==============================================

Button.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import android.content.Intent

class Button {

var boxOffset = BoxOffset()

var originalOffsetX = 0f

var originalOffsetY = 0f

var label = ""

var column = 0

var row = 0

var intent: Intent? = null

//var icon: Icon

}==============================================

ButtonParameters.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import androidx.compose.ui.unit.Density

import androidx.compose.ui.unit.Dp

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

object ButtonParameters {

val buttonGapIndexMax = 5

val buttonGapPercentage = arrayOf(

10.0f,

20.0f,

30.0f

)

val buttonSizeIndexMax = 5

var buttonHeightsDp = emptyArray<Dp>()

val buttonHeightsPx = arrayOf(

//102.7f,

//117.1f,

136.2f,

162.7f,

202.1f,

266.7f,

391.8f

)

var buttonWidthsDp = emptyArray<Dp>()

val buttonWidthsPx = arrayOf(

//102.7f,

//117.1f,

136.2f,

162.7f,

202.1f,

266.7f,

391.8f

)

val screenButtonColumns = arrayOf(

4,

3,

3,

3,

3,

2,

2

)

val screenButtonRows = arrayOf(

5,

4,

4,

4,

3,

3,

3

)

val buttonRoundedSizes = arrayOf( // 0.75

//6.sp,

//8.sp,

9.dp,

12.dp,

16.dp,

21.dp,

28.dp

)

val buttonTextSizes = arrayOf( // 0.75

//6.sp,

//8.sp,

11.sp,

14.sp,

19.sp,

26.sp,

34.sp

)

fun init(density: Density) {

for (buttonWidthPx in buttonWidthsPx) buttonWidthsDp +=

with(density) { buttonWidthPx.toDp() }

for (buttonHeightPx in buttonHeightsPx) buttonHeightsDp +=

with(density) { buttonHeightPx.toDp() }

}

fun buttonWidthDpToColumns(screenWidthDp: Dp, buttonWidthDp: Dp, gapPercentage: Float): Int =

(screenWidthDp / (buttonWidthDp \* (1 + 2 \* gapPercentage))).toInt()

fun columnsToButtonWidthDp(screenWidthDp: Dp, buttonColumns: Int, gapPercentage: Float) =

screenWidthDp / (buttonColumns \* (1 + 2 \* gapPercentage))

fun buttonHeightDpToRows(screenHeightDp: Dp, buttonHeightDp: Dp, gapPercentage: Float): Int =

(screenHeightDp / (buttonHeightDp \* (1 + 2 \* gapPercentage))).toInt()

fun columnsToButtonHeightDp(screenHeightDp: Dp, buttonRows: Int, gapPercentage: Float) =

screenHeightDp / (buttonRows \* (1 + 2 \* gapPercentage))

}==============================================

LogPointerEvents.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.offset

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.setValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.clip

import androidx.compose.ui.input.pointer.PointerEventType

import androidx.compose.ui.input.pointer.pointerInput

import androidx.compose.ui.layout.onGloballyPositioned

import androidx.compose.ui.platform.LocalConfiguration

import androidx.compose.ui.platform.LocalDensity

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.IntOffset

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import kotlin.math.roundToInt

@Composable

fun LogPointerEvents(

filter: PointerEventType? = null

) {

val configuration = LocalConfiguration.current

val density = LocalDensity.current

var state by remember { mutableStateOf(State(configuration, density)) }

var noClicks by remember { mutableStateOf(0) }

val stateChanged: () -> Unit = {

state = state.copy(clicked = ++noClicks)

}

//val context = LocalContext.current // Get the current context

state.init(configuration, density)

fun formatDecimals(number: Float, decimals: Int) = String.format("%.${decimals}f", number)

Column(

modifier = Modifier.fillMaxSize(), // Makes the Column take the full width

horizontalAlignment = Alignment.CenterHorizontally // Centers children horizontally

) {

Text("Adaptive UI", textAlign = TextAlign.Center, fontSize = 48.sp)

Text("screen size ${state.screen.width.dp} x ${state.screen.height.dp}", fontSize = 12.sp)

Text(

"screen size ${state.screen.width.px}.px x ${state.screen.height.px}.px",

fontSize = 12.sp

)

