

Карты, схемы и Compose

Александр Нозик

Что будет?

- Задача
- Карты это просто?
 - Геодезические координаты и геодезические прямые.
 - Равноугольные и равноплощадные проекции.
 - Дивный мир EPSG.
- Compose во всей красе
 - Десктоп (все еще) мертв?
 - Compose для тех, кто не в теме.
 - Compose != React. Работа со State-структурами.
 - Иерархический State.
- В канве
 - Как работает канва в Compose.
 - А теперь иконки!
 - Обработчики событий мыши.
- Собираем все вместе.

Обо мне

- Директор Центра Научного Программирования.
 - К. ф.–м. н. по физике частиц.
 - Преподаватель МФТИ.
 - (Со-)руководитель российского KUG.
-
- <https://sciprogramm.center/people/Nozik>
 - <https://twitter.com/noraltavir>
 - <https://t.me/noraltavir>



Немного о картах

ДОКЛАД

Урок географии для Java-разработчиков. География и Java в одном чайнике



Павел Кислов

Домклик

RU



Spring and Database

Joker 2022

Геодезические координаты

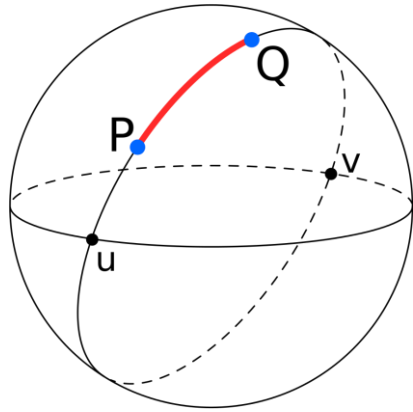
- Широта (угол)
- Долгота (угол)
- Иногда высота (расстояние)



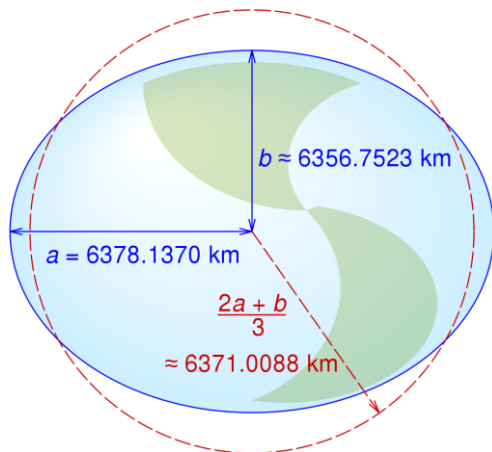
```
@Serializable
public class GeodeticMapCoordinates(
    public val latitude: Angle,
    public val longitude: Angle,
    public val elevation: Distance? = null,
)

public typealias Gmc = GeodeticMapCoordinates
```

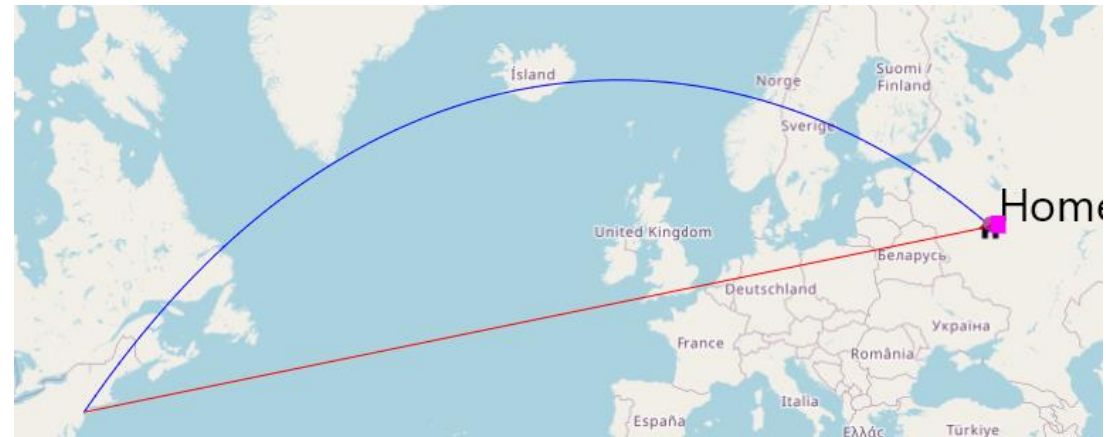
Геодезические отрезки



https://en.wikipedia.org/wiki/Great-circle_distance



- Геодезическая – прямой отрезок минимальной длины между двумя точками на поверхности.
- А что такое прямой?



