Git WikiPage

Team blend

# Inleiding:

Dit document bevat de documentatie in de wiki page op github.   
Hierdoor kan het zijn de dat structuur in dit document verwarrend kan zijn.

De Page is bedoel om uitleg te geven hoe de bestaande onderdelen gebruikt kunnen worden om de applicatie verder uit te breiden.

U kunt de wiki page vinden op :

https://github.com/Geddy05/ex-room/wiki

# Home page

Insight is een applicatie die samen met een hologrammen bril die je een beleving geeft bij nieuw producten in de Media Markt.

In deze pagina bespreken we de gebruikte software structuur en hoe dit project je als uitgangspunt kan gebruiken voor je eigen project.

[Install van de app](https://github.com/Geddy05/ex-room/wiki/Install-App)  
[Creeer een nieuwe scene/activity](https://github.com/Geddy05/ex-room/wiki/Creer-een-nieuwe-Scene-Activity)  
[Render een object](https://github.com/Geddy05/ex-room/wiki/Render-een-object)  
[Audioplayer](https://github.com/Geddy05/ex-room/wiki/Audioplayer)  
[Api](https://github.com/Geddy05/ex-room/wiki/Api)

# Audio Player

Tijdens het gebruik van de app word geluid afgespeeld via telefoon of boxen. De app stuurt een verzoek naar de rest-api om een bepaald nummer te spellen. Daarna zal de Rest-api een verzoek sturen naar de audioPlayer.py om de geven nummer te spelen en daarbij word een waarde gestuurd voor de muziekstuk.

Hierdoor word er als eerste de muziek file gekozen aan de hand van de gegeven waarde en dat gebeurt in de functie getMusic(). In de functie play() wordt er gebruik gemaakt van de library pygames, omdat hierdoor een mp3 bestand kan worden afgespeeld.

Als eerste word het music file geladen in de pygame mixer en daarna door pygame.mixer.music.play(). Dit zal niet gelijk gaan werken, omdat er een rust moet zijn tussen het starten en afspelen. Daarvoor word er pygame.time.Clock() gebruikt om te checken of het afspelen nog bezig is en anders wordt er pauze gestart.

def play(self, number):

musicFile = self.getMusic(int(number))

if musicFile != '':

pygame.init()

pygame.mixer.init()

clock = pygame.time.Clock()

pygame.mixer.music.load(musicFile)

pygame.mixer.music.play()

while pygame.mixer.music.get\_busy():

clock.tick(1000)

else:

print('leeg')

Functie stuurt muziek file terug.

def getMusic(self, musicNumber):

music = {0:'',1:'beethovenSymphony.mp3', 2:'beethovenSymphony.mp3'}

return music[musicNumber]

De rest-api stuurt een stop verzoek naar playAudio.py file om het muziek stuk te stopen. Hierbij werd het niet gelijk op dat moment gestopt. Dit kwam doordat het muziek file pas stopt als het gehele stuk is afgespeeld. Dit was ook niet op te lossen door threads of asyncio. Bij beide mogelijkheden worden de wachten processen gestopt, maar de gestarte proces. Die worden pas gestopt na dat het bestand is afgespeeld.

Daardoor is er door het projectgroep bedacht om de playAudio.py file te starten en killen in de rest-api. Daarvoor moest er eerst een argparse worden gemaakt. Daarin zijn de start en stop gedefinieerd. Hierdoor kan de rest-api via sys de file aanspreken.

if \_\_name\_\_ == '\_\_main\_\_':

from argparse import ArgumentParser

from playAudio import playAudio

parser = ArgumentParser(usage="gebruik zo ... ",)

parser.add\_argument("-p","--play", dest="play", type=int, action="store", help="Start the Database")

parser.add\_argument("-s","--stop", dest="stop", action="store\_true", default=False, help="Stop the proces with this argument")

args = parser.parse\_args()

kv = vars(args)

print(args)

# Creer een nieuwe Scene Activity

De huidige applicatie bestaat uit 1 activity die door middel van augmented reality een thema weer rendert op een bepaalde afbeelding. Op het moment dat gewenst is om keuze te kunnen maken in verschillende thema's bieden we de mogelijkheid door een nieuwe instantie te maken van VuforiaController. Op deze pagina wordt u stap voor stap mee genomen door het proces om nieuw activity aan te maken.

zorg er voor dat je activity de vuforiaActivity extends

public class <Your Activity> extends vuforiaActivity {

...

