Tampereen ammattikorkeakoulu

Pepper-robotin Java-ohjelmointi

Atte Yliverronen

SISÄLLYS

1. JOHDANTO

Tämä on Pepper-robotin (OS 2.9+) Java-ohjelmointi opas. Ohjelmoinnissa käytetään Android Studiota sekä siihen asennettavia lisäosia, joita tarvitaan Pepperin ohjelmointiin.

Android Studion saa https://developer.android.com/studio ja sieltä löytyy tarvittava ohjeistus sen asentamiseen.

Ohjeistus tarvittavien lisäosien asentamiseen löytyy https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/ch1_gettingstarted/installation.html

Tässä ohjeessa oletetaan, että Android Studio ja siihen tarvittavat lisäosat on asennettu. Ohjeessa myös oletetaan perusosaaminen Java-ohjelmoinnista.
Lisäksi oletetaan, että annetun linkin ohjeistusta on seurattu next steps - vaiheeseen asti.

Ohjeessa tullaan käymään läpi Pepperin perustoiminnallisuus. Työssä esitellään, miten se saadaan tuottamaan ja ymmärtämään puhetta. Lisäksi esitellään Pepperin saaminen elehtimään, sen tabletin hallinta ja perusliikkuminen.

2. PEPPER JA KESKUSTELU

2.1 Say ja Listen

On kaksi eri tapaa saada Pepper tuottamaan puhetta ja kaksi eri tapaa saada se ymmärtämään puhetta. Ensimmäinen tapa saada Pepper tuottamaan puhetta on käyttää Say-luokkaa. Tällä saadaan Pepper sanomaan pelkästään koodia kirjoittamalla helposti yksinkertaisia asioita, mutta mitään keskustelua on mahdoton saada aikaiseksi pelkästään tätä käyttäen. Tulevassa kuviossa 1 on esimerkki Say:n käytöstä.

```
Say say = SayBuilder.with(qiContext) // Create the builder
with the context.
        .withText("Hello human!") // Set the text to say.
        .build(); // Build the say action.
//Execute the action.
say.run();
```

Kuvio 1. Esimerkki Say:n käytöstä.

Ensimmäinen tapa kuunnella on käyttäen Listen-luokkaa. Tällä pystytään pelkkää koodia hyödyntäen kuuntelemaan helposti puheesta avainsanoja tai vaikka kokonaisia lauseita ja sen jälkeen voidaan reagoida niihin, miten halutaan. Tulevassa kuviossa 2 on esimerkki Listen:stä.

```
PhraseSet phraseSet = PhraseSetBuilder.with(qiContext) //Cre-
ate the builder with the context
        .withTexts("Hello", "Hi") //Set Phrases we are looking
for
        .build(); // Build the phrase set

Listen listen = ListenBuilder.with(qiContext) //CreateCreate
the builder with the context.
        .withPhraseSet(phraseSet) //Give Phrases to listen ac-
tion
        .build(); // Build the listen action

ListenResult listenResult = listen.run();
Log.i("testi", "Heard phrase: " + listenResult.getHeard-
Phrase().getText()); // Prints heard phrase.
```

Kuvio 2. Esimerkki Listen:n käytöstä

Yhdistelemällä Say:tä ja Listeni:ä on mahdollista saada Pepper käymään keskustelua ja reagoimaan erilaisiin sanottuihin asioi halutulla tavalla. Tämä muuttuu toki melko työlääksi heti, jos halutaan yhtään pidempiä tai monimutkaisempia keskusteluja ja onkin olemassa parempi tapa saada Pepper käymään keskusteluja, johon tutustutaan seuraavaksi. Seuraavassa kuviossa 3 on esimerkki Say:n ja Listen:in käytöstä yhdessä.

```
Say say = SayBuilder.with(qiContext) // Create the builder
with the context.
        .withText("Hello human!") // Set the text to say.
        .build(); // Build the say action.
//Execute the action.
say.run();

PhraseSet phraseSet = PhraseSetBuilder.with(qiContext) //Cre-
ate the builder with the context
        .withTexts("Hello", "Hi") //Set Phrases we are looking
for
        .build(); // Build the phrase set

Listen listen = ListenBuilder.with(qiContext) //CreateCreate
the builder with the context.
        .withPhraseSet(phraseSet) //Give Phrases to listen ac-
tion
        .build(); // Build the listen action

ListenResult listenResult = listen.run();
Log.i("testi", "Heard phrase: " + listenResult.getHeard-
Phrase().getText()); // Prints heard phrase.

//Checking if we heard what we wanted to hear and speak after
that
if(phraseSet.getPhrases().contains(listenResult.getHeardPh-
rase())){
    SayBuilder.with(qiContext).withText("how are
you?").build().run();
}
```

Kuvio 3. Esimerkki Say:n ja Listen:n yhdistämisestä keskustelua varten.

2.2 Chat

Chat yhdistää suoraan sekä Say:n että Listen:n toiminnat saman luokkaan kätevästi ja sen avulla onkin huomattavasti helpompi tehdä monimutkaisempia ja pidempiä keskusteluja.

Ensimmäisenä kun aloitetaan chat:n käyttöä niin luodaan topic-tiedosto (file->new->chat topic). Topic-tiedosto pitää sisällään kaiken mitä robotti tarvitsee kyseisestä asiasta keskustelemiseen. Se pitää sisällään, mistä keskustelu alkaa, josta robotti tunnistaa kyseisen keskustelun alkavan ja koko keskustelun kulun.

