Prototyping With Jetpack Compose

Ben Oberkfell
Chicago Roboto 2020

About Me

- Android Engineer at the New York Times since 2017
- Having fun on the NYT Games Team
- @benlikestocode

My Journey into Compose


ER WEEK

The Mini Crossword

5x5 grid

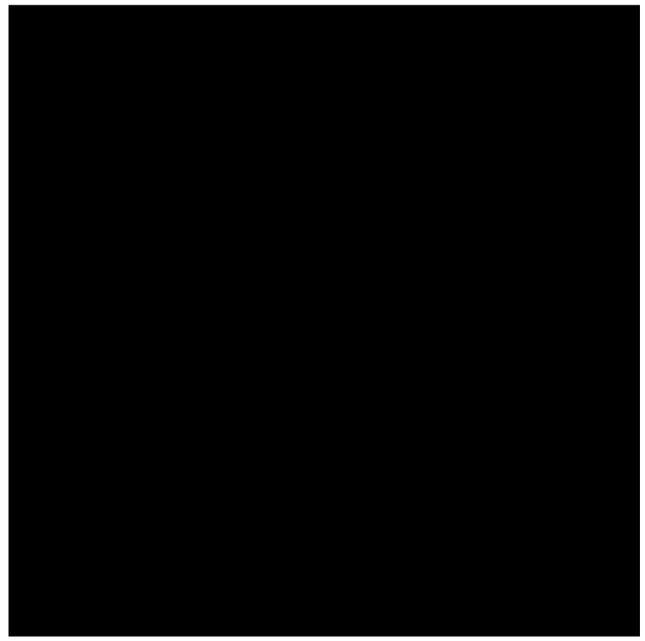
Easy Solve

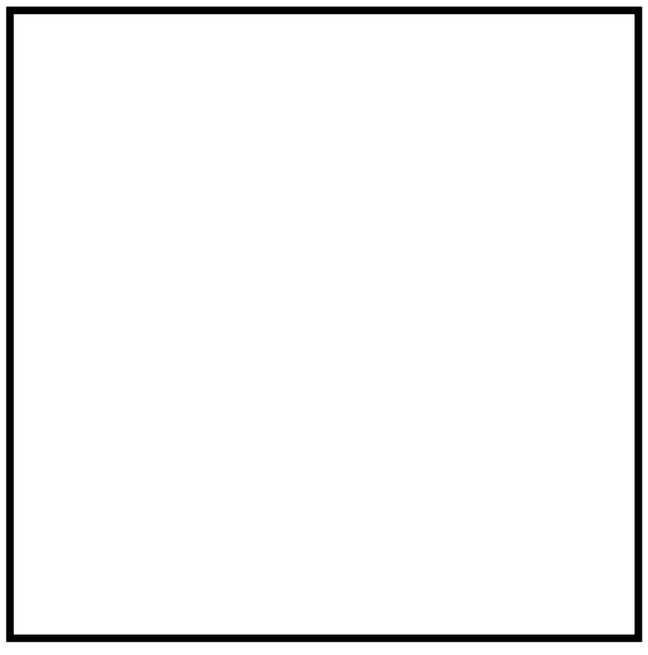
Free to Play

Fewer/no “special” puzzles


9:00
LTE
1:14
1 S
2
3 I
4
5 K
T
6
A
C
7 U
T
O
H
8 R E L Y
Nevada neighbor
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M


9:00
LTE
1:14
1 S
2
3 I
4
5 K
T
6
A
C
7 U
T
O
H
8 R
E
L
Y
Nevada neighbor
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M





Black & White Squares

```
@Composable
fun ColoredSquareBackground(
        color: Color,
        modifier: Modifier) {
    Box(
            modifier = modifier,
            backgroundColor = color,
            shape = RectangleShape
    )
}

@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.White,
            modifier = modifier)
}

@Composable
fun BlackSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.Black,
            modifier = modifier)
}
```

Black & White Squares

```
@Composable
fun ColoredSquareBackground(
        color: Color,
        modifier: Modifier) {
    Box(
            modifier = modifier,
            backgroundColor = color,
            shape = RectangleShape
    )
}
```

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.White,
            modifier = modifier)
}

@Composable
fun BlackSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.Black,
            modifier = modifier)
}
```

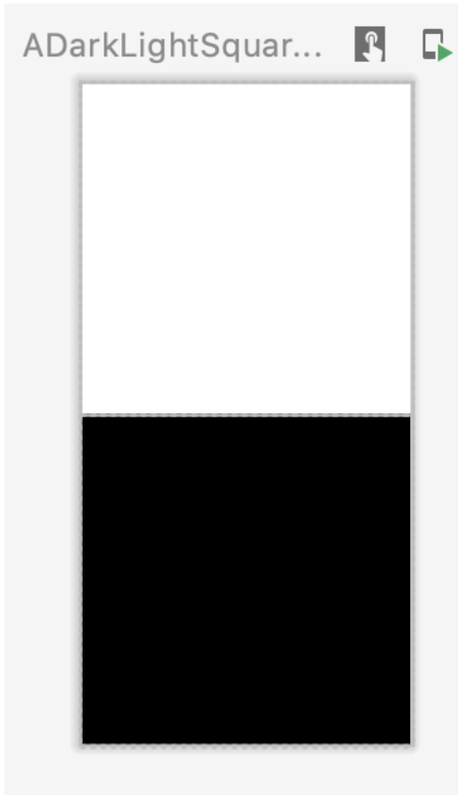
Black & White Squares

```
@Composable
fun ColoredSquareBackground(
        color: Color,
        modifier: Modifier) {
    Box(
            modifier = modifier,
            backgroundColor = color,
            shape = RectangleShape
    )
}
```

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.White,
            modifier = modifier)
}

@Composable
fun BlackSquare(modifier: Modifier = Modifier.fillMaxSize()) {
    ColoredSquareBackground(
            color = Color.Black,
            modifier = modifier)
}
```

Black & White Squares

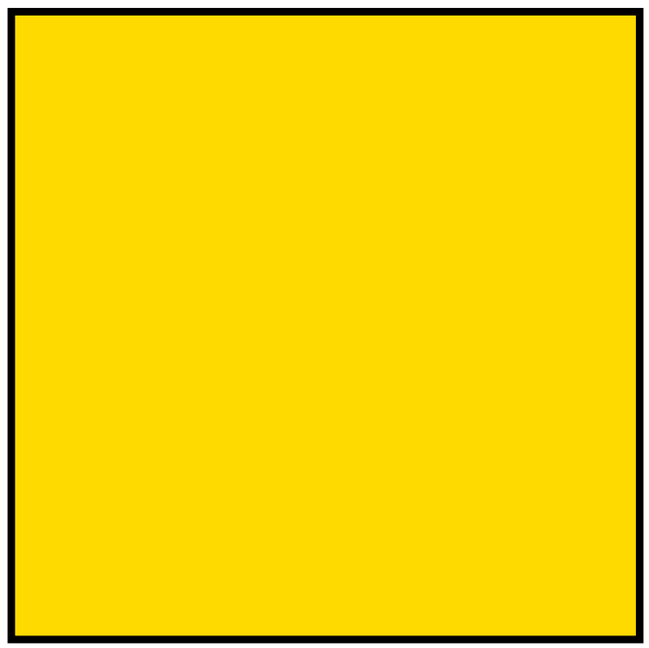
```
@Preview
@Composable
fun ADarkLightSquarePreview() {
    Column() {
        Row {
            WhiteSquare(modifier =
                Modifier.preferredSize(100.dp))
        }
        Row {
            BlackSquare(modifier =
                Modifier.preferredSize(100.dp))
            }
        }
    }
}
```



