TUL

5.1 Návrh aplikace

Po poradě s vedoucím a práce, zvážení dostupných možností, uvážení časového limitu a dostupnosti jsem zvolil účel aplikace. Aplikace bude využívána při speciálních akcích, jako je například den otevřených dveří, k podávání informací, sdílení novinek, kontrole a prezentování.

Chci, aby v aplikaci byly splněny následující cíle:

- Chatbot, obsahující základní informace o TUL
- Schopnost robota pohybovat se alespoň po části dolního patra budovy A na TUL
- Využití animací v aplikaci
- Využití rozpoznání obličeje

Příprava aplikace

Před samotným vytvářením aplikace bylo na robotu Pepper na TUL potřeba vykonat několik změn. Robota bylo prvotně nutné aktualizovat na verzi 2.9. pro práci s QISDK a na to je třeba komunikovat s podporou firmy Softbank robotics. Podpora odpovídala poměrně rychle a snažila se vycházet vstříc, ale v některých chvílích jejich znalost anglického jazyka sotva stačila pro vysvětlení požadovaného úkonu na robotu. Pro aktualizaci robota bylo třeba nahrát nový obraz do uložiště robota. Obraz byl zaslán elektronickou poštou a do robota se nahrával pomocí protokolu FTP. Přenost FTP bylo třeba povolit v nastavení robota. Po nahrání obrazu je třeba robot aktualizovat, to řeší podpora centrálně a je proto třeba robot připojit na internet. Několikrát se stalo, že se aktualizaci z neznámých důvodů nezdařila a proto bylo třeba tento proces několikrát opakovat a to trvalo několik týdnů.

Dále je třeba připravit software pro vývoj aplikace. To znamená Android studio verze 3.4. a novější a přidat SDK pro robot Pepper. Návod pro přidání SDK lze najít na tomto odkazu:

Při mé instalaci SDK, nastalo několik chyb a bylo nutné jednotlivé chybějící soubory ručně doimportovávat pomocí příkazů v konzoli Android studia. Doporučuji si postup průběžně kontrolovat a může být nutné udělit výjimku antivirovému programu.

5.2 Tvorba prázdné aplikace

Návod pro tvorbu prázdné aplikace také lze nalézt na stránkách Softbank robotics:

https://developer.softbankrobotics.com/pepper-qisdk/getting-started/creating-robot-application

Bohužel tenhle návod není stoprocentně funkční. Při mé tvorbě aplikace jsem sice postupoval podle něj, ale musel jsem doimportovávat knihovny. Tato vada je pravděpodobně zapříčiněna aktualizacemi knihoven a neaktualizováním návodu. Pro využití základních metod pro robot Pepper je třeba doimportovat následující:

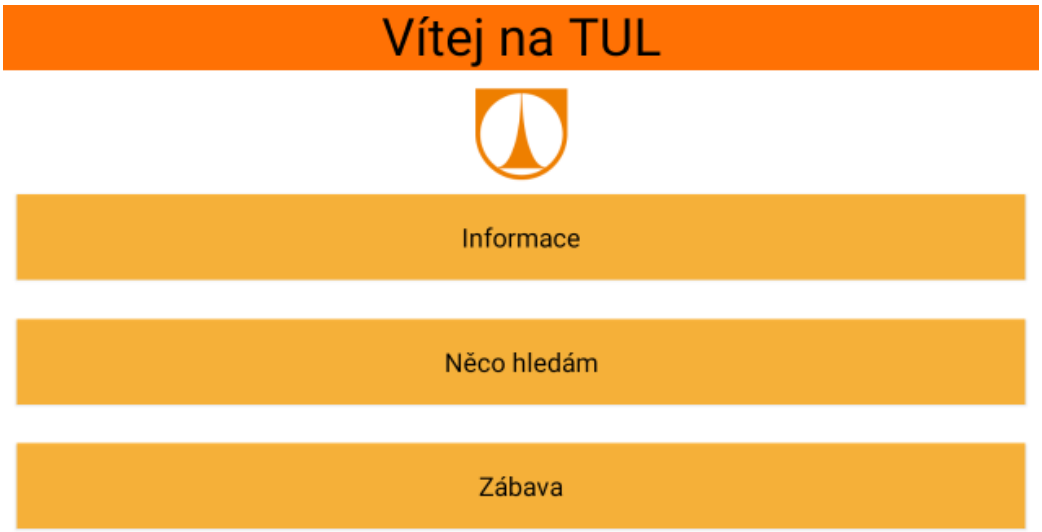
```
import com.aldebaran.qi.sdk.QiContext

import com.aldebaran.qi.sdk.QiSDK

import com.aldebaran.qi.sdk.RobotLifecycleCallbacks

import com.aldebaran.qi.sdk.design.activity.RobotActivity
```