}

in de onCreate() methode nadat de super.onCreate() is aangeroepen Instatieer je een nieuwe instantie van de VuforiaController.

De Contructor heeft een 1 parameter dat is de activity waar het object op getekent moet worden. Om gebruik te kunnen maken van audio optie moet de een AudioPlayer aangemaakt worden met een application object.

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

app = (App) getApplication();

mAudio = new AudioPlayer(app);

startLoadingAnimation();

mVuforiaController = new VuforiaController(this);

}

Vervolgens is aan te raden om de onResume te overschrijven. hier kan de audio opnieuw worden opgestart als de applicatie actief wordt van de achtergrond.

# Install App

Om aan dit project te beginnen worden de volgende stappen verwacht. Als eerste basis programmeer kennis verwacht over:

* Android,
* JPCT-AE,
* Vuforia.

Het project is via GitHub te clone naar u eigen computer. In Androidstudio is mogelijk om de app te runnen en programmeren. Voordat u begint met programmeren, moet als eerste een account worden aangemaakt bij de onderstaande link.<http://dev.vuforia.ptc.velir.com/Tools%20and%20Resources>

Na het aanmaken van een account download u de vufori sdk files en plaats u .so en .jar files in de jniLibs map.  
C:..\AndroidStudioProjects\ex-room\app\src\main\jniLibs

Om te begin met programmeren is het heel belangrijk om de documentatie eerste te lezen. Hierdoor is het makkelijker om de klasse structuur en functie te begrijpen.

Een aandachtspunt is dat de maxSdkVersion van android 23 mag zijn, omdat vuforia bij hogere android versies niet werkt. Dit was geconcludeerd op 5 mei 2016.

API

# Render een object

In het hoofdstuk app install wordt er basis kennis verwacht bij het project. In dit hoofdstuk zal er een uitleg worden gegeven over het aanmaken van object. Hiervoor wordt de libarry JPCT-ae gebruikt, maar in de code word het ook wel opengl genoemd. Als eerste word een uitgelegd gegeven over waar het proces van object rendering word gestart. Daarna zal het proces beschreven wordt van het aanmaken van een scene. Als laatste word er nog gekeken naar het aanroepen van het object en het wijzigen.

In de klasse VuforiaController worden verschillende processen gestart zoals tracker van het patroon gestart en gestopt, maar ook het object renderen. In dit zal een westernscene gerenderd worden. De functie initApplicationAR() start op het moment als een patroon word herkend. In de functie wordt het object westernscene aangemaakt en doorgestuurd. Hierdoor wordt klasse westernscene gestart en word het object doorgestuurd naar opengl libarry. Hieronder is het stukje code ervan te zien.

Class vuforia/VuforiaController.java

@Override

private void initApplicationAR() {

int depthSize = 16;

int stencilSize = 0;

boolean translucent = Vuforia.requiresAlpha();

mGlView = new ExRoomGL(mActivity);

mGlView.init(translucent, depthSize, stencilSize);

mRenderer = new WesternScene(mActivity);

mGlView.setRenderer(mRenderer);

}

In de klasse westernscene wordt implementatie gedaan van de klasse GLSurfaceView.Renderer. Hiervoor werd het object westernscene al meegestuurd. Door implementatie is het mogelijk om altijd tijden het proces wijzigingen en update uit te voeren op het object. Je bent wel verplicht om in het object de gegeven functie van de implementatie aan te roepen. Hierdoor worden de drie functies getoond.

public interface Renderer {

void onSurfaceCreated(GL10 gl, EGLConfig config);

void onSurfaceChanged(GL10 gl, int width, int height);

void onDrawFrame(GL10 gl);

}

De functienamen spreken een beetje voor zich, maar elke functie word even uitleg. De functie onDrawFrame() wordt het object rendering gestart. Daarbij moet gedacht worden aan het teken van de wereld (world.renderScene(fb),world.draw(fb)). Het tonen van de wereld op het scherm gebeurd bij de functie onSurfaceCreated(). De laatste functie onSurfaceChanged() zorgt er voor dat bij verandering van camera positie het object geupdated word. Hierbij moet er gedacht worden aan verplaatsing van het object of rotatie.