Topic:ssa on muutama asia, josta on hyvä tässä kohtaa puhua. Ensimmäinen niistä on concept. Concept on tunniste samaa asiaa tarkoittavista sanoista ja lauseista. Kuvassa 1 näkyy yksinkertainen esimerkki conceptista.

```
concept:(hello) [hello hi hey "good *" greetings]
```

Kuva 1. Esimerkki conceptista.

Yllä olevan esimerkin mukaisesti luodaan tervehdys-concepti.

Toinen asia mistä puhumme nyt, on conceptien käyttö ja keskustelun kulku. Keskustelu kulkee varsin yksinkertaisesti niin, että käyttäjä puhuu ja robotti reagoi käyttäjän puheeseen. Seuraavassa kuvassa 2 on esimerkki siitä, kuinka robotti saadaan vastaamaan käyttäjän tervehdykseen.

```
topic: ~esimerkki() #Defines topic of the file, by default same as file name
# Defining extra concepts out of words or group of words
concept:(hello) [hello hi hey "good *" greetings]
concept:(reply) [hello hi hey "good morning" greetings]
# Replying to speech
u:(~hello) ~reply
```

Kuva 2. esimerkki siitä kuinka robotti saadaan vastamaan käyttäjän tervehdykseen.

Annetussa esimerkissä käyttäjä tervehtii robottia jollain hello conceptin sisällä olevista sanoista tai lauseista ja robotti vastaa tälle yhdellä replyn alla olevista.

Topic:lla pystytään helposti rakentamaan myös pidempiä keskusteluja, joissa on eri keskustelu haaroja. Kuvasta 3 näkyy kuinka topic:n sisällä voidaan rakentaa haarautuva keskustelu.

```
topic: ~introduction()
# Defining extra concepts out of words or group of words
concept:(hello) [hello hi hey "good morning" greetings]
concept:(good) [fine good "doing fine" "doing good"]
concept:(bad) [bad tired sad "feeling down"]
# Replying to speech
u:(~hello) how are you?
    u1:(~good) thats good to hear!
    u1:(~bad) aww, thats sad. did something happen?
        u2:(yes) what happened?
        u2:(no) im sure you will feel better soon!
```

Kuva 3. Esimerkki haarautuvasta keskustelusta.

Ja sitten hyödynnämme topic-tiedostoa itse keskustelussa, joka onkin sitten paljon yksinkertaisempi asia. Kuviosta 4 näkyy kuinka topic-tiedostoa käytetään chatBotin avulla.

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
    Topic topic = TopicBuilder.with(qiContext).withRe-
source(R.raw.esimerkki).build(); //Gets topic file for chatbot
to use
    QiChatbot qiChatbot = QiChatbotBuilder.with(qiCon-
text).withTopic(topic).build();
    //Creates QiChatbot for Chat to use
    Locale locale=new Locale(Language.FINNISH,Region.FINLAND);
    chat = ChatBuilder.with(qiContext).withChatbot(qiChat-
bot).withLocale(locale).build(); //Creates chat using the
chatbot and gives it languege to speak
    chat.addOnStartedListener(() -> Log.i("Testi", "Discussion
started.")); //adds listener for when chat is started and
prints to log when it's going.
    Future<Void> chatFuture = chat.async().run(); //Starts the
chat asyncronically
    //prints if chat has error starting
    chatFuture.thenConsume(future -> {
        if(future.hasError()){
            Log.e("Error", "Discussion finished with error.",
future.getError());
        }
    });
```

Kuvio 4. QiChatBotin käyttäminen

Ja sitten tarvitsee vain poistaa startListener chat:stä kun robotti menettää focuksen ja lisätä chat luokan sisäiseksi muuttujaksi. Kuviossa 5 on esimerkki Chat:n lisäämisestä luokka muuttujaksi ja startListenerin poistaminen chatistä, kun robotti menettää focuksen.

```
public class MainActivity extends RobotActivity implements Ro-
botLifecycleCallbacks {
    private Chat chat;
```

```
public void onRobotFocusLost() {
    // The robot focus is lost.
    if(chat!=null){
        chat.removeAllOnStartedListeners();
    }
```

Kuvio 5. Esimerkit chat luokkamuuttujasta ja onStartedListenerin poistamisesta.

Ja tässä onkin kaikki mitä tarvitsee tehdä siihen, että robotti osaa käydä yhden keskustelun.

Myös sen tekeminen, että robotin saa käymään useampaa mahdollista keskustelua samaan aikaan on varsin helppoa. Luodaan vain qiChatBot useamman topic:n kanssa. Tulevassa kuviossa 6 on esimerkki useamman topicin käytöstä samassa QiChatbotissa.