a následně je možné využít základní doporučenou strukturu metod pro chování robota s základními metodami:

- onCreate – volání po vytvoření objektu
- onDestroy – volání po zničení objektu
- onRobotFocusGained – volání, jakmile se objekt stane aktivní
- onRobotFocusLost – volání, jakmile objekt ztratí aktivitu
- onRobotFocusRefused – volání, pokud při získávání aktivity dojde k chybě

5.3 Interface

Navržený interface se skládá ze čtyř základních oken. Hlavní okno se zapíná ihned po startu aplikace a funguje jako rozcestník na druhotná okna pro jednotlivé funkce.



Obrázek 39 Hlavní okno aplikace

Záložka informace odkazuje na okno, které má za úkol zodpovídat na otázky a podávat informace. V této akci robot reaguje na soubor klíčových slov a následně odpovídá předdefinovanými dialogy. Okno se skládá z hlavičky a textového pole, do kterého se vypisuje odpověď, kterou robot povídá v textové formě.

Obrázek 40 Záložka informace

Záložka „Něco hledám“ odkazuje na okno, kde jsou dostupné přidané body zájmu. V tomto okně se vygeneruje ke každému bodu zájmu ovládací prvek, pomocí kterého se nastaví trasa robotu. Trasování má několik možností pohybu. Pohyb maximální rychlostí, pohyb rovně (pravoúhle). Přepínač trasovací smyčka umožňuje zřetězit body zájmu za sebe a podle pořadí zvoleného na vygenerovaném ovládacím prvku se vytvoří trasa. Spodní řada tlačítek jsou přídavné funkce pro náhodnou zastávku, zastavování během trasy a zkratka pro zastávku s nabíjecí stanicí.

Obrázek 41 Okno mapy

V okně zábavy lze spustit ukázky, které mají za úkol předvést základní funkce robota. Uvedeny jsou tři typy ukázek a to:

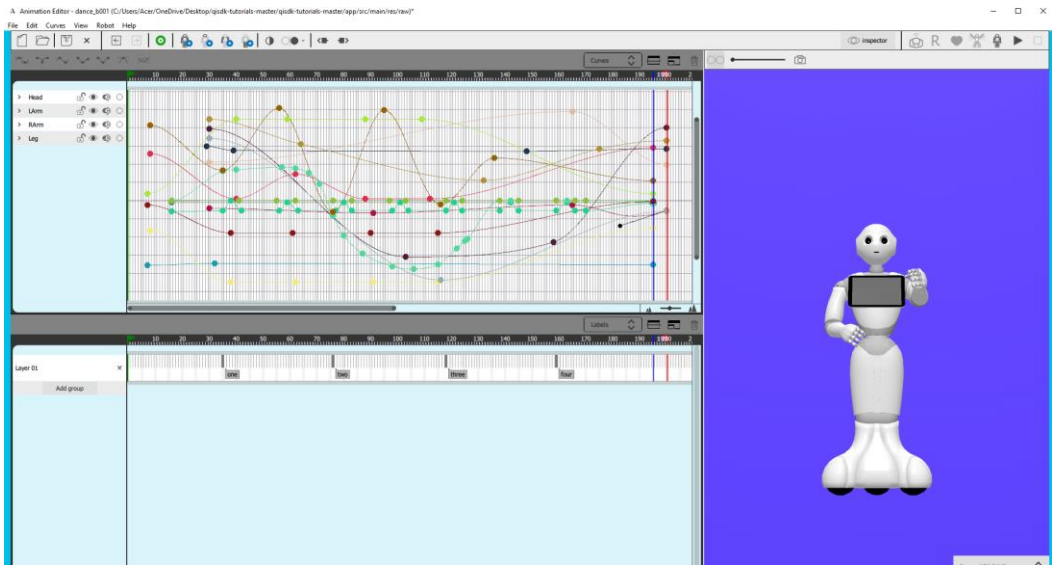
- Vtip
- Tanec
- Kvíz

Tyto ukázky se zapínají třemi tlačítky.

Obrázek 42 okno zábavy

Tvorba animací

Při tvorbě vlastních animací jsem vycházel z již předpřipravených vzorů. Tvorba animace je poměrně náročný proces a správné sladění pohybů dohromady vyžaduje hodně detailů a plánování, proto jsem převzal základ z příkladů a předělal pouze hlavní sekvenci pohybů. Výsledek není dokonalý, ale snad postačující, jelikož animace slouží v aplikaci pouze jako ukázka.



Obrátek 47 Schéma animace

5.4 Chatbot

Chatbot je v aplikaci využíván pro dvě použití:

- Řídící systém pro komunikaci a předávání informací mezi člověkem a robotem
- Řídící systém pro hlasové ovládání aplikace

Tvorba chatbota

Pro správné založení chatbota je třeba inicializovat několik komponentů.

- Topic – soubor se zápisem dialogů
- qiChatbot –přiřazení souboru tématu do chatbotu
- chat – nastavit chatbot jako vstupní informaci pro řečový systém

Kroky pro použití Chatbotu:

- inicializace komponentů
- spuštění řečového systému
- přepínání mezi kódem aplikace a dialogovým souborem

Funkce chatbot vyžaduje zdrojový soubor. To je soubor obsahující dialogy (pravidla s manuálem, jak se má chatbot chovat). Zdrojový soubor má následující strukturu:

- Hlavička souboru
- Hlavní pravidla
- Případná podpravidla

Zdrojový soubor pro informační použití

Použití pro předávání informací jsem řešil pomocí struktury pravidel a podpravidel. Cílem tohoto řešení bylo limitovat možný počet vstupních dialogů a co nejvíce zvýšit spolehlivost rozpoznání správného dialogu. Principiálně robot konverzaci rozkládá do několika menších úseků, ze kterých vybírá možné odpovědi. Tímto limituje chybovost, protože automaticky nemůže zvolit odpověď z jiného úseku (offtopic).

Informační soubor obsahuje:

- Zakladní informace o vedení fakulty
- Informace o studijním oddělení
- Termín podání přihlášek
- Reakce na pozdrav, kýchnutí a jiné

Tento zdrojový soubor využívá několik dalších funkcí.

```
proposal: %VOICE – proaktivní dialog (spouští se odkazem z kódu)

$showText         - dialogová proměnná

%SHOW_TEXT        - odkaz na záložku v kódu

^stayInScope      - funkce, zamezující přechod na jinou dialogovou úroveň
```

Hlavní využití dialogové proměnné v tomto souboru je předat informaci o textu dialogu, který robot říká, aby mohl být následně vypsán na obrazovku tabletu.