//Text("box size ${state.boxWidthDp} x ${state.boxHeightDp}", fontSize = 12.sp)

//Text("box size ${state.boxWidthPx}.px x ${state.boxHeightPx}.px", fontSize = 12.sp)

Text(

"button size ${state.getButtonWidthDp()} x ${state.getButtonHeightDp()}",

fontSize = 12.sp

)

Text(

"button size ${state.getButtonWidthPx()}.px x ${state.getButtonHeightPx()}.px",

fontSize = 12.sp

)

Text(

"offsetX ${formatDecimals(state.boxOffset.x, 1)}.px " +

"offsetY ${

formatDecimals(state.boxOffset.y, 1)

}.px",

fontSize = 12.sp

)

if (!state.showDialog) {

Column() {

Box(

modifier = Modifier

.fillMaxWidth()

.height(600.dp)

.background(MaterialTheme.colorScheme.secondaryContainer)

.onGloballyPositioned { coordinates ->

state.box = Dimensions(

Extent.pxToExtent(density, coordinates.size.width.toFloat()),

Extent.pxToExtent(density, coordinates.size.height.toFloat()),

)

}

.pointerInput(filter) {

awaitPointerEventScope {

while (true) {

val event = awaitPointerEvent()

// handle pointer event

if (filter == null || event.type == filter) {

PointerEvents.onBoxPointerEvent(

event,

state,

stateChanged

)

}

}

}

}

) {

// The Button composable placed inside the Box

Box(

contentAlignment = Alignment.Center,

modifier = Modifier

.offset {

IntOffset(

state.boxOffset.x.roundToInt(),

state.boxOffset.y.roundToInt()

)

}

.size(

state.getButtonWidthDp(),

state.getButtonHeightDp()

)

//.align(Alignment.Center) // Center the button within the Box

.clip(RoundedCornerShape(ButtonParameters.buttonRoundedSizes[state.buttonSizeIndex]))//28.dp)) // Apply rounded corners

.background(MaterialTheme.colorScheme.primary)

.pointerInput(filter) {

awaitPointerEventScope {

while (true) {

val event = awaitPointerEvent()

// handle pointer event

if (filter == null || event.type == filter) {

PointerEvents.onButtonPointerEvent(

event,

state,

stateChanged

)

}

}

}

}

) {

Text(

text = "Click Me",

color = MaterialTheme.colorScheme.onPrimary,

fontSize = ButtonParameters.buttonTextSizes[state.buttonSizeIndex]

)

}

}

Row() {

MaximizeButton(state.maximizeButton, stateChanged)

MinimizeButton(state.minimizeButton, stateChanged)

IncrementButton(state.incrementButton, stateChanged)

DecrementButton(state.decrementButton, stateChanged)

ExpandButton(state.incrementButton, stateChanged)

CompressButton(state.decrementButton, stateChanged)

}

}

} else {

ConfirmButtonTapDialog(state.onConfirm, state.onDismiss, stateChanged)

}

}

}

==============================================

PointerEvents.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import android.util.Log

import androidx.compose.ui.input.pointer.PointerEvent

import androidx.compose.ui.input.pointer.PointerEventType

object PointerEvents {

fun log(message: String) = Log.d("LogPointerEvents", message)

val onBoxPointerEvent: (

PointerEvent,

State,

stateChanged: () -> Unit

) -> Unit =

{ event,

state,

stateChanged

->

log("box ${event.type}, ${event.changes.first().position}, ${event.changes.first().pressure}, ${event.changes.first().uptimeMillis}")

when (event.type) {

PointerEventType.Press -> {

if (state.buttonMoving) {

state.setFirstPosition(event.changes.first().position)

}

when (state.pointerEventState) {

PointerEventState.START -> {

state.setPointerEventState(PointerEventState.BOX\_PRESS)

}

PointerEventState.BUTTON\_PRESS -> {

state.setPointerEventState(PointerEventState.BUTTON\_BOX\_PRESS)

}

PointerEventState.BUTTON\_RELEASE -> {

//setPointerEventState(PointerEventState.BUTTON\_BOX\_PRESS)