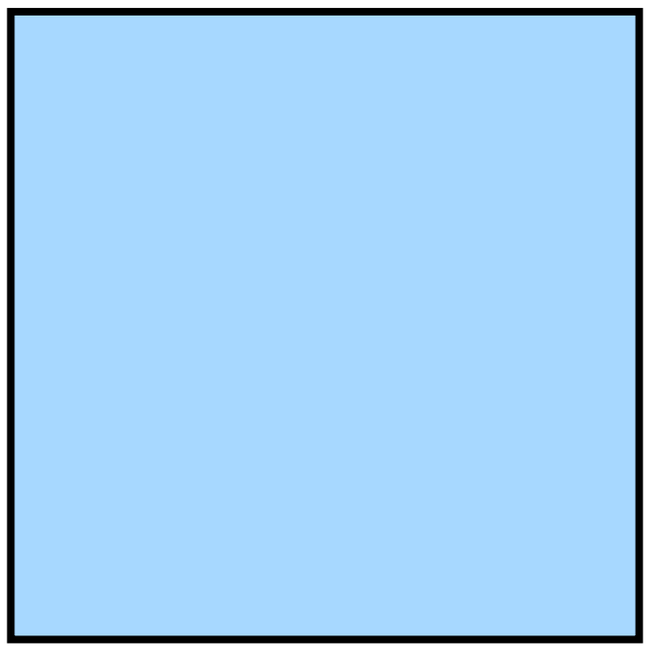
Cursors


9:00
LTE
1:14
1 S
2
3 I
4
5 K
T
6
A
C
7 U
T
O
H
8 R
E
L
Y
Nevada neighbor
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M

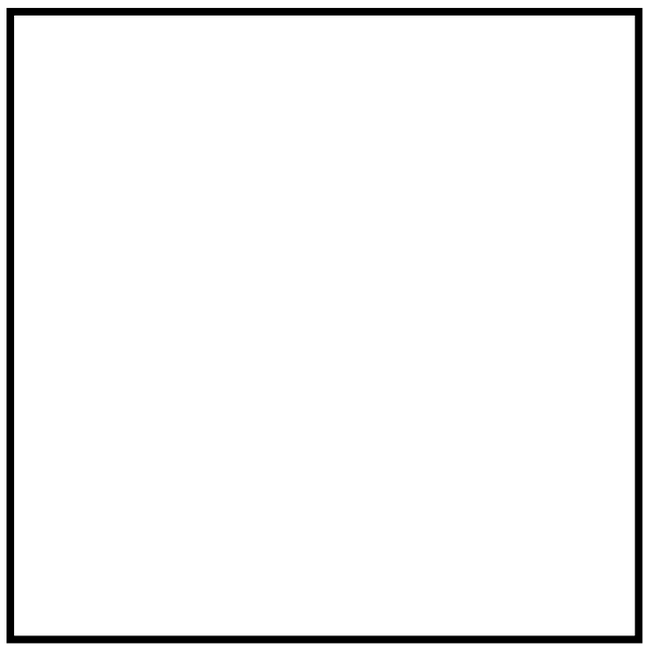
Cursors



Cursor



Active Clue



Unselected

Cursors

```
enum class SelectionMode {
    NONE, // plain old white square
    CURSOR, // the cursor is on this square
    ACTIVE_CLUE // this is the clue we're solving
}
```

```
enum class CellBackground constructor(val color: Color) {
    NONE(Color.White),
    CURSOR(Color(red = 0xFF, green = 0xDA, blue = 0x00)),
    ACTIVE_CLUE(Color(red = 0xA7, green = 0xD8, blue = 0xFF)),
    BLACK_SQUARE(Color.Black)
}
```

Cursors

```
object CellColorHelper {
    fun colorForSelectionMode(mode: SelectionMode) = when(mode) {
        SelectionMode.NONE -> CellBackground.NONE.color
        SelectionMode.CURSOR -> CellBackground.CURSOR.color
        SelectionMode.ACTIVE_CLUE -> CellBackground.ACTIVE_CLUE.color
    }
```

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode) {
    ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = modifier)
}
```

Cursors

```
object CellColorHelper {
    fun colorForSelectionMode(mode: SelectionMode) = when(mode) {
        SelectionMode.NONE -> CellBackground.NONE.color
        SelectionMode.CURSOR -> CellBackground.CURSOR.color
        SelectionMode.ACTIVE_CLUE -> CellBackground.ACTIVE_CLUE.color
    }
```

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode) {
    ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = modifier)
}
```

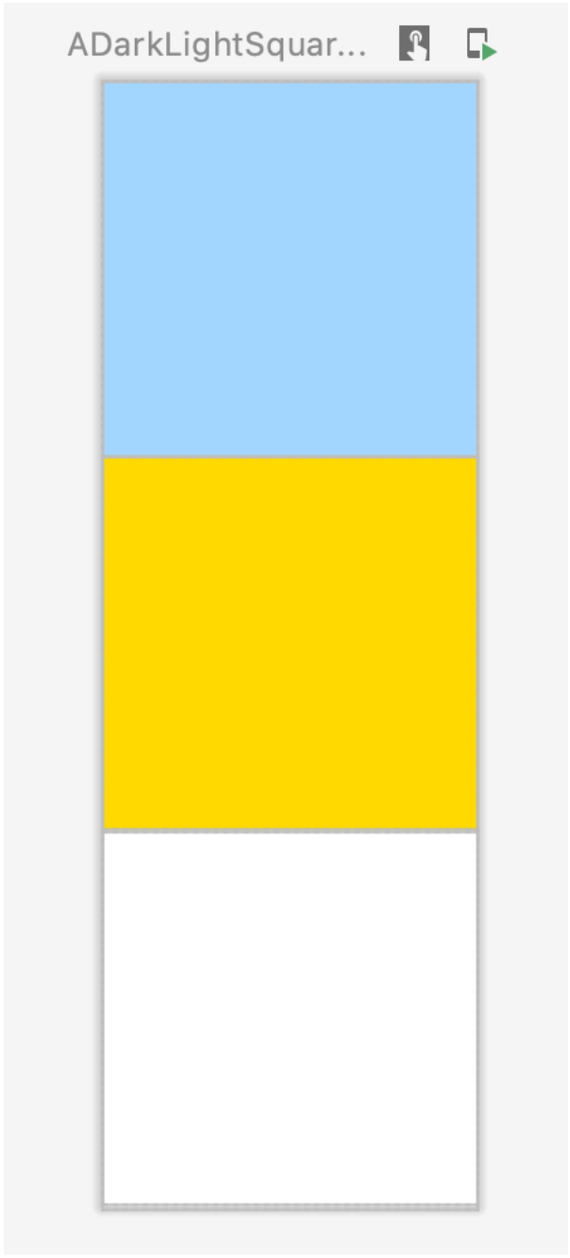
Cursors

```
object CellColorHelper {
    fun colorForSelectionMode(mode: SelectionMode) = when(mode) {
        SelectionMode.NONE -> CellBackground.NONE.color
        SelectionMode.CURSOR -> CellBackground.CURSOR.color
        SelectionMode.ACTIVE_CLUE -> CellBackground.ACTIVE_CLUE.color
    }
```