Zdrojový soubor pro hlasové ovládání

Logika pro hlasové ovládání se trochu liší od informační. V tomto případě není moc možných výstupů (3 – 4 ovládací prvky uživatelského rozhraní) ale chceme, aby robot reagoval na co nejvíce možných synonymních vstupních dat. Toto řeším rozpoznáváním klíčových slov. Tato metoda pomocí předem definovaných syntaktických znaků a funkcí nejdříve vstupní data upraví a pak až porovná. Nevýhodou je snížená přesnost rozpoznání, to je ovšem limitováno malým počtem výstupů a druhou nevýhodou je skutečnost, že v češtině se skloňuje a časuje, takže je nutno dát pozor na možné tvary výrazu.

Tento dialogový soubor využívá:

```
proposal: %VOICE – proaktivní dialog (spouští se odkazem z kódu)

$showText         - dialogová proměnná

^rand             - funkce volící pseudonáhodný prvek v dialogu
```

Na rozdíl od informačního souboru, tento soubor využívá dialogovou proměnnou jako vlajku pro dosažení určitého bodu v dialogu a pro následné propojení s hlavním kódem pomocí neblokujícího propojení (záložky). Vyhnul jsem se použití execute, protože se jedná o asynchronní systém a je pro tuto aplikaci vhodnější úkon zpozdit a uložit do fronty, nežli předchozí úkon přerušit.

Metoda klíčových slov využívá syntaktický znak „*“ ,takzvanou divokou kartu, která označuje jakékoliv vstupní dato. V kombinaci se zadaným konkrétním vstupním datem vytvoří filtr, který se spustí kdykoliv, když bude vstup obsahovat zadané slovo. Dialog následně odkazuje na záložku, která je propojená s kódem, který přepíná aktivity v aplikaci.

5.5 Orientace a pohyb v prostoru

Pro pohyb a orientaci aplikace vychází z volně dostupné demo aplikace od Softbank Robotics dostupné zde: *https://github.com/softbankrobotics-labs/maplocalizeandmove*

Systém této aplikace se skládá ze dvou fází:

- mapování a skenování prostředí
- pohyb ve zmapovaném prostoru

Při mapování je nutné robot tlačit do důležitých míst, kde se má ve druhé fázi pohybovat. V každém z těchto bodů si robot naskenuje své prostředí a z těchto útržků si postupně skládá mapu okolí, je tedy nutné skenovat prostředí dostatečně frekventovaně a zároveň při skenování dbát na odstranění dočasných překážek a dostatečně si od robota odstoupit.

Ve fázi pohybu se robotu nastaví na vygenerované mapě trasa, po které se má pohybovat. Při pohybu po trase na robotu stále běží rutinní program skenování okolí a robot se snaží vyhnout případným překážkám, které nemá na naskenované mapě.

Princip kódu pohybu

Pohybový systém aplikace se skládá z hlavní aktivity pohybu, která slouží jako knihovna metoda, které volají podpůrné fragmenty. Fragmenty slouží převážně k přepínání grafického rozhraní a volání připravených funkcí. Dále jsou v aplikaci takzvané „utility“ skripty, které jsou zaměřené na jednu specifickou činnost.

Funkce hlavní aktivity pohybu

- Získání povolení pro použití uložiště
- Kontrola úrovně nabití baterie
- Kontrola připojení napájení -> povolení nebo zakázání pohybu robota
- Přepínání fragmentů
- Automatické přidání lokace pro nabíjení baterie (vyžaduje nabíjecí podložku)
- Přechod do cílové lokace
- Přechod do náhodné lokace
- Uložení lokace
- Mapování okolí

Funkce jednotlivých fragmentů

Hlavní fragment

- Hlavní obrazovka pro mapovací systém
- Výběr mezi skenováním mapy a pohybem mezi lokacemi

Fragment nastavení

- Výběr mezi tvorbou/úpravou mapy a uložením bodů zájmu

Fragment mapování

- Umožňuje vytvořit novou mapu nebo rozšířit stávající mapu

Fragment ukládání pozic

- Umožňuje uložit a upravovat pozice v naskenované mapě

Fragment produkce

- Umožňuje přejít na lokalizaci robota nebo do režimu tras

Fragment lokalizace

- Umístí robota do naskenované mapy a povolí režim tras

Fragment režimu tras

- Umožňuje robotu naplánovat trasu mezi uloženými body zájmu

Fragment splash a loading

- Pouze jako přechodová obrazovka

Funkce utility skriptů

- Inicializace
- Výpočet trajektorie
- Ukládání souborů
- Tvorba modelu mapy
- Animace na UI