```
List<Topic>topics=new ArrayList<>();
//Create list to hold topics
topics.add(TopicBuilder.with(qiContext).withRe-
source(R.raw.esimerkki).build());
//add topic to the list
topics.add(TopicBuilder.with(qiContext).withRe-
source(R.raw.keskustelu).build());
//add topic to the list
topics.add(TopicBuilder.with(qiContext).withRe-
source(R.raw.food).build());
//add topic to the list
QiChatbot qiChatbot = QiChatbotBuilder.with(qiContext).withTo-
pics(topics).build();
//Creates QiChatbot for Chat to use
Locale locale=new Locale(Language.FINNISH,Region.FINLAND);
chat = ChatBuilder.with(qiContext).withChatbot(qiChat-
bot).withLocale(locale).build(); //Creates chat using the
chatbot and gives it languege to speak
chat.addOnStartedListener(() -> Log.i("Testi", "Discussion
started.")); //adds listener for when chat is started and
prints to log when it's going.
Future<Void> chatFuture = chat.async().run(); //Starts the
chat asyncronically
//prints if chat has error starting
chatFuture.thenConsume(future -> {
    if(future.hasError()){
        Log.e("Error", "Discussion finished with error.", fu-
ture.getError());
    }
});
```

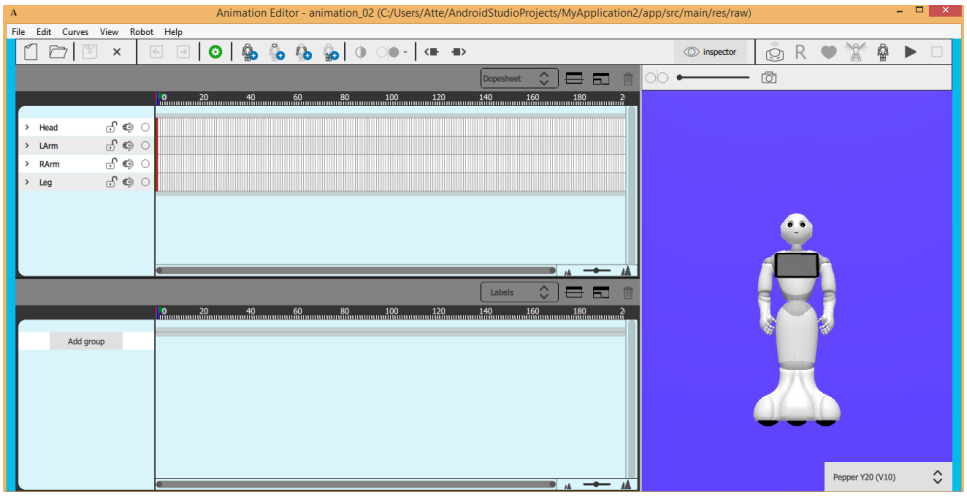
Kuvio 6. Esimerkki useamman topic:n lisäämisestä chatBottiin.

Useampaa topic:a käytettäessä samaan aikaan täytyy vain muistaa muutama asia. Kun keskustelusta vaihdetaan toiseen Pepper ei muista mihin vanhassa keskustelussa jäätiin. Toinen asia, jota kannattaa varoa on, ettei samoja avain sanoja esiintyisi eri topic:ssa, koska muuten Pepper voi vaihtaa keskustelua topic:sta toiseen, vaikka käyttäjä ei olisi näin tarkoittanut.

3. PEPPER JA ELEHTIMINEN

3.1 Eleiden luominen

Tässä eleillä tarkoitetaan kaikki Pepperin tekemiä liikkeitä, jotka eivät liikuta Pepperiä pois samasta paikasta lattialla, eli siis esim. vaikka käden heiluttamista, kumarruksia tai pään kääntelyä. Eleitä kutsutaan tästä eteenpäin animaatioiksi, koska niin niitä kutsutaan ohjelman sisällä. Uuden animaation luominen: File > new > animation timeline. Anna haluamasi nimi ja se luo .qianim-päätteisen tiedoston, jota kaksoisklikkaamalla avautuu animaatioeditori. Tulevasta kuvasta 4 näkee minkä näköinen animaatioeditori on.



Kuva 4. Editori näyttää tältä.

Tulevassa kuvassa 5 näkyy ikonit, mistä valitaan, mitä ruumiin osaa halutaan, että pepper liikuttaa.

Kuva 5. Näistä ikoneista valitaan mitä, ruumiinosaa Pepperin halutaan liikuttavan.

Laitetaan Pepper nostamaan kätensä esimerkkinä. Ensimmäisenä painetaan hiiren vasemmalla painikkeella johonkin kohtaan LArm-sarakkeella. Kuvasta 6 näkyy sarake, mitä pitää painaa.

Kuva 6. Kuvasta näkyy sarake, mitä pitää painaa.

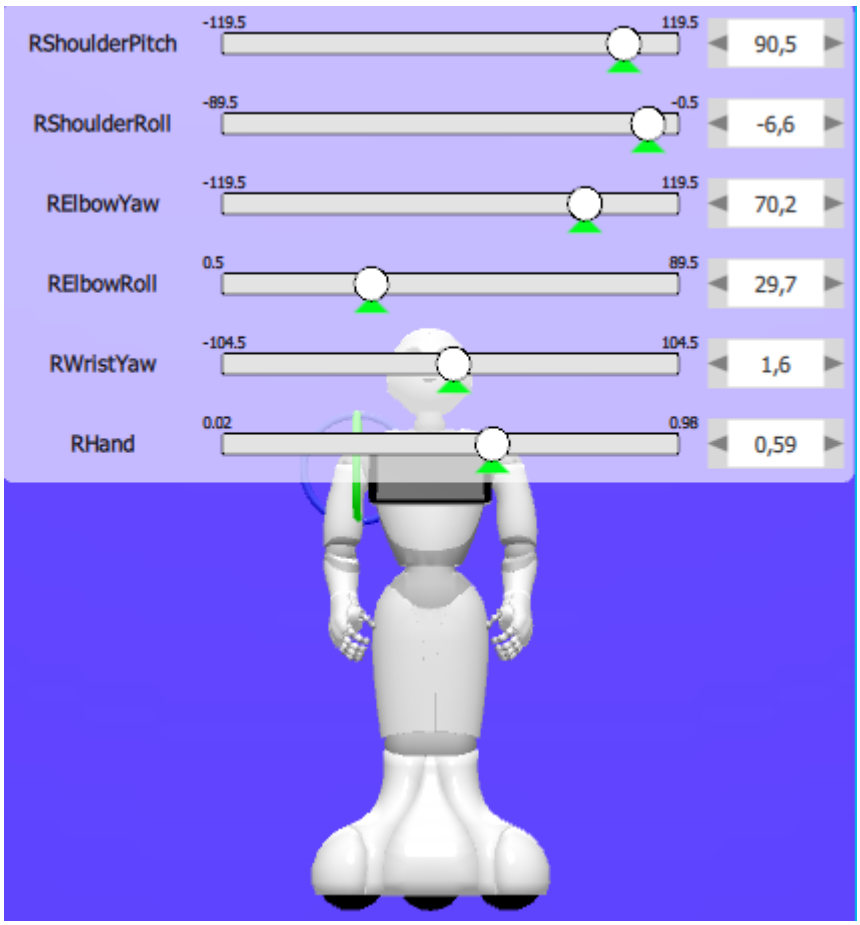
Huomaa, että sininen viiva siirtyi kohtaan, mihin painoit. Seuraavaksi valitaan Create Arms key(s) joka näkyy tulevassa kuvassa 7.

Kuva 7. Kuvassa näkyy Create Arms key(s) ikoni.

Huomaa kuinka LArm- ja RArm-sarakkeisiin ilmestyi kuusikulmiot, joka näkyykin tulevassa kuvassa 8.

Kuva 8. Kuva siitä miltä, pitäisi suurin piirtein näyttää, jos teki kaiken oikein.

Kuusikulmiot (key) merkitsevät hetkeä, mihin mennessä liike pitää olla valmis ja samalle ruumiinosalle voidaan antaa uusi liike alkamaan tästä hetkestä. Seuraavaksi valitaan Pepperin käsi kuten tulevassa kuvassa 9 on tehty.



Kuva 9. Tältä näyttää, kun pepperin käsi on valittuna.

Kaikki sliderit määrittävät käden yhdentyyppistä liikettä ja sitä mihin, liike pyritään lopettamaan. Kannattaa huomata esikatselussa näkyvä asento ja loppuasento eivät välttämättä ole samat.

Säädetään esimerkiksi ShoulderPitch-slideriä niin kauan, että käsi menee ylös esikatselussa. Luodaan uusi Arms key nykyistä myöhemmäksi ja säädetään siinä käsi ala-asentoon. Paina play-nappia ja katso mitä tapahtuu. Pepperin kaikki eleet luodaan samalla tavalla ja samaan animaatioon voidaan laittaa periaatteessa ihan niin pitkät elesarjat kuin halutaan.

Animaation käyttäminen koodissa on varsin yksinkertaista kuten tulevassa kuviossa 7 näkyy.