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode) {
    ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = modifier)
}
```

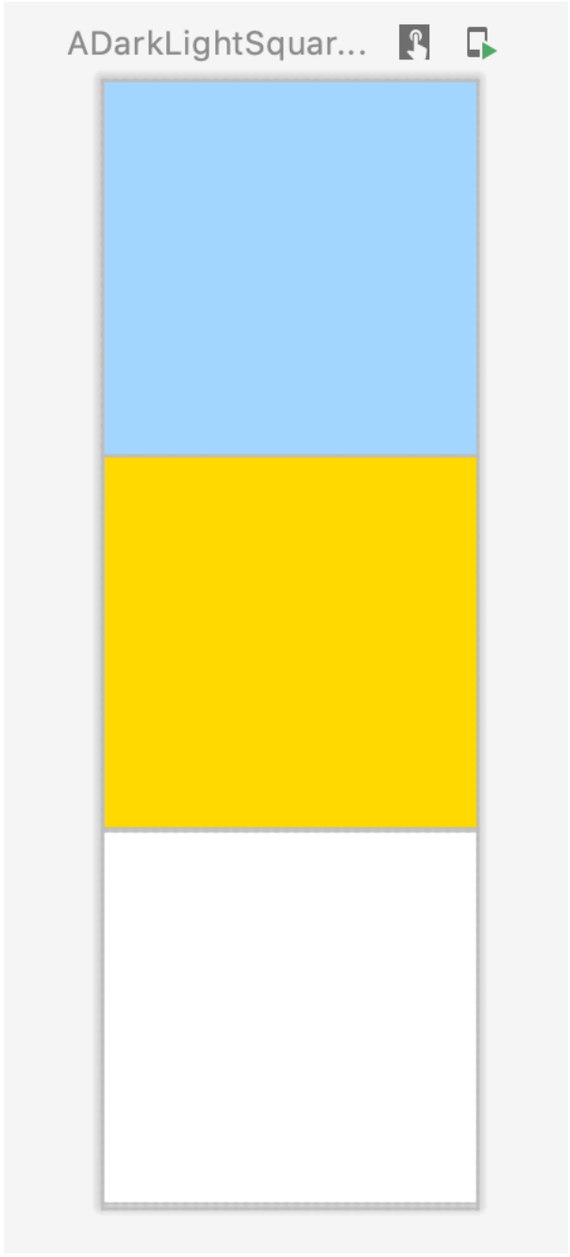
Cursors

```
@Preview
@Composable
fun ADarkLightSquarePreview() {
    Column() {
        Row {
            WhiteSquare(
                    selectionMode = ACTIVE_CLUE,
                    modifier = sizedTo100)
        }
        Row {
            WhiteSquare(
                    selectionMode = CURSOR,
                    modifier = sizedTo100)
        }
        Row {
            WhiteSquare(
                    selectionMode = NONE,
                    modifier = sizedTo100)
        }
    }
}
```



Cursors

```
@Preview
@Composable
fun ADarkLightSquarePreview() {
    Column() {
        Row {
            WhiteSquare(
                    selectionMode = ACTIVE_CLUE,
                    modifier = sizedTo100)
        }
        Row {
            WhiteSquare(
                    selectionMode = CURSOR,
                    modifier = sizedTo100)
        }
        Row {
            WhiteSquare(
                    selectionMode = NONE,
                    modifier = sizedTo100)
        }
    }
}
```



Text

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode,
                text: String? = null) {


    Stack(modifier = modifier) {
        ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = Modifier
                    .fillMaxSize()
                    .gravity(Alignment.Center))
    }
}
```

Text

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode,
                text: String? = null) {

    Stack(modifier = modifier) {
        ColoredSquareBackground(
                color = colorForSelectionMode(selectionMode),
                modifier = Modifier
                        .fillMaxSize()
                        .gravity(Alignment.Center))
    }
}
```

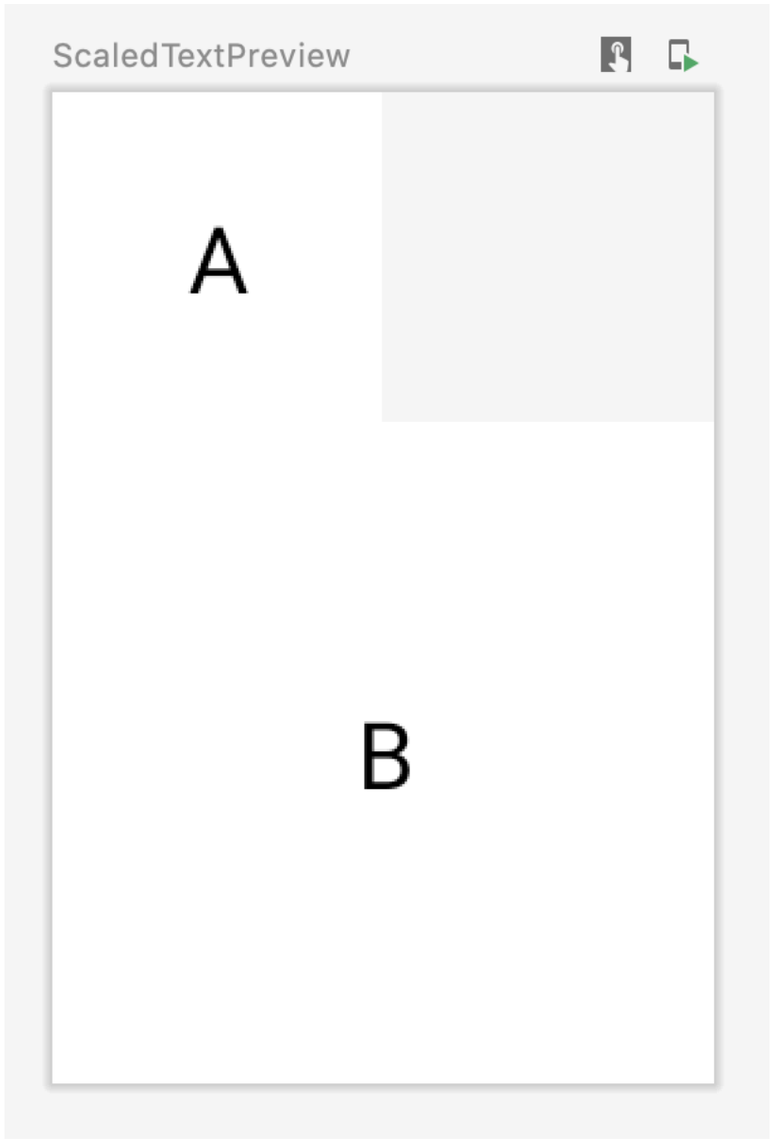
Text

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode,
                text: String? = null) {


    Stack(modifier = modifier) {
        ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = Modifier
                    .fillMaxSize()
                    .gravity(Alignment.Center))
    }
}
```