5.6 Možná rozšíření aplikace

Vzhledem k nedostupnosti robota během roku je v aplikaci pořád spoustu místa pro rozšíření a úpravu. Hlasové ovládání v aplikaci sice funguje, ale jeho přesnost není maximalizována. Pro zlepšení spolehlivosti hlasového ovládání a odstranění nechtěných reakcí na šum by bylo dobré přidat více hlasových vstupů pro spuštění dialogů a akcí.

Aplikace neobsahuje využití rozpoznávání obličeje, kterého je robot Pepper schopen. Implementací této funkce lze například rozlišit některé zaměstnance děkanátu nebo upravit dialog na základě odhadovaného věku osoby. Je třeba ovšem brát ohled na spolehlivost, a tudíž nepřiřazovat dialogu vysokou důležitost a používat jej pouze doplňkově.

Další možný upgrade aplikace by mohlo být rozšíření použití na celou budovu a nejenom spodní patro. Problém je pohybu nahoru a dolu, jelikož robot jezdí na podvozku se třemi koly, tak nedokáže využívat schodiště budovy. Řešením by bylo použití výtahu, a to vyžaduje interakci s hmotnými objekty (stisknutí tlačítka pro otevření dveří a zvolení patra).

Pro vylepšení vystupování robota je možné vytvořit jednu standardní mapu, jelikož aplikace je konstruována na použití na konkrétním místě, obsahující doprovodné dialogy k bodům zájmu jako například informace o provozních hodinách.

Jako poslední vylepšení uvádím optimalizace kódu a ošetření výjimek. Sice jsem při používání aplikace žádné výjimky nenašel, ale bylo provedeno poměrně málo testů finální verze aplikace.

6 Bezpečnostní rizika

Připojení k robotu Pepper je možné dvěma způsoby. Prvním způsobem je připojení pomocí portálu Command center zmíněný již dříve v bakalářské práci. Ten slouží jako hlavní způsob pro připojení k robotu a umožňuje nastavovat robot, nahrávat aplikace a spravovat přistup dalších uživatelů k robotu. Portál je chráněn heslem a pokud nejste vlastníkem robota, majitel musí váš účet pozvat pro práci s robotem. Existují 4 stupně správy robota v Command center a to „divák“, umožňující pouze sledování změn na robotu. „Uživatel“, umožňuje nahrávat vydané aplikace do robota. „Vývojář“, kompletní přístup k robotu kromě přidávání nových uživatelů a „vlastník“, který má dostupné veškeré možnosti.

Druhý způsob pro připojení k robotu je přímo spojit robota s PC. Tento způsob se využívá ve vývoji k testování aplikace nebo jejích jednotlivých částí a je umožněn skrze Android studio. I toto připojení je chráněno heslem a podmínkou přímého připojení k robotu je být připojený ke stejné síti. Přímo spojit se s robotem je také možné pomocí protokolu FTP. Tento způsobem je využívám hlavně jako nouzová možnost při problémech, které nelze vyřešit online (robot nejde připojit k internetu nebo chyba aktualizací a odmítání další aktualizace). Já tento způsob využil při instalaci nové verze operačního systému robota z verze 2.1. na verzi 2.9. Obě metody pro přímé připojení sdílí stejné heslo, ovšem pomocí FTP se lze dostat až k souborům robotu, kdežto Android studio umožňuje pouze nahrát aplikaci, takže přímé připojení pomocí FTP je nebezpečnější.