```
Animation animation= AnimationBuilder.with(qiCon-
text).withResources(R.raw.animation_00).build(); //builds Ani-
mation
Animate animate = AnimateBuilder.with(qiContext).withAnima-
tion(animation).build(); //builds runnable animation
animate.run(); //starts the animation
```

Kuvio 7. Animaation käyttäminen koodissa

4. PEPPER JA TABLET

4.1 Pepperin tabletti

Pepperin tabletti on ihan perus Androidi. Siinä voi pyörittää ihan normaaleja Android-ohjelmia. Tabletti myös hallitsee Pepperiä käyttöliittymäversiosta 2.9 eteenpäin. Tabletin näytön hallitseminen on pääosin varsin yksinkertaista ja toimii samalla tavalla kuin normi Androidi.

Esimerkkinä luodaan nyt sovellus, joka ottaa kuvan, siitä mitä Pepper näkee. Ensimmäisenä avataan ulkoasun määrittelemiseen tarkoitettu xml-tiedosto tässä tapauksessa activity_main.xml (res->layout->activity_main.xml).
Luodaan uloimman Layout:in sisälle ImageView ja Button. Tulevassa kuviossa 8 näkyy esimerkki ImageView:stä ja Button:sta

```
<ImageView
    android:id="@+id/picture_view"
    android:layout_width="300dp"
    android:layout_height="300dp"
    android:layout_above="@+id/take_pic_button"
    android:layout_centerInParent="true"                    />

<Button
    android:id="@+id/take_pic_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="Take a picture"/>
```

Kuvio 8. Esimerkki ImageView:stä ja Button:sta.

Seuraavaksi siirrytään itse ohjelman puolelle eli tässä tapauksessa MainActivity:n. Luodaan sinne luokan sisäisiksi muuttujiksi ImageView ja Button kuten tulevassa kuviossa 9 näkyy.

```
// The button used to start take picture action.
private Button button;
// An image view used to show the picture.
private ImageView pictureView;
```

Kuvio 9. Esimerkki Button:sta ja ImageView:stä

Seuraavaksi tehdään koko homman tärkein vaihe, jota ei mainita netistä löytyvissä esimerkeissä. Eli laitetaan, MainActivityn onCreateen setContentView(R.layout.activity_main) kuten seuraavassa kuviossa 10 näkyy.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Register the RobotLifecycleCallbacks to this Activity.
    QiSDK.register(this, this);
}
```

Kuvio 10. Esimerkki layout:n valitsemisesta.