Text

```
Stack(modifier = modifier) {
    ColoredSquareBackground(
            color = colorForSelectionMode(selectionMode),
            modifier = Modifier
                    .fillMaxSize()
                    .gravity(Alignment.Center))

    text?.let {userAnswer ->
        Text(text = userAnswer,
                modifier = Modifier.gravity(Alignment.Center))
    }
}
```

Text



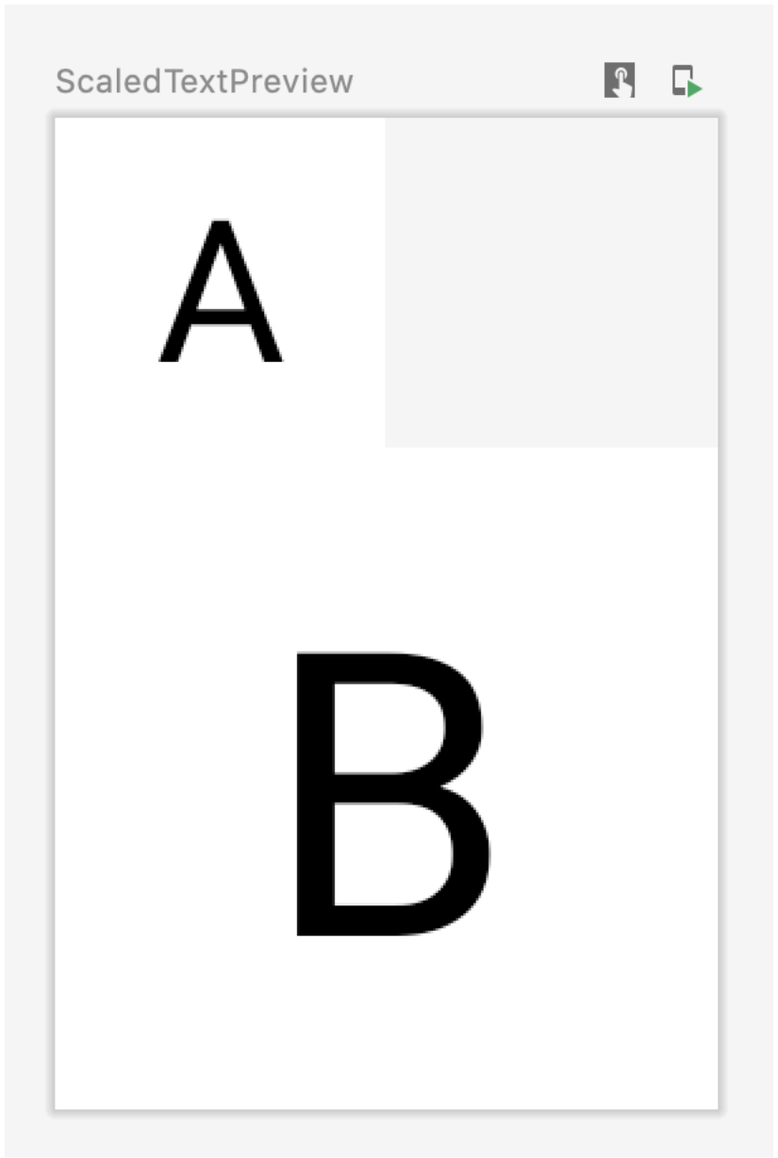
Scaled Text

```
WithConstraints {
    val textSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .6f).toSp()
    }

    Stack {
        ColoredSquareBackground(
                color = colorForSelectionMode(selectionMode),
                modifier = Modifier
                        .fillMaxSize()
                        .gravity(Alignment.Center))

        text?.let { userAnswer ->
            Text(text = userAnswer,
                    style = TextStyle(fontSize = textSize),
                    modifier = Modifier.gravity(Alignment.Center))
        }
    }
}
```

Scaled Text

```
WithConstraints {
    val textSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .6f).toSp()
    }

    Stack {
        ColoredSquareBackground(
                color = colorForSelectionMode(selectionMode),
                modifier = Modifier
                        .fillMaxSize()
                        .gravity(Alignment.Center))

        text?.let { userAnswer ->
            Text(text = userAnswer,
                    style = TextStyle(fontSize = textSize),
                    modifier = Modifier.gravity(Alignment.Center))
        }
    }
}
```

Scaled Text



Home

PUBLIC

Stack Overflow

Tags

Users

Job Board

TEAMS What's this?

Q&A for Work

261 results

Relevance Newest More

819 votes

A: Converting pixels to dp

/** * This method converts **dp** unit to equivalent pixels, depending on device density. * * @param **dp** A value in **dp** (density independent pixels) unit. Which we need to **convert** into pixels … * @param context Context to get resources and device specific display metrics * @return A float value to represent **px** equivalent to **dp** depending on device density */ public static float convertDpToPixel …

answered Mar 5 '12 by Muhammad Nabeel Arif

48 votes

4 answers

Q: How to convert DP, PX, SP among each other, especially DP and SP?