Насколько земля не плоская?

$$\frac{1}{f} = \frac{a}{a - b}$$

Reference ellipsoid name	Equatorial radius (m)	Polar radius (m)	Inverse flattening	Where used
Maupertuis (1738)	6,397,300	6,363,806.283	191	France
Plessis (1817)	6,376,523.0	6,355,862.9333	308.64	France
Everest (1830)	6,377,299.365	6,356,098.359	300.80172554	India
...
Australian National (1966)	6,378,160	6,356,774.719	298.25	Australia
New International (1967)	6,378,157.5	6,356,772.2	298.24961539	
GRS-67 (1967)	6,378,160	6,356,774.516	298.247167427	
South American (1969)	6,378,160	6,356,774.719	298.25	South America
WGS-72 (1972)	6,378,135	6,356,750.52	298.26	USA/DoD
GRS-80 (1979)	6,378,137	6,356,752.3141	298.257222101	Global ITRS
WGS-84 (1984)	6,378,137	6,356,752.3142	298.257223563	Global GPS
IERS (1989)	6,378,136	6,356,751.302	298.257	
IERS (2003)	6,378,136.6	6,356,751.9	298.25642	

https://en.wikipedia.org/wiki/Earth_ellipsoid

Расстояния в геодезии

Расстояние по большому кругу

$$\Delta\sigma = \arccos(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda)).$$

```
/**
 * https://en.wikipedia.org/wiki/Great-circle_distance
 */
fun greatCircleAngleBetween(
    r1: Gmc,
    r2: Gmc,
): Radians = acos(
    sin(r1.latitude) * sin(r2.latitude) +
    cos(r1.latitude) * cos(r2.latitude) *
    cos(r1.longitude - r2.longitude)
).radians
```

```
val tanU1 = (1.0 - f) * tanphi1
val U1: Double = atan(tanU1)
val sinU1: Double = sin(U1)
val cosU1: Double = cos(U1)
val tanphi2: Double = tan(phi2)
val tanU2 = (1.0 - f) * tanphi2
val U2: Double = atan(tanU2)
val sinU2: Double = sin(U2)
val cosU2: Double = cos(U2)
val sinU1sinU2 = sinU1 * sinU2
val cosU1sinU2 = cosU1 * sinU2
val sinU1cosU2 = sinU1 * cosU2
val cosU1cosU2 = cosU1 * cosU2

// eq. 13
val lambda: Angle = omega

// intermediates we'll need to compute 's'
var A = 0.0

var sigma = 0.0
var deltasigma = 0.0
var lambda0: Angle
var converged = false
for (i in 0..19) {
    lambda = lambda0
    val sinlambda: Double = sin(lambda)
    val coslambda: Double = cos(lambda)

    // eq. 14
    val sin2sigma =
        cosU2 * sinlambda * cosU2 * sinlambda +
        (cosU1sinU2 - sinU1cosU2 * coslambda) * (cosU1sinU2 - sinU1cosU2 * coslambda)
    val sinsigma: Double = sqrt(sin2sigma)

    // eq. 15
    val cossigma = sinU1sinU2 + cosU1cosU2 * coslambda

    // eq. 16
    sigma = atan2(sinsigma, cossigma)

    // eq. 17 Careful! sin2sigma might be almost 0!
    val sinalpha = if (sin2sigma == 0.0) 0.0 else cosU1cosU2 * s:
    val alpha: Double = asin(sinalpha)
    val cosalpha: Double = cos(alpha)
    val cos2alpha = cosalpha * cosalpha

    // eq. 18 Careful! cos2alpha might be almost 0!
    val cos2sigmam = if (cos2alpha == 0.0) 0.0 else cossigma - 2
    val u2 = cos2alpha * a2b2b2
    val cos2sigmam2 = cos2sigmam * cos2sigmam

    // eq. 3
    A = 1.0 + u2 / 16384 * (4096 + u2 * (-768 + u2 * (320 - 175 * u2)))

    // eq. 4
    val B = u2 / 1024 * (256 + u2 * (-128 + u2 * (74 - 47 * u2)))

    // eq. 6
    deltasigma =
        B * sinsigma * (cos2sigmam + B / 4 * (cossigma * (-1 + 2 * cos2sigmam2) -
        B / 6 * cos2sigmam * (-3 + 4 * sin2sigma) * (-3 + 4 * cos2sigmam2)))

    // eq. 10
    val C = f / 16 * cos2alpha * (4 + f * (4 - 3 * cos2alpha))

    // eq. 11 (modified)
    lambda = omega + (
        (1 - C) * f * sinalpha *
        (sigma + C * sinsigma * (cos2sigmam + C * cossigma * (-1 + 2 * cos2sigmam2)))
    ).radians

    // see how much improvement we got
    val change: Double = abs(lambda - lambda0) / lambda
    if (i > 1 && change < precision) {
        converged = true
        break
    }
}
```