Ilman edellistä vaihetta ohjelma ei tiedä, että sen pitäisi käyttää jotain tiettyä layout:ia. Seuraavaksi mennään onRobotFocusGained-metodiin, lisätään sinne, mitä napista tapahtuu ja määritetään aikaisemmin luodut button sekä pictureView kuten tulevassa kuviossa 11.

```
private void pepperTakePicture(){
    Future<TakePicture> takePictureFuture = TakePicture-
Builder.with(qiContext).buildAsync();
    // Find the button and the imageView in the onCreate
method
    button= (Button) findViewById(R.id.take_pic_button);
    pictureView = findViewById(R.id.picture_view);
    // Set the button onClick listener.
    button.setOnClickListener(v -> takePicture(takePictureFu-
ture));
}
```

Kuvio 11. Esimerkissä määritetään muuttujia sekä mitä button:sta tapahtuu.

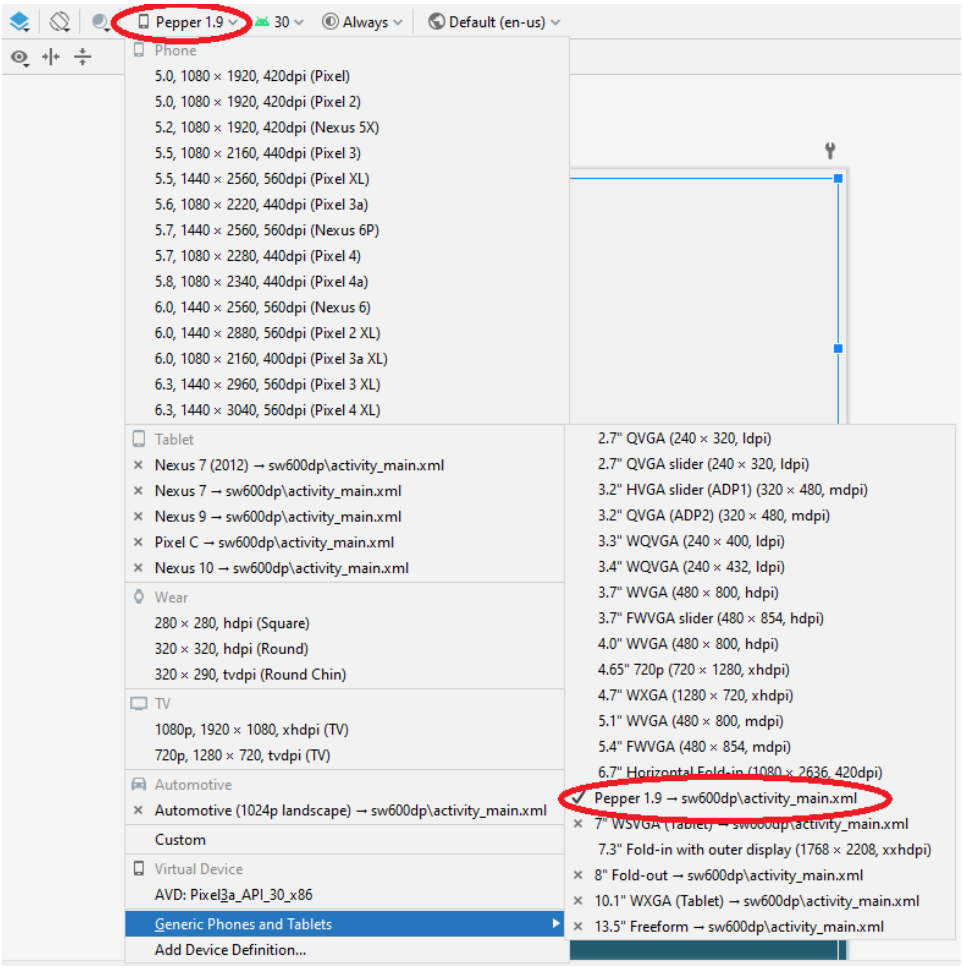
Seuraavaksi luodaan metodi takePicture missä tapahtuu itse kuvan ottaminen ja sen esittäminen ruudulla. Tulevassa kuviossa 12 näkyy, miten se tapahtuu.

```
public void takePicture(Future<TakePicture> takePictureFuture,QiContext
qiContext) {
    String TAG="kuva";
    // Check that the Activity owns the focus.
    if (qiContext == null) {
        return;
    }

    // Disable the button.
    button.setEnabled(false);

    Future<TimestampedImageHandle> timestampedImageHandleFuture = takePic-
tureFuture.andThenCompose(takePicture -> {
        Log.i(TAG, "take picture launched!");
        return takePicture.async().run();
    });timestampedImageHandleFuture.andThenConsume(timestampedImageHandle ->
{
        // Consume take picture action when it's ready
        Log.i(TAG, "Picture taken");
        // get picture
        EncodedImageHandle encodedImageHandle = timestampedImageHan-
dle.getImage();

        EncodedImage encodedImage = encodedImageHandle.getValue();
        Log.i(TAG, "PICTURE RECEIVED!");

        // get the byte buffer and cast it to byte array
        ByteBuffer buffer = encodedImage.getData();
        buffer.rewind();
        final int pictureBufferSize = buffer.remaining();
        final byte[] pictureArray = new byte[pictureBufferSize];
        buffer.get(pictureArray);

        Log.i(TAG, "PICTURE RECEIVED! (" + pictureBufferSize + " Bytes)");
        // display picture
        Bitmap pictureBitmap = BitmapFactory.decodeByteArray(pictureArray,
0, pictureBufferSize);
        runOnUiThread(() -> pictureView.setImageBitmap(pictureBitmap));
    });
}
```

Kuvio 12. Esimerkki kuvan ottamisesta ja käsittelystä Pepper:llä.