}

PointerEventState.BOX\_PRESS -> {

}

else -> {

log("unexpected box event type ${event.type} in state $state.pointerEventState")

state.setPointerEventState(PointerEventState.START)

}

}

}

PointerEventType.Move -> {

if (state.buttonMoving) {

state.setChangePosition(event.changes.first())

stateChanged()

}

}

PointerEventType.Release -> {

if (state.buttonMoving) {

state.setFinalPosition()

state.setButtonMoving(false)

}

when (state.pointerEventState) {

PointerEventState.BOX\_PRESS -> {

// pointerEventState = PointerEventState.BOX\_TAP // PointerEventState.BOX\_RELEASE

state.setPointerEventState(PointerEventState.START)

state.incrementButtonSize()

stateChanged()

}

PointerEventState.BUTTON\_RELEASE -> {

//setPointerEventState(PointerEventState.BUTTON\_BOX\_RELEASE)

//setPointerEventState(PointerEventState.BUTTON\_TAP)

state.setPointerEventState(PointerEventState.START)

if (state.setButtonRelease(event.changes.first().uptimeMillis)) {

if (state.buttonSizeIndex > (ButtonParameters.buttonSizeIndexMax / 2)) {

state.setShowDialog()

stateChanged()

} else {

state.decrementButtonSize()

stateChanged()

}

}

}

else -> {

log("unexpected box event type ${event.type} in state $state.pointerEventState")

state.setPointerEventState(PointerEventState.START)

}

}

}

PointerEventType.Move -> {}

else ->

log("unexpected box event type ${event.type}")

}

}

val onButtonPointerEvent: (

PointerEvent,

State,

stateChanged: () -> Unit

) -> Unit =

{ event,

state,

stateChanged

->

log("button ${event.type}, ${event.changes.first().position}, ${event.changes.first().pressure}, ${event.changes.first().uptimeMillis} ")

when (event.type) {

PointerEventType.Press -> {

state.setButtonMoving(true)

when (state.pointerEventState) {

PointerEventState.START -> {

state.setPointerEventState(PointerEventState.BUTTON\_PRESS)

state.setButtonPress(event.changes.first().uptimeMillis)

}

PointerEventState.BUTTON\_PRESS -> {

}

else -> {

log("unexpected box event type ${event.type} in state $state.pointerEventState")

state.setPointerEventState(PointerEventState.BUTTON\_PRESS)

}

}

}

PointerEventType.Release -> {

when (state.pointerEventState) {

PointerEventState.BUTTON\_BOX\_PRESS -> {

state.setPointerEventState(PointerEventState.BUTTON\_RELEASE)

}

else -> {

log("unexpected box event type ${event.type} in state $state.pointerEventState")

state.setPointerEventState(PointerEventState.START)

}

}

}

PointerEventType.Move -> {}

else ->

log("unexpected button event type ${event.type}")

}

}

}

==============================================

PointerEventState.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

enum class PointerEventState {

START,

BOX\_PRESS,

BOX\_RELEASE,

BOX\_TAP,

BUTTON\_PRESS,

BUTTON\_BOX\_PRESS,

BUTTON\_BOX\_RELEASE,

BUTTON\_RELEASE,

BUTTON\_TAP

}==============================================

State.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import android.content.res.Configuration

import androidx.compose.ui.geometry.Offset

import androidx.compose.ui.input.pointer.PointerInputChange

import androidx.compose.ui.unit.Density

import androidx.compose.ui.unit.Dp

import androidx.compose.ui.unit.dp

import kotlin.math.max

import kotlin.math.min

data class Extent(val dp: Dp = 0.dp, val px: Float = 0f) {

companion object {

fun dpToExtent(density: Density, dp: Dp) =

Extent(dp, with(density) { dp.toPx() })

fun pxToExtent(density: Density, px: Float) =

Extent(with(density) { px.toDp() }, px)