I have known the difference among **DP**, SP and **PX**. And after searching this topic, I found nothing satisfying me completely. Maybe this post is a duplicate, but I still want to know what is the formula … of converting from **DP** to **PX**, and **DP** to SP, from SP to **PX**, from **PX** to SP, from SP to **DP**, from **DP** to SP? I have known some codes to do this, but they are imperfect. …

android pixel density-independent-pixel

asked Apr 16 '15 by SilentKnight

22 votes

1 answer

Q: Android Convert Px to Dp (Video Aspect Ratio) [duplicate]

Possible Duplicate: converting pixels to **dp** in android I'm trying to **convert** pixels to **dp**. What is the formula? Lets **convert** 640 and 480 into **dp**. The docs say this The conversion … of **dp** units to screen pixels is simple: **px** = **dp** * (dpi / 160) But I don't think that is what I need (and I don't know how to use this). I guess I just need the forumla. I have the code ready …

android screen-resolution aspect-ratio

asked Jul 11 '11 by spentak

325 votes

A: How to calculate dp from pixels in android programmatically

All the answers here show a **dp**->**px** conversion rather than **px**->**dp**, which is what the OP asked for. Note that TypedValue.applyDimension cannot be used to **convert** **px**->**dp**, for that you must use the …) (**dp** * Resources.getSystem().getDisplayMetrics().density); } public static int pxToDp(int **px**) { return (int) (**px** / Resources.getSystem().getDisplayMetrics().density); } …

answered Nov 13 '13 by Vicky Chijwani

182 votes

Q: How to determine the screen width in terms of dp or dip at runtime in Android?

width in pixels (**px**). To **convert** this to **dp**, I coded: int **dp** =pixel/(int)getResources().getDisplayMetrics().density ;

Adding a Number

```
@Composable
fun WhiteSquare(modifier: Modifier = Modifier.fillMaxSize(),
                selectionMode: SelectionMode,
                text: String? = null,
                cellNumber: String? = null) {
```

Adding a Number

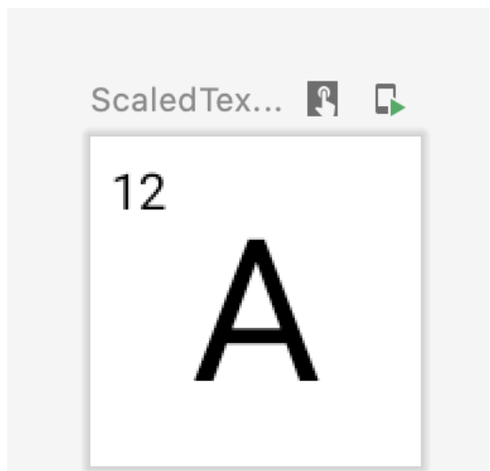
```
WithConstraints {
    val textSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .6f).toSp()
    }

    val numberSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .15f).toSp()
    }

Stack {

    // rest omitted for clarity
    cellNumber?.let {
        Text(text = it,
                style = TextStyle(fontSize = numberSize),
                modifier = Modifier.gravity(Alignment.TopStart))
    }
}
```

Adding a Number

```
WithConstraints {
    val textSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .6f).toSp()
    }

    val numberSize = with(DensityAmbient.current) {
        (maxWidth.toPx() * .15f).toSp()
    }

    Stack {

        // rest omitted for clarity
        cellNumber?.let {
            Text(text = it,
                 style = TextStyle(fontSize = numberSize),
                 modifier = Modifier.gravity(Alignment.TopStart))
        }
}
```

Adding a Number

```
WhiteSquare(
        text = "A",
selectionMode = NONE,
cellNumber = "12",
modifier = Modifier.preferredSize(50.dp))
```



BZZZT, wrong


9:00
LTE
1:14
1 S
2
3 I
4
5 K
T
6
A
C
7 U
T
O
H
8 R
E
L
Y
Nevada neighbor
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M

BZZZT, wrong

```
@Composable
fun StrikeOut() {
    val strokeWidth = with(DensityAmbient.current) {
        5.dp.toPx()
    }

    Canvas(modifier = Modifier.fillMaxSize()) {
        drawLine(Color.Red,
                 Offset(this.size.width, 0f),
                 Offset(0f, this.size.height),
                 strokeWidth)
    }
}
```

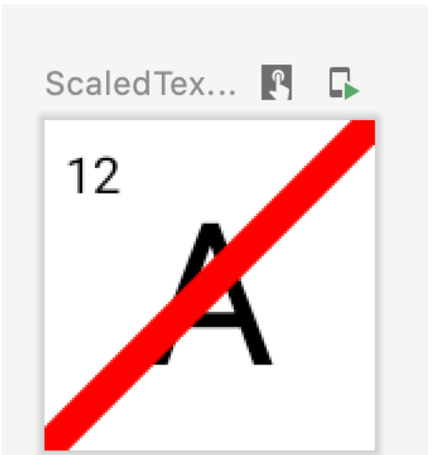
BZZZT, wrong

In our stack:

```
if (strikeOut) {
    StrikeOut()
}
```

BZZZT, wrong

```
WhiteSquare(
        text = "A",
        selectionMode = NONE,
        cellNumber = "12",
        strikeOut = true,
        modifier = Modifier.preferredSize(50.dp))
```



Adapting to a Model

```
enum class SquareType {
    BLACK,
    LETTER
}
```

```
class Square private constructor(
    val squareType: SquareType,
    val answer: String? = null,
    val cellNumber: String? = null,
    var userAnswer: String? = null,
    var checked: Boolean = false,
    var selectionMode: SelectionMode = SelectionMode.NONE) {

    companion object {
        fun forLetter(answer: String, cellNumber: String? = null): Square {
            return Square(
                squareType = SquareType.LETTER,
                answer = answer,
                cellNumber = cellNumber,
                selectionMode = SelectionMode.NONE
            )
        }

        fun forBlack() : Square {
            return Square(squareType = SquareType.BLACK)
        }
    }
}
```

Adapting to a Model

```
enum class SquareType {
    BLACK,
    LETTER
}

class Square private constructor(
    val squareType: SquareType,
    val answer: String,
    val cellNumber: String? = null,
    var userAnswer: String? = null,
    var checked: Boolean = false,
    var selectionMode: SelectionMode = SelectionMode.NONE) {

    companion object {
        fun forLetter(answer: String, cellNumber: String? = null): Square {
            return Square(
                squareType = SquareType.LETTER,
                answer = answer,
                cellNumber = cellNumber,
                selectionMode = SelectionMode.NONE
            )
        }

        fun forBlack() : Square {
            return Square(squareType = SquareType.BLACK)
        }
    }
}
```

Adapting to a Model

```
enum class SquareType {
    BLACK,
    LETTER
}

class Square private constructor(
    val squareType: SquareType,
    val answer: String,
    val cellNumber: String? = null,
    var userAnswer: String? = null,
    var checked: Boolean = false,
    var selectionMode: SelectionMode = SelectionMode.NONE) {

    companion object {
        fun forLetter(answer: String, cellNumber: String? = null): Square {
            return Square(
                squareType = SquareType.LETTER,
                answer = answer,
                cellNumber = cellNumber,
                selectionMode = SelectionMode.NONE
            )
        }

        fun forBlack() : Square {
            return Square(squareType = SquareType.BLACK)
        }
    }
}
```