Ja siinä on miten Pepper saadaan ottamaan kuva ja näyttämään se tabletillaan.

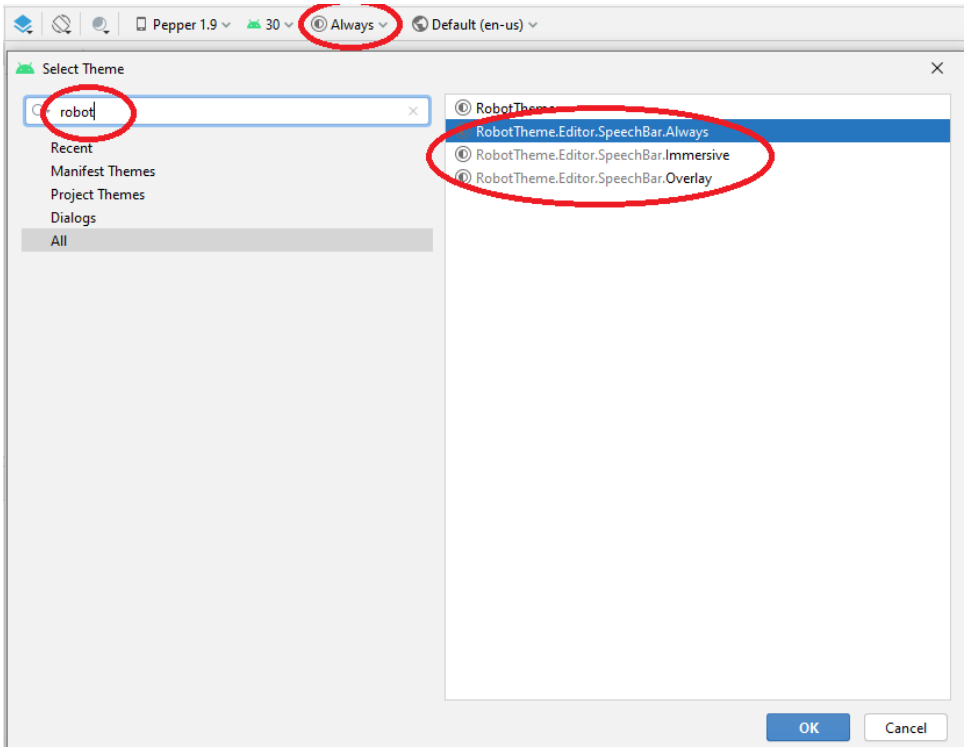
Vielä voitasiin, vähän tutustua tarkemmin mitä layout editorissa on erityistä Pepperiin liittyen. Avaa layout editor kaiksoisklikkaamalla jotakin layouttia (esim. layout -> activity_main.xml). Sieltä löytyy kaksi asiaa, jotka ovat mainitsemisen arvoisia erityisesti Pepperiin littyen. Ensimmäisenä on se, että valitaan esikatseluun oikea laite. Se tapahtuu avaamalla laitevalikko (Device for preview), menemällä siellä generic phones and tablets -kohtaan ja siellä valitsemalla Pepper kuten kuvasta 10 näkyy. Muista laittaa esikatselu landscape-tilaan.



Kuva 10. Oikean laitteen valinta esikatseluun.

Seuraava kohta on, että voidaan valita teema, joka on Pepperille oma. Valitaan theme for preview, sieltä valitaan more themes ja sitten kirjotetaan hakukenttään

robot. Näkyviin pitäisi tulla RobotTheme.Editor.SpeechBar.Always/Immersive/Overlay -teemat.
Kuvasta 11 näkyy mitä pitää valita.



Kuva 11. Teeman valinta

Näistä voi valita sen mitä käyttää sovelluksessa ja esikatselun olisi tarkoitus mukautua valitun teeman mukaiseksi. Jos mikään ei muutu, ei kannata panikoida sillä omassa kokeilussani en ole huomannut mitään eroa mutta, tästä ominaisuudesta mainittiin dokumentaatioissa niin se on nyt tuotu esiin. Jos tahtoo tarkemmin tutustua, miten tabletin layoutteja tehdään, niin kannattaa tutustua androidin ohjeistukseen osoitteessa: https://developer.android.com/studio/write/layout-editor

5. PEPPER JA LIIKE

Pepper pystyy liikkumaan ilman ihmisen avustusta kohtuullista tahtia. Korkeat kynnykset, jyrkät rampit tai muut isot korkeuserot voivat aiheuttaa liikkuessa ongelmia tai jopa aiheuttaa Pepperin kaatumisen, jos niitä ei muista varoa. Lisäksi ennen kuin yrittää saada Pepperin liikkumaan niin pitää muistaa tarkistaa latauspistokkeen luukun tila, kun luukku on auki ihminen voi työntää Pepperiä jos sitä tarvitsee liikuttaa mistä vain syystä mutta Pepper ei pysty itse liikkumaan. Kun luukku on kiinni niin Pepper pystyy liikkumaan itse, mutta sitä ei pysty työntämään.