Формула Винсенти

https://en.wikipedia.org/wiki/Vincenty%27s_formulae



Mike Gavaghan

mgavaghan

<https://github.com/mgavaghan/geodesy>

Thanks Mike!

Start by calculating the following:

$$U_1 = \arctan[(1 - f) \tan \phi_1]$$

$$\sigma_1 = \arctan2(\tan U_1, \cos \alpha_1)$$

$$\sin \alpha = \cos U_1 \sin \alpha_1$$

$$u^2 = \cos^2 \alpha \left(\frac{a^2 - b^2}{b^2} \right) = (1 - \sin^2 \alpha) \left(\frac{a^2 - b^2}{b^2} \right)$$

$$A = 1 + \frac{u^2}{16384} (4096 + u^2 [-768 + u^2 (320 - 175u^2)])$$

$$B = \frac{u^2}{1024} (256 + u^2 [-128 + u^2 (74 - 47u^2)])$$

Then, using an initial value $\sigma = \frac{s}{bA}$, iterate the following equations until there is no significant change in σ :

$$2\sigma_m = 2\sigma_1 + \sigma$$

$$\Delta\sigma = B \sin \sigma \left\{ \cos(2\sigma_m) + \frac{1}{4} B \left(\cos \sigma [-1 + 2 \cos^2(2\sigma_m)] - \frac{B}{6} \cos[2\sigma_m] [-3 + 4 \sin^2 \sigma] [-3 + 4 \cos^2(2\sigma_m)] \right) \right\}$$

$$\sigma = \frac{s}{bA} + \Delta\sigma$$

Once σ is obtained to sufficient accuracy evaluate:

$$\phi_2 = \arctan2 \left(\sin U_1 \cos \sigma + \cos U_1 \sin \sigma \cos \alpha_1, (1 - f) \sqrt{\sin^2 \alpha + (\sin U_1 \sin \sigma - \cos U_1 \cos \sigma \cos \alpha_1)^2} \right)$$

$$\lambda = \arctan2(\sin \sigma \sin \alpha_1, \cos U_1 \cos \sigma - \sin U_1 \sin \sigma \cos \alpha_1)$$

$$C = \frac{f}{16} \cos^2 \alpha [4 + f (4 - 3 \cos^2 \alpha)]$$

$$L = \lambda - (1 - C) f \sin \alpha \{ \sigma + C \sin \sigma (\cos[2\sigma_m] + C \cos \sigma [-1 + 2 \cos^2(2\sigma_m)]) \}$$