Model-Driven

```
LetterSquare(
    dimension = 150.dp,
    square = Square.forLetter(answer = "A")
)
```

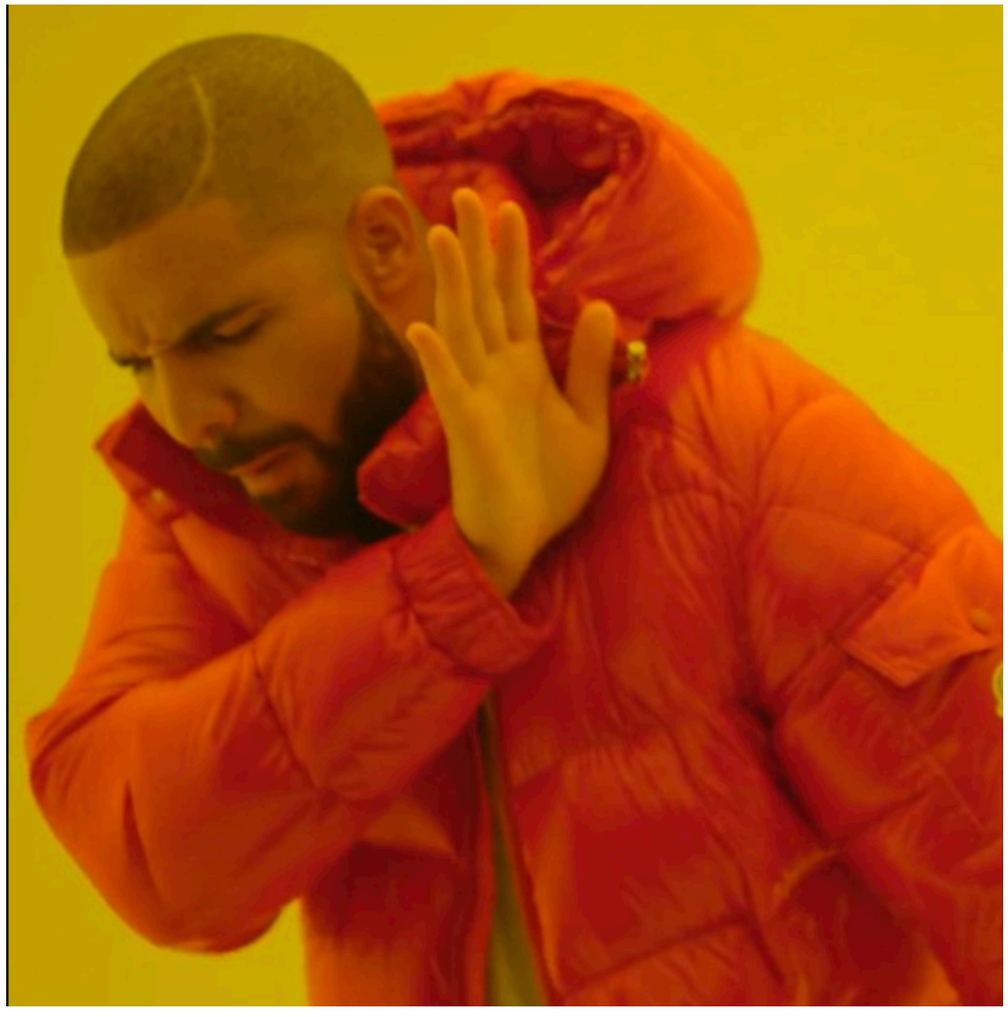
Build a Puzzle

```
class Board(val edgeCount: Int,
            val squares: List<Square>)


fun board() : Board {
    val squares = listOf(

        Square.forBlack(),
        Square.forLetter(answer = "S", cellNumber = "1"),
        Square.forLetter(answer = "P", cellNumber = "2"),
        Square.forLetter(answer = "I", cellNumber = "3"),
        Square.forLetter(answer = "T", cellNumber = "4"),

        // 20 more of these…
    )

     return Board(squares = squares,
         edgeCount = 5)

}
```

Render the Board

```
board.squares.chunked(board.edgeCount).forEach { row ->
    Row {
        row.forEach { square ->
            when (square.squareType) {
                SquareType.LETTER ->
                    LetterSquare(
                        modifier = Modifier.preferredSize(squareWidth),
                        square = square)
                SquareType.BLACK -> BlackSquareBackground(
                        modifier = Modifier.preferredSize(squareWidth),
                )
            }
        }
    }
}
```

Render the Board



Render the Board

BoardHost

	1	2	3	4
	5			
6				
7				
8				

Clues

```
class Clue(val direction: Direction,
           val cells: List<Int>,
           val clueText: String)

val clues = listOf(
    Clue(direction = Direction.ACROSS,
        cells = listOf(1, 2, 3, 4),
        clueText = "Card Game That Rewards Speed"),

    Clue(direction = Direction.ACROSS,
        cells = listOf(6, 7, 8, 9),
        clueText = "Low-carb, high-fat diet, familiarly"),

    Clue(direction = Direction.ACROSS,
        cells = listOf(10, 11, 12, 13, 14),
        clueText = "Hi-Falutin'"),

    Clue(direction = Direction.ACROSS,
        cells = listOf(15, 16, 17, 18),
        clueText = "Nevada Neighbor”),

    …
)
```

Cursors and Clues

```
class BoardViewModel: ViewModel() {

    fun enterLetter(letter: String) {
        //
    }

    fun selectSquare(square: Square) {
        //
    }

}
```

Wiring Up The Click Event

```
@Composable
fun GameBoard(squares: List<Square>,
              squareOnClick: (Square) -> Unit) {

LetterSquare(
    modifier = Modifier.preferredSize(squareWidth),
    square = square,
    onClick = {
        squareOnClick(it)
    }
)

@Composable
fun LetterSquare(modifier: Modifier = Modifier.fillMaxSize(),
                 square: Square,
                 onClick: (Square) -> Unit) {

    Box(modifier = modifier.clickable(onClick = {onClick(square)})) {
```

Wiring Up The Click Event

```
@Composable
fun GameBoard(squares: List<Square>,
              squareOnClick: (Square) -> Unit) {

LetterSquare(
    modifier = Modifier.preferredSize(squareWidth),
    square = square,
    onClick = {
        squareOnClick(it)
    }
)

@Composable
fun LetterSquare(modifier: Modifier = Modifier.fillMaxSize(),
                 square: Square,
                 onClick: (Square) -> Unit) {

    Box(modifier = modifier.clickable(onClick = {onClick(square)})) {
```