Z hlediska zdravotních rizik nepředstavuje robot žádné nebezpečí. Senzory robot zastavují s dostatečnou rezervou. Dokonce i při selhání bezpečnostních opatření robot nedisponuje velkým výkonem a neměl by být schopen člověku ublížit.

7 Závěr

V této bakalářské práci jsem se seznámil s humanoidním robotem Pepper pro použití na pracovišti školitele. Byla provedena rešerše stávajících dostupných úloh zaměřená zejména na vlastnosti úloh a jejich použití v aplikaci.

Byl vytvořen návrh aplikace pro komunikaci s návštěvníky děkanátu, která je řešena jak hlasovým vstupem, tak pomocí tabletu robota. Tento návrh se skládá z několika obrazovek, každá umožňující jiné interakce s robotem. Cílem návrhu bylo v aplikaci využít co nejvíce dostupných úloh robota a zároveň dodržet účel, podávání informací a prezentace robota na dnech otevřených dveří. Návrh obsahuje veškeré úlohy robota, ke kterým se mi podařilo dohledat dokumentaci nebo jinak získat informace pro jejich použití.

Návrh jsem i přes velké množství problémů se zastaralou verzí robota, neaktualizovanými informacemi o robotu a samotným přístupem k robotu realizoval dostatečně pro ověření funkcionality. Aplikace umožňuje komunikaci robota s návštěvníky (hlasem i přes tablet), podání informací ohledně školy, ukázat pohyb robota pomocí animace tance a pohyb na vybraná místa po děkanátu podle předem naskenované mapy. Pro pohyb robota po děkanátu byla využita volně dostupná ukázková aplikace z webových stránek Softbank robotics. Oproti návrhu realizovaná aplikace postrádá implementaci modulu rozpoznání obličeje, z důvodu problematiky implementace vlastního modelu.

Při testování aplikace jsem nalezl několik zranitelných míst. Systém pro rozpoznání hlasu není zdaleka bezchybný a stává se, že i zvuky, které nejsou řeč, jsou zaznamenávány a robot se na ně snaží reagovat. Dalším zranitelným místem aplikace je mapování místností s omezeným prostorem. Pro tvorbu kvalitní mapy je třeba držet robot alespoň 1,5 m daleko od překážek, jinak se robot může mapování odmítnout. Z těchto důvodů aplikace není vhodná pro použití v uzavřených hlučných prostorách se špatnou akustikou, to však při pohybu po děkanátu nenastává a pokud aplikace nerozpozná dialog, není problém otázku zopakovat.

Posledním cílem bakalářské práce je diskutovat výhody a bezpečnostní rizika. Výhodou a účelem, za kterým byla aplikace navržena, je její jednoduchost a přehlednost. Pro použití jako průvodce po děkanátu je třeba, aby se v aplikaci zorientoval jakýkoliv uživatel nehledě na jeho znalosti s robotem nebo jinými elektronickými zařízeními, takže aplikace je navržena, aby byla co nejjednodušší. Nevýhodou aplikace jsou její již zmíněná zranitelná místa, takže spolehlivost. Ohledně bezpečnosti pojednává kapitola 6 a veškerý přístup k robotu je řešen minimálně standardní ochrannou, heslem.

Seznam literatury

[1] Softbank *robotics: Pepper* [online]. Softbank robotics, 2021 [cit. 2021-4-6]. Dostupné z: https://www.softbankrobotics.com/emea/en/pepper

[2] *Softbank robotics: Pepper robot technical specification* [online]. Softbank robotics, 2021 [cit. 2021-4-6]. Dostupné z: https://pepper.generationrobots.com/technical-specifications.html?lang=en

[3] *Aldebaran: Pepper tablet* [online]. Softbank robotics, 2021 [cit. 2021-4-6]. Dostupné z: http://doc.aldebaran.com/2-4/family/pepper_technical/tablet_pep.html

[4] *GenerationRobots: Pepper datasheet* [online]. Softbank robotics, 2017 [cit. 2021-4-6]. Dostupné z: https://pepper.generationrobots.com/Pepper%20Datasheet%201.8a%2020170116%20EMEA.pdf