$$L_2 = L + L_1$$

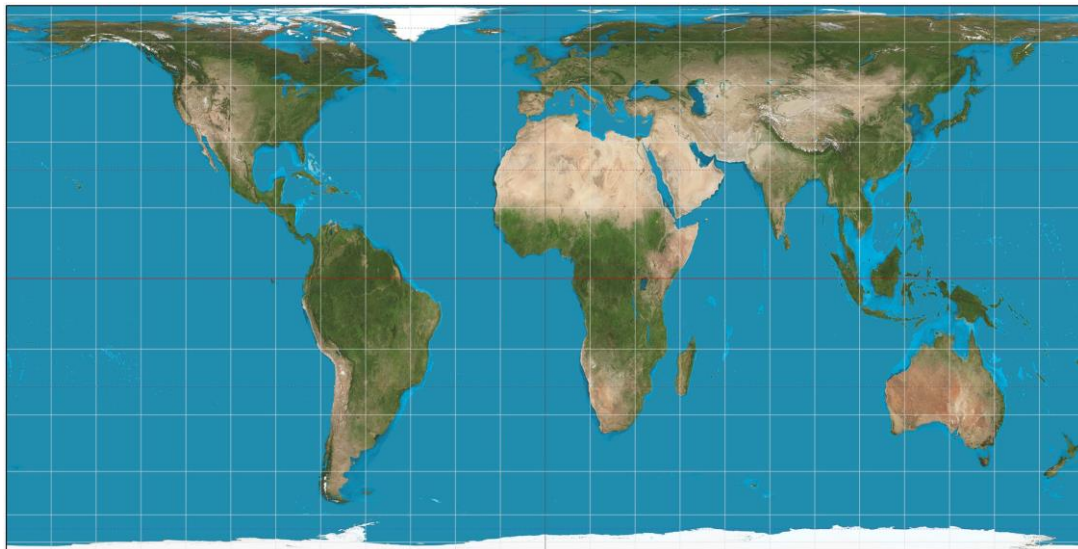
$$\alpha_2 = \arctan2(\sin \alpha, -\sin U_1 \sin \sigma + \cos U_1 \cos \sigma \cos \alpha_1)$$

Реализация на Котлин:

<https://github.com/SciProgCentre/maps-kt/blob/dev/maps-kt-core/src/commonMain/kotlin/center/sciprogram/maps/coordinates/GmcCurve.kt>