5.1 Pepper ja yksinkertainen liike

Kun puhutaan Pepperin liikkumisesta täytyy puhua luokasta nimeltä Frame. Frame on se, kuinka Pepper käsittää ympäristöään ja kaikki Pepperin liikkuminen perustuu Framien muokkaamiseen ja siihen että Pepper siirtyy Framistä toiseen. Seuraavaksi käsitellään esimerkki siitä, kuinka Pepper saadaan siirtymään puoli metriä sivullensa. Luodaan MainActivityn alle luokkamuuttujaksi private Goto goto kuten tulevassa kuviossa 13 näkyy.

```
// Store the GoTo action.
private GoTo goTo;
```

Kuvio 13. Luokan sisäinen Goto-muuttuja.

Seuraavaksi laitetaan onRobotFocusGained-metodiin aika paljon koodia, että robotti saadaan liikkumaan mutta koodin itsessään pitäisi olla melko selkeää, kuten kuviosta 14 näkyy.

```
   String TAG="GOTO";
    //get acutation
    Actuation actuation = qiContext.getActuation();
    //get Peppers current location as Frame
    Frame robotFrame=actuation.robotFrame();
    //Make transformation for movement
    Transform transform = TransformBuilder.create().from2DTranslation(0,-
0.5);
    Mapping mapping=qiContext.getMapping();
    //create empty Frame
    FreeFrame targetFrame=mapping.makeFreeFrame();
    //update the empty frame to be target position
    targetFrame.update(robotFrame,transform,0L);
    //tell pepper to figure out movement to target position
    goTo =  GoToBuilder.with(qiContext).withFrame(target-
Frame.frame()).build();
    goTo.addOnStartedListener(() -> Log.i(TAG, "GoTo action started."));
    Future<Void> goToFuture=goTo.async().run();
    goToFuture.thenConsume(future -> {
        if (future.isSuccess()) {
            Log.i(TAG, "GoTo action finished with success.");
        } else if (future.hasError()) {
            Log.e(TAG, "GoTo action finished with error.", future.getError());
        }
    });
    goTo = GoToBuilder.with(qiContext)
            .withFrame(targetFrame.frame())
            .withMaxSpeed(0.2f) // Set the max speed.
            .build();
}
```

Kuvio 14. Esimerkki Pepperin liikkeestä sivulle.

Kun kokeilet tätä muista tarkistaa, että Pepperin latausluukku on kiinni.

5.2 Pepper ja liike kahden pisteen välillä

Seuraavaksi sitten katsotaankin kuinka Pepper saadaan liikkumaan kahden tai useamman sinun valitseman pisteen välillä. Esimerkiksi, jos tahdot että, Pepper opastaa ihmisen, vaikka vessaan. Tässä osiossa näytän esimerkin sovelluksesta, jossa voit tallentaa Pepperin sen hetkisen olinpaikan ja käskeä Pepperin liikkumaan tallennettuun olinpaikkaan. Huomaa, että tallennus ei säily eri käynnistyskertojen välillä. Ensimmäisenä tehdään tabletille näkymä, missä voi tallentaa Pepperin sen hetkisen, sijainnin tai käskeä Pepperin menemään aikaisemmin tallennettuun sijaintiin. Kuviossa 15 näkyy kaikki mitä tarvitsee laittaa xml-tiedostoon.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:focusableInTouchMode="true"
    tools:context=".MainActivity">
    <!--Button to save current location-->
    <Button
        android:id="@+id/save_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="save"
        app:layout_constraintRight_toLeftOf="@+id/add_item_edit"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/goto_button"
        app:layout_constraintTop_toTopOf="parent" />

    <!--Button to goto saved location-->
    <Button
        android:id="@+id/goto_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="go_to_text"
        app:layout_constraintLeft_toLeftOf="@+id/save_button"
        app:layout_constraintRight_toRightOf="@+id/save_button"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/save_button" />

    <!--Text field to give name to current location to be saved as-->
    <EditText
```

```
android:id="@+id/add_item_edit"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="text"
        android:labelFor="@+id/add_item_edit"
        android:hint="location"
        tools:text="Location"
        app:layout_constraintBaseline_toBaselineOf="@+id/save_button"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintLeft_toRightOf="@+id/save_button" />

    <!--Spinner to select saved locations as target-->
    <Spinner
        android:id="@+id/spinner"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="@+id/add_item_edit"
        app:layout_constraintRight_toRightOf="@+id/add_item_edit"
        app:layout_constraintTop_toTopOf="@+id/goto_button"
        app:layout_constraintBottom_toBottomOf="@+id/goto_button"
        app:layout_constraintHorizontal_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Kuvio 15. Asiat mitä tarvitsee laittaa xml tiedostoon sijainnin valintaa ja tallennusta varten.

Seuraavaksi luodaan luokka muuttujia, jotka tarvitaan. Kuvio 16 näkyy tarvittavat luokka muuttujat.

```
private Button gotoButton;
private QiContext qiContext;
private Button saveButton;
private ArrayAdapter<String> spinnerAdapter;
// Store the selected location.
private String selectedLocation;
```

Kuvio 16. Luokka muuttujat, joita tarvitaan sijainnin valintaan, tallennukseen ja muuhun.