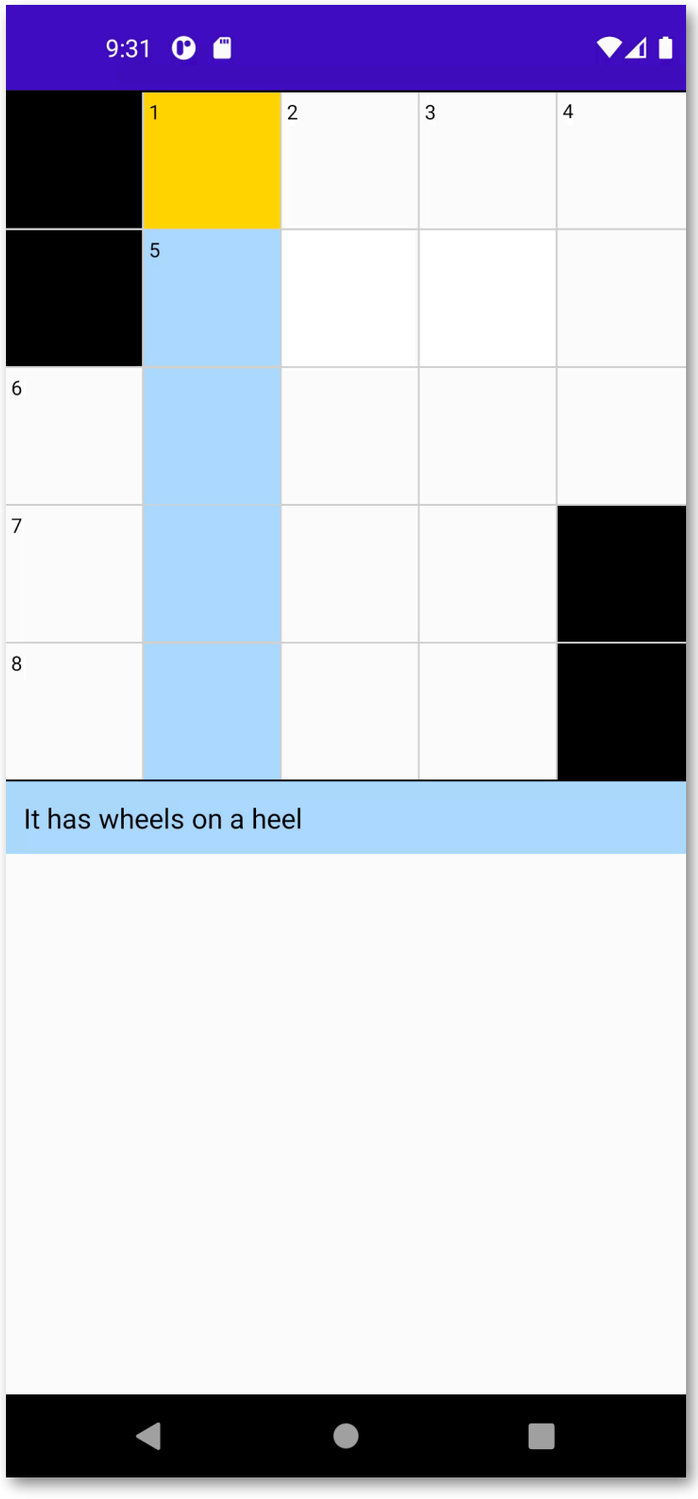
Wiring Up The Click Event

```
@Composable
fun GameBoard(squares: List<Square>,
              squareOnClick: (Square) -> Unit) {

LetterSquare(
    modifier = Modifier.preferredSize(squareWidth),
    square = square,
    onClick = {
        squareOnClick(it)
    }
)

@Composable
fun LetterSquare(modifier: Modifier = Modifier.fillMaxSize(),
                 square: Square,
                 onClick: (Square) -> Unit) {

    Box(modifier = modifier.clickable(onClick = {onClick(square)})) {
```

Wiring Up The Click Event

```
setContent {
    XWDComposeTheme {
        Surface(color = MaterialTheme.colors.background) {
            Column {
                viewModel.boardState?.let {boardState ->
                    GameBoard(boardState.squares,
                        viewModel::selectSquare)
                    ClueBar(clue = boardState.selectedClue)
                }
            }
        }
    }
}
```

Wiring Up The Click Event



Filling the Board

```
@Composable
fun CustomKeyboard(onKeyClicked: (String) -> Unit) {
    val context = ContextAmbient.current
    val keyboardView = KeyboardView(context, null)
    keyboardView.keyboard = Keyboard(context, R.xml.keyboard)

    AndroidView({ keyboardView }) { keyboard ->
        keyboard.setOnKeyboardActionListener(object :
KeyboardView.OnKeyboardActionListener {
            override fun onKey(primaryCode: Int, keyCodes: IntArray?) {
                onKeyClicked(primaryCode.toChar().toString())
            }

        })
    }
}
```

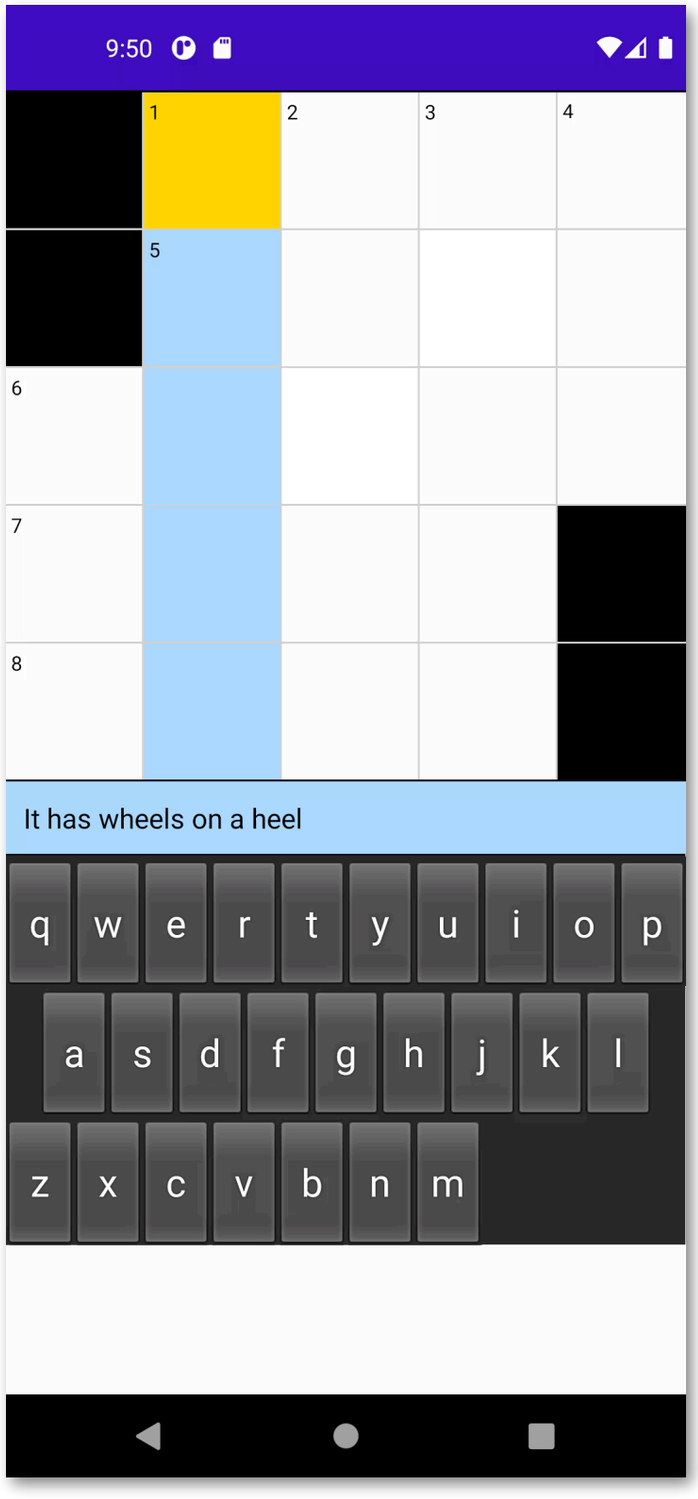
Filling the Board

```
@Composable
fun CustomKeyboard(onKeyClicked: (String) -> Unit) {
    val context = ContextAmbient.current
    val keyboardView = KeyboardView(context, null)
    keyboardView.keyboard = Keyboard(context, R.xml.keyboard)

    AndroidView({ keyboardView }) { keyboard ->
        keyboard.setOnKeyboardActionListener(object :
KeyboardView.OnKeyboardActionListener {
            override fun onKey(primaryCode: Int, keyCodes: IntArray?) {
                onKeyClicked(primaryCode.toChar().toString())
            }

        })
    }
}
```