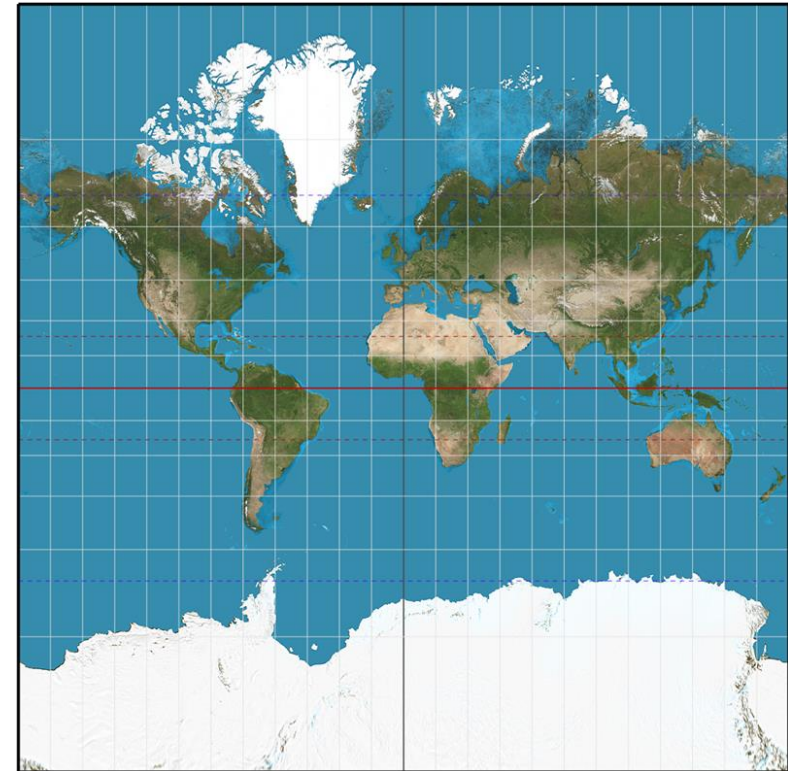
Проекции

Равноплощадная



https://en.wikipedia.org/wiki/Hobo%E2%80%93Dyer_projection

Равноугольная



https://en.wikipedia.org/wiki/Mercator_projection

Дивный мир EPSG



Это только проекции, специфичные для России

- [WGS 84 / EPSG Russia Polar Stereographic](#)
EPSG:5940
Area of use: Northern hemisphere - north of 60°N onshore and offshore, including Arctic.
- [WGS 84 / North Pole LAEA Russia](#)
EPSG:3576
Area of use: Northern hemisphere - north of 45°N, including Arctic.
- [Pulkovo 1942 / CS63 zone C0](#)
EPSG:3350 with transformation: 15865
Area of use: Armenia; Azerbaijan; Belarus; Estonia (accuracy: 4.5)
- [WGS 84 / EPSG Arctic zone 3-13](#)
EPSG:6077
Area of use: Arctic (Russia onshore and offshore)
- [WGS 84 / EPSG Arctic zone 3-15](#)
EPSG:6078
Area of use: Arctic (Russia onshore and offshore)
- [WGS 84 / EPSG Arctic zone 3-17](#)
EPSG:6079
Area of use: Arctic (Russia onshore and offshore)
- [WGS 84 / EPSG Arctic zone 3-19](#)
EPSG:6080
Area of use: Arctic (Russia onshore and offshore)
- [EVRF2019 height](#)
EPSG:9389
Area of use: Europe
- [EVRF2019 mean-tide height](#)
EPSG:9390
Area of use: Europe

Дивный мир EPSG

Вы думали, что у вас плохо со стандартизацией?



Export

OGC WKT

OGC WKT 2

ESRI WKT

PROJ.4

Proj4js

JSON

GeoServer

MapServer

Mapnik

PostGIS

Definition: OGC Well Known Text

[Open](#)

```
PROJCS["WGS 84 / EPSG Russia Polar Stereographic",  
  GEOGCS["WGS 84",  
    DATUM["WGS_1984",  
      SPHEROID["WGS 84",6378137,298.257223563,  
        AUTHORITY["EPSG","7030"]],  
      AUTHORITY["EPSG","6326"]],  
    PRIMEM["Greenwich",0,  
      AUTHORITY["EPSG","8901"]],  
    UNIT["degree",0.0174532925199433,  
      AUTHORITY["EPSG","9122"]],  
      AUTHORITY["EPSG","4326"]],  
  PROJECTION["Polar_Stereographic"],  
  PARAMETER["latitude_of_origin",90],  
  PARAMETER["central_meridian",105],  
  PARAMETER["scale_factor",0.994],  
  PARAMETER["false_easting",2000000],  
  PARAMETER["false_northing",2000000],  
  UNIT["metre",1,  
    AUTHORITY["EPSG","9001"]],  
  AUTHORITY["EPSG","5940"]]
```

MapProjection

```
/**
 * @param T the type of projection coordinates
 */
public interface MapProjection<T : Any> {
    public fun toGeodetic(pc: T): GeodeticMapCoordinates
    public fun toProjection(gmc: GeodeticMapCoordinates): T

    public companion object {
        public val epsg3857: MercatorProjection = MercatorProjection()
    }
}
```

Из проекции в геодезические координаты

Обратно

А это Меркатор



Web Mercator

```
public data class WebMercatorCoordinates(val zoom: Int, val x: Float, val y: Float)
```



<https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/>

- Положение кодируется тремя координатами:
 - x – координата по долготе в пикселях (0 сверху)
 - y – координата по широте в пикселях (0 слева)
 - $zoom$ – логарифмическая координата «удаления».
- Размер каждого «тайла» 256x256.

Web Mercator

```
public fun scaleFactor(zoom: Float): Float = (256.0 / 2 / PI * 2f.pow(zoom)).toFloat()

public fun toGeodetic(mercator: WebMercatorCoordinates): GeodeticMapCoordinates {
    val scaleFactor = scaleFactor(mercator.zoom.toFloat())
    val longitude = mercator.x / scaleFactor - PI
    val latitude = (atan(exp(PI - mercator.y / scaleFactor)) - PI / 4) * 2
    return GeodeticMapCoordinates.ofRadians(latitude, longitude)
}

public fun toMercator(gmc: GeodeticMapCoordinates, zoom: Int): WebMercatorCoordinates? {
    if (abs(gmc.latitude) > MercatorProjection.MAXIMUM_LATITUDE) return null

    val scaleFactor = scaleFactor(zoom.toFloat())
    return WebMercatorCoordinates(
        zoom = zoom,
        x = scaleFactor * (gmc.longitude.radians + PI).toFloat(),
        y = scaleFactor * (PI - ln(tan(PI / 4 + gmc.latitude.radians / 2))).toFloat()
    )
}
```

Не для всех широт работает

Загрузка тайлов

```
public data class TileId(  
    val zoom: Int,  
    val i: Int,  
    val j: Int,  
)
```