[5] *GenerationRobots: Romeo* [online]. Softbank robotics, 2018 [cit. 2021-4-6]. Dostupné z: https://blog.generationrobots.com/en/romeo-naos-big-brother-robot/

[6] *Honda: Asimo* [online]. Honda, 2021 [cit. 2021-4-6]. Dostupné z: https://asimo.honda.com/

[7] *NASA: R5* [online]. NASA, 2021 [cit. 2021-5-10]. Dostupné z: https://www.nasa.gov/feature/r5/

[8] *Hanson Robotics: Sophia* [online]. Hanson Robotics, 2021 [cit. 2021-5-10]. Dostupné z: https://www.hansonrobotics.com/sophia/

[9] *Softbank robotics: Command center* [online]. Softbank robotics, 2021 [cit. 2021-5-10]. Dostupné z: https://developer.softbankrobotics.com/pepper-qisdk/lessons/how-deploy-android-application-pepper-qisdk-command-center

[10] *Softbank robotics: robot application* [online]. Softbank robotics, 2021 [cit. 2021-5-10]. Dostupné z: https://developer.softbankrobotics.com/pepper-qisdk/lessons/realistic-approach-robotic-behaviour-development

[11] *Softbank robotics: realistic approach* [online]. Softbank robotics, 2021 [cit. 2021-5-10]. Dostupné z: https://developer.softbankrobotics.com/pepper-qisdk/lessons/basics-animation-creation

[12]Softbank robotics: Multimodal presentation [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/pepper-qisdk/lessons/making-multimodal-presentation/part-1-speech-and-images

[13]Softbank robotics: Mastering pronounciation [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/pepper-qisdk/lessons/mastering-pronunciation/basic-tweaks

[14]Softbank robotics: mastering pronounciation [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/pepper-qisdk/lessons/discovering-qichat-sbr-language-creating-chatbots

[15]Softbank robotics: qichat [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/pepper-qisdk/lessons/linking-qichat-and-code/asking-smart-questions-bookmarks-variables-and-dynamic

[16] *Softbank robotics: qichat variables* [online]. Softbank robotics, 2021 [cit. 2021-5-10]. Dostupné z:
https://developer.softbankrobotics.com/pepper-qisdk/lessons/integrating-chatbot-dialogflow

[17] *Softbank robotics: dialogflow agent* [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/blog/pepper-face-mask-detection

[18] *Pepper at Costa cruiz* [online]. Costa cruiz, 2021 [cit. 2021-5-10].
Dostupné z:
https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQfSsbs8exP1VEzRIwv2bxAQluL4CwnaPX9kQ&usqp=CAU

[19] *Softbank robotics: Romeo documentation* [online]. Softbank robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://developer.softbankrobotics.com/nao-naoqi-2-1/romeo-documentation

[20] *Asimo* [online]. Honda, 2021 [cit. 2021-5-10].
Dostupné z:
https://upload.wikimedia.org/wikipedia/commons/thumb/0/0c/Honda_ASIMO_%28ver._2011%29_2011_Tokyo_Motor_Show.jpg/1200px-Honda_ASIMO_%28ver._2011%29_2011_Tokyo_Motor_Show.jpg

[21] *R5* [online]. NASA, 2021 [cit. 2021-5-10].
Dostupné z:
https://www.irozhlas.cz/sites/default/files/images/03634106.jpeg

[22] *Sophia* [online]. Hansom Robotics, 2021 [cit. 2021-5-10].
Dostupné z:
https://upload.wikimedia.org/wikipedia/commons/1/1e/Sophia_at_the_AI_for_Good_Global_Summit_2018_%2827254369347%29_%28cropped%29.jpg

Obsah přiloženého CD

Přiložené CD obsahuje:

- Text bakalářské práce
- Soubor s vybranými zdrojovými kódy