Seuraavaksi määritetään onCreatessa mitä kaikissa aikaisemmin luoduissa napeissa ja muissa tapahtuu. Ja sen lisäksi siinä muistetaan valita xml-tiedosto, jossa aikaisemmin laitetut asiat ovat kuten kuviossa 17 näkyy.

```
setContentView(R.layout.activity_main);
final String TAG= "liike";
final EditText addItemEdit = (EditText) findViewById(R.id.add_item_edit);
final Spinner spinner = (Spinner) findViewById(R.id.spinner);

// Save location on save button clicked.
saveButton = (Button) findViewById(R.id.save_button);
saveButton.setOnClickListener(v -> {
    String location = addItemEdit.getText().toString();
    addItemEdit.setText("");
    // Save location only if new.
    if (!location.isEmpty() && !savedLocations.containsKey(location)) {
        spinnerAdapter.add(location);
        saveLocation(location);
    }
});

// Go to location on go to button clicked.
gotoButton = (Button) findViewById(R.id.goto_button);
gotoButton.setOnClickListener(v -> {
    if (selectedLocation != null) {
        gotoButton.setEnabled(false);
        saveButton.setEnabled(false);
        goToLocation(selectedLocation);
    }
});

// Store location on selection.
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedLis-
tener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int posi-
tion, long id) {
        selectedLocation = (String) parent.getItemAtPosition(position);
```

```
        Log.i(TAG, "onItemSelected: " + selectedLocation);
```

```
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        selectedLocation = null;
        Log.i(TAG, "onNothingSelected");
    }
});

// Setup spinner adapter.
spinnerAdapter = new ArrayAdapter<>(this, android.R.layout.simple_spin-
ner_item, new ArrayList<String>());
spinnerAdapter.setDropDownViewResource(android.R.layout.simple_spin-
ner_dropdown_item);
spinner.setAdapter(spinnerAdapter);
```

Kuvio 17. Asiat mitä tarvitsee laittaa onCreateen.

Seuraavaksi voitaisiin käydä läpi metodit, joita aikaisemmassa vaiheessa käytetään aloittaen saveLocationista. Siinä nimensä mukaisesti tallennetaan robotin sen hetkinen sijainti käyttömuistiin myöhempää hyödyntämistä varten kuten kuviosta 18 näkee.

```
void saveLocation(final String location) {
    // Get the robot frame asynchronously.
    Future<Frame> robotFrameFuture = actuation.async().robotFrame();
    robotFrameFuture.andThenConsume(robotFrame -> {
        // Create a FreeFrame representing the current robot frame.
        FreeFrame locationFrame = mapping.makeFreeFrame();
        Transform transform = TransformBuilder.create().fromXTranslation(0);
        locationFrame.update(robotFrame, transform, 0L);

        // Store the FreeFrame.
        savedLocations.put(location, locationFrame);
    });
}
```

Kuvio 18. SaveLocation metodi.

Ja sitten tehdään goToLocation-metodi, joka kertoo Pepperille, että sen pitää mennä johonkin ja minne sen pitää mennä. Kuviossa 19 näkyy kyseinen metodi.

```
void goToLocation(final String location) {
    String TAG="liike";
    // Get the FreeFrame from the saved locations.
    FreeFrame freeFrame = savedLocations.get(location);

    // Extract the Frame asynchronously.
    Future<Frame> frameFuture = freeFrame.async().frame();
    frameFuture.andThenCompose(frame -> {
        // Create a GoTo action.
        goTo = GoToBuilder.with(qiContext)
                .withFrame(frame)
                .build();

        // Display text when the GoTo action starts.
        goTo.addOnStartedListener(() -> Log.i(TAG, "Moving..."));

        // Execute the GoTo action asynchronously.
        return goTo.async().run();
    }).thenConsume(future -> {
        if (future.isSuccess()) {
            Log.i(TAG, "Location reached: " + location);
        } else if (future.hasError()) {
            Log.e(TAG, "Go to location error", future.getError());
        }
    });
}
```

Kuvio 19. goToLocation Metodi.

Seuraavaksi mennään tekemään onRobotFocusGained-metodiin vielä viimeiset tarvittavat toiminnot. Kuviossa 20 näkyy, mitä kaikkea sinne tarvitsee laittaa.

```
// Store the provided QiContext and services.
this.qiContext = qiContext;
actuation = qiContext.getActuation();
mapping = qiContext.getMapping();
waitForInstructions();
```

Kuvio 20. OnRobotuFocusGained metodin sisältö.

Sitten tehdään vielä tuo waitForInstructions-metodi. Siinä vain kerrotaan Pepperille että nyt voi alkaa kuunnella nappien painalluksia. Kuviosta 21 näkyy itse koodi.

```
private void waitForInstructions() {
    Log.i("liike", "Waiting for instructions...");
    runOnUiThread(() -> {
        saveButton.setEnabled(true);
        gotoButton.setEnabled(true);
    });
}
```

Kuvio 21. waitForInstructions-metodin sisältö.

Ja näin, nyt Pepper osaa liikkua sinne tallennettujen pisteiden välillä napin painalluksella. Ja jos Pepperin liikkumisen tai liikuttelun kanssa tulee ongelmia niin muista katsoa ensimmäisenä katso onko lataus pistokkeen kansi oikeassa asennossa.

LOPPUSANAT

Ohjeessa käytiin Pepperin kaikki perustoiminnot pintapuolisesti ja mahdollisimman yksinkertaisesti läpi. Seuraavat vaiheet, kuten vaikka esimerkkien yhdistely tai mikä tahansa muu jää itse ohjeen lukijan tehtäväksi. Lisäksi lisään tähän ithub-linkin mistä löytyy kaikki käyttämäni koodi. Se on ollut vain omassa käytössäni eikä sitä ole tarkemmin testattu. https://github.com/nyyppa/PepperGit.git

LÄHTEET

https://github.com/aldebaran/qisdk-tutorials

https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/ch1_gettingstarted/getting_started.html

https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/ch1_gettingstarted/next_step.html