@Override

public void onDrawFrame(GL10 gl) {

if (!mIsActive)

return;

renderFrame();

updateCamera();

world.renderScene(fb);

world.draw(fb);

fb.display();

}

@Override

public void onSurfaceCreated(GL10 gl, EGLConfig config) {

initRendering();

vuforiaAppSession.onSurfaceCreated();

}

@Override

public void onSurfaceChanged(GL10 gl, int width, int height) {

if (fb != null) {

fb.dispose();

}

fb = new FrameBuffer(width, height);

Config.viewportOffsetAffectsRenderTarget = true;

updateRendering(width, height);

vuforiaAppSession.onSurfaceChanged(width, height);

}

Hiervoor is er uitleg gegeven over het starten van het rendering van de wereld. Nu zal het maken van de wereld/scene worden toegelicht.

Als eerste zijn er aantal files nodig voor het creëren van object. Hieronder zal een huisje worden aangemaakt. Je hebt drie soorten files nodig .jpg, .obj en .mtl file. In de .jpg file staat huid van huisje als afbeelding opgeslagen. In .obj file worden texture matrix waarden opgeslagen en daardoor is het straks mogelijk om de opgeslagen afbeelding op de .jpg file precies om het object te plaatsen. Het object heeft ook een body nodig en dat staat opgeslagen in het .mtl file.

Lijst van benodigde files

Cottage Texture.jpg

Snow covered CottageOBJ.obj

Snow covered CottageOBJ.mtl

De files worden in een inputstream gezet. In de functies loadModel() en createObject() worden de inputstreams gebruikt om de objecten te creëren. De createObject() zorgt er ook voor dat aan het object de texture (.jpg file) word toegevoegd. Elke object moet een uniek inputstream hebben, omdat als een inputstream wordt hergebruikt bij een andere object zullen de objecten niet getoond worden. Dit komt doordat inputstream dan al in gebruik is voor het eerste object. Hieronder zijn de functies te zien.

public static Object3D loadModel(String nameObject, InputStream streamObj, InputStream streamMtl, int texSampler2DHandle) {

Object3D[] model = Loader.loadOBJ(streamObj,streamMtl, 1.5f);

Object3D o3d = new Object3D(0);

o3d = Object3D.mergeObjects(o3d, createObject(model));

o3d.strip();

o3d.build();

return o3d;

}

public static Object3D createObject(Object3D[] model) {

Object3D temp = null;

for (int i = 0; i < model.length; i++) {

temp = model[i];

temp.setCenter(SimpleVector.ORIGIN);

temp.setTexture("texture");

temp.setRotationMatrix(new Matrix());

}

return temp;

}

Na het creëren van de objecten zullen ze worden toevoegt in de WesternScene() functie world (scene/wereld). Hierdoor is een scene gecreëerd en zal het worden doorgestuurd naar de klasse GLSurfaceView.Renderer om het te renderen/weergeven aan de gebruiker. In de functie WesternScene() worden de texture en inputstreams aangemaakt. Daarna worden de variables doorgestuurd naar de loadModel() functie. In de WesternScene() functie kan voor het laatste een verplaatsing of aanpassing worden meegeven aan het object. Door translate() aan te roepen of het object en X,Y,Z-coördinaten mee te geven. In de documentatie van JPCT-ae staan nog vele andere functies voor verplaatsing of rotatie.