}

}

data class Dimensions(val width: Extent, val height: Extent)

data class BoxOffset(var x: Float = 0f, var y: Float = 0f)

data class State(

val configuration: Configuration,

val density: Density,

var clicked: Int = 0,

var pointerEventState: PointerEventState = PointerEventState.START,

var buttonPadding: Dp = 0.dp,

var previousPosition: Offset = Offset.Zero,

var showDialog: Boolean = false,

var buttonGapIndex: Int = 0,

var buttonMoving: Boolean = false,

var buttonPressMillis: Long = 0L,

val buttonTapThresholdMillis: Int = 250,

var buttonSizeIndex: Int = 0,

var gapPercentage: Float = 0.25f,

val screen: Dimensions = Dimensions(

Extent.dpToExtent(density, configuration.screenWidthDp.dp),

Extent.dpToExtent(density, configuration.screenHeightDp.dp)

),

var boxOffset: BoxOffset = BoxOffset(),

var box: Dimensions = Dimensions(

Extent(),

Extent()

),

var first: Boolean = true

) {

fun getButtonGapPercentage() = ButtonParameters.buttonGapPercentage[buttonGapIndex]

fun getButtonHeightDp() = ButtonParameters.buttonHeightsDp[buttonSizeIndex]

fun getButtonHeightPx() = ButtonParameters.buttonHeightsPx[buttonSizeIndex]

fun getButtonWidthDp() = ButtonParameters.buttonWidthsDp[buttonSizeIndex]

fun getButtonWidthPx() = ButtonParameters.buttonWidthsPx[buttonSizeIndex]

fun getScreenButtonColumn() = ButtonParameters.screenButtonColumns[buttonSizeIndex]

fun getScreenButtonRow() = ButtonParameters.screenButtonRows[buttonSizeIndex]

fun getButtonRoundedSize() = ButtonParameters.buttonRoundedSizes[buttonSizeIndex]

fun getButtonTextSize() = ButtonParameters.buttonTextSizes[buttonSizeIndex]

fun extentFromDp(dp: Dp) = Extent(dp, with(density) { dp.toPx() })

fun extentFromPx(px: Float) = Extent(with(density) { px.toDp() }, px)

fun init(configuration: Configuration, density: Density) {

ButtonParameters.init(density)

if (first) {

boxOffset.x = screen.width.px / 2 - getButtonWidthPx() / 2

//offsetY = screenHeightPx / 2 - getButtonHeightPx() / 2

boxOffset.y = 675 - getButtonHeightPx() / 2

first = false

}

}

// common

val setButtonSizeIndex: (Int) -> Unit =

{ newButtonSizeIndex ->

buttonSizeIndex = newButtonSizeIndex

boxOffset.x = min(

boxOffset.x,

box.width.px - getButtonWidthPx()

)

boxOffset.y = min(

boxOffset.y,

box.height.px - getButtonHeightPx()

)

}

val setPointerEventState: (PointerEventState) -> Unit =

{ newPointerEventState -> pointerEventState = newPointerEventState }

// other

val decrementButtonSize: () -> Unit = {

if (buttonSizeIndex > 0)

setButtonSizeIndex(buttonSizeIndex - 1)

}

val decrementButton: () -> Unit = {

decrementButtonSize()

}

val incrementButtonSize: () -> Unit = {

if (buttonSizeIndex < ButtonParameters.buttonSizeIndexMax - 1)

setButtonSizeIndex(buttonSizeIndex + 1)

}

val incrementButton: () -> Unit = {

incrementButtonSize()

}

val maximizeButton: () -> Unit = {

setButtonSizeIndex(ButtonParameters.buttonSizeIndexMax - 1)

}

val minimizeButton: () -> Unit = {

setButtonSizeIndex(0)

}

val onConfirm: () -> Unit = {

decrementButton()

showDialog = false

setPointerEventState(PointerEventState.START)

}

val onDismiss: () -> Unit = {

incrementButton()

showDialog = false

setPointerEventState(PointerEventState.START)