Filling the Board

```
Column {
    viewModel.boardState?.let {boardState ->
        GameBoard(boardState.squares,
            viewModel::selectSquare)
        ClueBar(clue = boardState.selectedClue)
    }
    Box(modifier = Modifier.fillMaxWidth()) {
        CustomKeyboard {
            viewModel.enterLetter(it)
        }
    }
}
```

Filling the Board



Checking Your Work

```
square.userAnswer?.let { userAnswer ->
    if (square.checked && userAnswer != square.answer) {
        StrikeOut()
    }
}
```

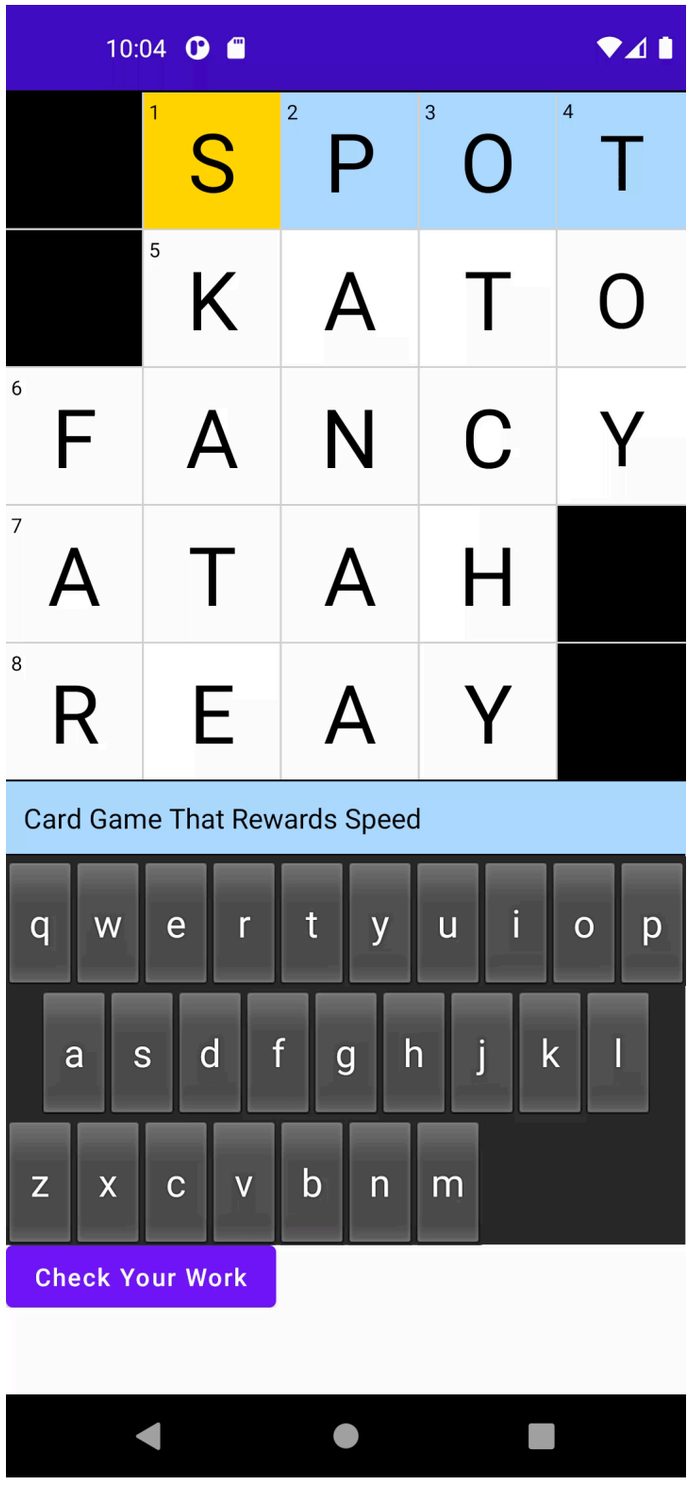
Checking Your Work

```
fun checkBoard() {
    squares = squares.map {
        it.checked = true
        it
    }
    emitBoardState()
}
```

Checking Your Work

```
Column {
    viewModel.boardState?.let {boardState ->
        GameBoard(boardState.squares,
            viewModel::selectSquare)
        ClueBar(clue = boardState.selectedClue)
    }
    Box(modifier = Modifier.fillMaxWidth()) {
        CustomKeyboard {
            viewModel.enterLetter(it)
        }
    }
    Button(onClick = {viewModel.checkBoard()}){
        Text("Check Your Work")
    }
}
```

Checking Your Work



We Did It! 

What'd We Build?

- We built a Jetpack Compose UI around a data model that's highly similar to the existing data model in the shipping Crossword app
- Decomposed the Crossword UI into its most basic components and rebuilt from the "leaves" inward
- Had some fun!

Some Parting Gifts

https://tinyurl.com/xwd-compose

THANK YOU!

P.S. We're hiring
http://nytco.com/careers