Class utils/WesternScene.java

public WesternScene(vuforiaActivity activity) {

mActivity = activity;

App app =(App) mActivity.getApplication();

vuforiaAppSession = app.vuforiaSession;

world= new World();

world.setAmbientLight(100, 100, 100);

world.setClippingPlanes(2.0f, 3000.0f);

sun = new Light(world);

sun.setIntensity(250, 250, 250);

texSampler2DHandle = GLES20.glGetUniformLocation(shaderProgramID, "texSampler2D");

try {

TextureManager.getInstance().addTexture("texture" ,new Texture(mActivity.getAssets().open("Cottage Texture.jpg")));

InputStream streamObj = mActivity.getAssets().open("Snow covered CottageOBJ.obj");

InputStream streamMtl = mActivity.getAssets().open("Snow covered CottageOBJ.mtl");

InputStream streamObj2 = mActivity.getAssets().open("Snow covered CottageOBJ.obj");

InputStream streamMtl2 = mActivity.getAssets().open("Snow covered CottageOBJ.mtl");

home1 = null;

home2 = null;

home1 = loadModel("home1", streamObj, streamMtl,texSampler2DHandle);

home1.translate(100.0f, 0.0f, 0.0f);

home1.rotateX(30.0f);

home2 = loadModel("home2", streamObj2, streamMtl2,texSampler2DHandle);

home2.translate(0.0f, 0.0f, 0.0f);

home2.rotateX(30.0f);

world.addObject(home1);

world.addObject(home2);

world.buildAllObjects();

}catch (Exception e){

System.out.println("Not rendering obj");

System.out.println(e);

}

cam = world.getCamera();

MemoryHelper.compact();

}

# Installatie & gebruik

Installeer Python 3.5.x of later

<https://www.python.org/downloads/>

Installeer **pip**

<https://pip.pypa.io/en/stable/installing/>

Installeer met pip de volgende framework: virtualenv

<https://virtualenv.pypa.io/en/stable/>

### Maak een nieuwe "virtual environment" aan door middel van**virtualenv**

virtualenv MAP\_NAAM

en vervolgens activeer je de environment:

. /MAP\_NAAM/bin/activate

Kopieer alle bestanden van de REST API in de environment en installeer alle **dependencies**.

pip install -r requirements.txt

# Zelf aanpassingen maken

## De API kan gebruikt worden door middel van de volgende calls:

* GET
* POST
* PUT
* DELETE
* HEAD

## Routes toevoegen

Een nieuwe Route toevoegen wordt gedaan in **api.py**

api.add\_resource(CONTROLLER\_CLASS\_NAME, '/path/<string:path>', endpoint='path')

Of om een endpoint toe te voegen waar een waarde aan meegegeven moet worden

api.add\_resource(CONTROLLER\_CLASS\_NAME, '/path/<string:path>/<int:id>')

## Controller toevoegen

Dit wordt gedaan in de map: **Controllers**

Maak hier een nieuwe bestand aan en hou rekening met de **Naming Convention**. Als voorbeeld:

verhicles\_controller.py

Per Controller implement de call functies

class VerhicleController(Controller):

def get(self):

...

return output\_json(...)

def post(self):

...

return output\_json(...)

def put(self):

...

return output\_json(...)

def delete(self):

...

return output\_json(...)

## Resources / Models

Omdat dit project momenteel geen database nodig had wilt niet zeggen dat er geen rekening mee gehouden moet worden. Momenteel wordt er gebruik gemaakt van Resources. Dit zijn Classes die de functies uitvoeren. Dit wordt ook wel **Services** genoemd.

Models wordt nog toegevoegd. Resources kunnen aangemaakt worden in het mapje**Resources**

## Schema's

Maak gebruik van Schema's om je PATHS en Functies aan elkaar te koppelen. De schema's kan aangepast worden in het gelijknamige bestandje **schema.py**.

PATH (count) is gekoppeld aan de FUNCTIE (unit\_count)

PATH\_SPEAKERS = {

'count': 'unit\_count',

'calibrate': 'calibrate',

'volume': 'volume',

'unit': 'unit',

'state': 'unit\_states',

}

Input & Response kunnen beveiligd worden door deze verplichte type aan te geven. Dit gebeurt door middel van **fields**

from flask\_restful import fields

MEDIA\_RESPONSE\_FIELDS = {

'play': {

'status\_code', fields.Integer

},

'stop': {

'status\_code', fields.Integer

},

'pause': {

'status\_code', fields.Integer

},

'is\_playing': {

'is\_playing': fields.Boolean

},

'is\_stopped': {

'is\_stopped': fields.Boolean

},

'is\_pause': {

'is\_pause': fields.Boolean

}

}

is\_playing kan alleen antwoorden in de type Boolean.