}

val setButtonMoving: (Boolean) -> Unit = { newButtonMoving ->

buttonMoving = newButtonMoving

}

val setButtonPress: (Long) -> Unit =

{ newButtonPressMillis ->

buttonPressMillis = newButtonPressMillis

}

val setButtonRelease: (Long) -> Boolean =

{ buttonReleaseMillis ->

if (buttonReleaseMillis - buttonPressMillis < buttonTapThresholdMillis) {

buttonPressMillis = 0

true

} else false

}

val setChangePosition: (PointerInputChange) -> Unit =

{ change ->

// Calculate the difference from the previous position

val deltaX = change.position.x - previousPosition.x

val deltaY = change.position.y - previousPosition.y

// Update boxOffset.x and offsetY based on the drag amount (or delta from previous)

boxOffset.x += deltaX

boxOffset.x = max(boxOffset.x, 0f)

boxOffset.x = min(

boxOffset.x,

box.width.px - getButtonWidthPx()

)

boxOffset.y += deltaY

boxOffset.y = max(boxOffset.y, 0f)

boxOffset.y = min(

boxOffset.y,

box.height.px - getButtonHeightPx()

)

// Update previous position for the next onDrag call

previousPosition = change.position

change.consume()

}

val setFinalPosition: () -> Unit =

{

previousPosition = Offset.Zero

}

val setFirstPosition: (Offset) -> Unit =

{ startOffset ->

previousPosition = startOffset

}

val setShowDialog: () -> Unit =

{ showDialog = true }

}

==============================================

UIControls.kt

==============================================

package com.toddkushnerllc.android\_adaptive\_ui

import androidx.compose.material3.AlertDialog

import androidx.compose.material3.Button

import androidx.compose.material3.ButtonDefaults

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Text

import androidx.compose.material3.TextButton

import androidx.compose.runtime.Composable

import androidx.compose.ui.unit.sp

@Composable

fun ConfirmButtonTapDialog(

onConfirm: () -> Unit,

onDismiss: () -> Unit,

stateChanged: () -> Unit

) {

AlertDialog(

onDismissRequest = onDismiss, // Called when the user dismisses the dialog (e.g., taps outside)

title = { Text(text = "Confirmation", color = MaterialTheme.colorScheme.primary) },

text = { Text(text = "Ok to run command", color = MaterialTheme.colorScheme.secondary) },

confirmButton = {

TextButton(onClick = {

onConfirm()

stateChanged()

}

) {

Text("OK")

}

},

dismissButton = {

TextButton(

onClick =

{

onDismiss()

stateChanged()

}) {

Text("Cancel")

}

}

)

}

@Composable

fun MaximizeButton(

maximizeButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

maximizeButton()

stateChanged()

},

colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u2191", color = MaterialTheme.colorScheme.onPrimary,

fontSize = 36.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_upward\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Maximize button"

)

\*/

}

}

@Composable

fun MinimizeButton(

minimizeButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

minimizeButton()

stateChanged()

}, colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u2193",

color = MaterialTheme.colorScheme.onPrimary,

fontSize = 36.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_downward\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Reset button"

)

\*/

}

}

@Composable

fun IncrementButton(

incrementButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

incrementButton()

stateChanged()

},

colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u002B", color = MaterialTheme.colorScheme.onPrimary,

fontSize = 36.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_plus\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Grow button"

)

\*/

}

}

@Composable

fun DecrementButton(

decrementButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

decrementButton()

stateChanged()

},

colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u002D", color = MaterialTheme.colorScheme.onPrimary,

fontSize = 36.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_minus\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Shrink button"

)

\*/

}

}

@Composable

fun ExpandButton(

expandButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

expandButton()

stateChanged()

},

colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u02c2 \u02c3", color = MaterialTheme.colorScheme.onPrimary,

fontSize = 30.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_expand\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Shrink button"

)

\*/

}

}

@Composable

fun CompressButton(

compressButton: () -> Unit,

stateChanged: () -> Unit

) {

Button(

onClick = {

compressButton()

stateChanged()

},

colors = ButtonDefaults.buttonColors(

containerColor = MaterialTheme.colorScheme.tertiary, // Sets the background color of the button

contentColor = MaterialTheme.colorScheme.onTertiary // Sets the color of the text/content inside the button

)

) {

Text(

text = "\u02c3 \u02c2", color = MaterialTheme.colorScheme.onPrimary,

fontSize = 30.sp

)

/\*

Image(

painter = painterResource(id = R.drawable.arrow\_compress\_24dp),//48dp), // Assuming "my\_image.png" was imported

contentDescription = "Shrink button"

)

\*/

}

}