<https://github.com/SciProgCentre/maps-kt/blob/dev/maps-kt-compose/src/commonMain/kotlin/center/sciprogram/maps/compose/MapTileProvider.kt>

```
public data class MapTile(  
    val id: TileId,  
    val image: Image,  
)
```

```
public interface MapTileProvider {  
    public fun CoroutineScope.loadTileAsync(tileId: TileId): Deferred<MapTile>
```

Асинхронная загрузка тайлов

```
    public val tileSize: Int get() = DEFAULT_TILE_SIZE
```

```
    public fun toIndex(d: Float): Int = floor(d / tileSize).toInt()
```

Конверсия координаты в индекс тайла

```
    public fun toCoordinate(i: Int): Float = (i * tileSize).toFloat()
```

Определение координаты тайла

```
    public companion object {  
        public const val DEFAULT_TILE_SIZE: Int = 256  
    }  
}
```

Загрузка тайлов OSM

- Загрузка тайла происходит по урлу с индексами

```
private fun TileId.osmUrl() = URL("$osmBaseUrl/${zoom}/${i}/${j}.png")
```

```
private fun TileId.cacheFilePath() = cacheDirectory.resolve("${zoom}/${i}/${j}.png")
```

1. Конструируем URL тайла.
2. Конструируем локальный путь в кэше к файлу.
3. Проверяем, есть ли локальный файл и читается ли он.
4. Если читается, используем.
5. Если не читается, удаляем.
6. Если нет, то скачиваем.

Compose it

<https://github.com/SciProgCentre/maps-kt>

Глоссарий

- Compose – компиляторный плагин, управляющий состоянием.
- Jetpack Compose – UI фреймворк, созданный компанией Google для Android.
- Compose Multiplatform – фреймворк, созданный JetBrains для разработки в Kotlin Multiplatform (Android, Desktop, HTML, Wasm, Ios).

Compose, но не тот (пример)

<https://github.com/JakeWharton/mosaic>

```
Tests: 10 total  
Time: 0s
```



Десктоп (все еще) мертв?



Desktop applications are dead!

- Разрабатывать десктопные приложения дорого – надо разрабатывать и тестировать под разные платформы.
- Не работает на мобилке.
- Не работает в вебе.
- Зачем, если есть electron?

Анатомия Compose для тех, кто не в теме

```
@Composable
fun OsmMapPanel(
    project: Project,
    mapDescriptor: MapDescriptor.Osm,
    selectedObject: AttributeContainer?,
    onSelect: (AttributeContainer?) -> Unit,
) = with(project) {

    var polygonConstruction by remember {
        mutableStateOf<List<Gmc>?>(null)
    }
    ...
}
```

Волшебная аннотация, которая «красит код»
и включает плагин компилятора Compose.

Не волшебная функция, делегат.

Волшебная функция, которая
исключает то, что происходит внутри
из рекомпозиции.

Создает объект типа MutableState. Об этом потом.

Анатомия Compose для тех, кто не в теме

```
@Composable
fun OsmMapPanel(
    project: Project,
    mapDescriptor: MapDescriptor.Osm,
    selectedObject: AttributeContainer?,
    onSelect: (AttributeContainer?) -> Unit,
) = with(project) {

    var polygonConstruction by remember {
        mutableStateOf<List<Gmc>?>(null)
    }
    ...
}
```

- Composable функция осуществляет отрисовку и перерисовку (рекомпозицию).
- Функция вызывается каждый раз, когда ее аргументы меняются по значению.
- Рекомпозиция также происходит когда меняется значение State внутри Composable функции.
- Для того, чтобы вызвать изменение в UI нужно изменить какое-то состояние.

Compose != React.

React

```
const [age, setAge] = useState(28);
```

- Любая функция может быть компонентом.
- Хуки можно описывать только в «преамбуле» функции.
- Состояние передается только через параметры (на самом деле не совсем).

Compose

```
var age by remember{mutableStateOf(28)}
```

- Только @Composable функция является компонентом.
- Состояния можно описывать где угодно внутри @Composable функции.
- Можно передать объект типа State.

Работа со State-структурами.

```
@Stable
interface State<out T> {
    val value: T
}
```

Когда компилятор видит State, он автоматически подписывается на его изменения.

```
@Stable
interface MutableState<T> : State<T> {
    override var value: T
    operator fun component1(): T
    operator fun component2(): (T) -> Unit
}
```

Когда происходит изменение (по значению), то срабатывают все триггеры, на которые есть подписка.

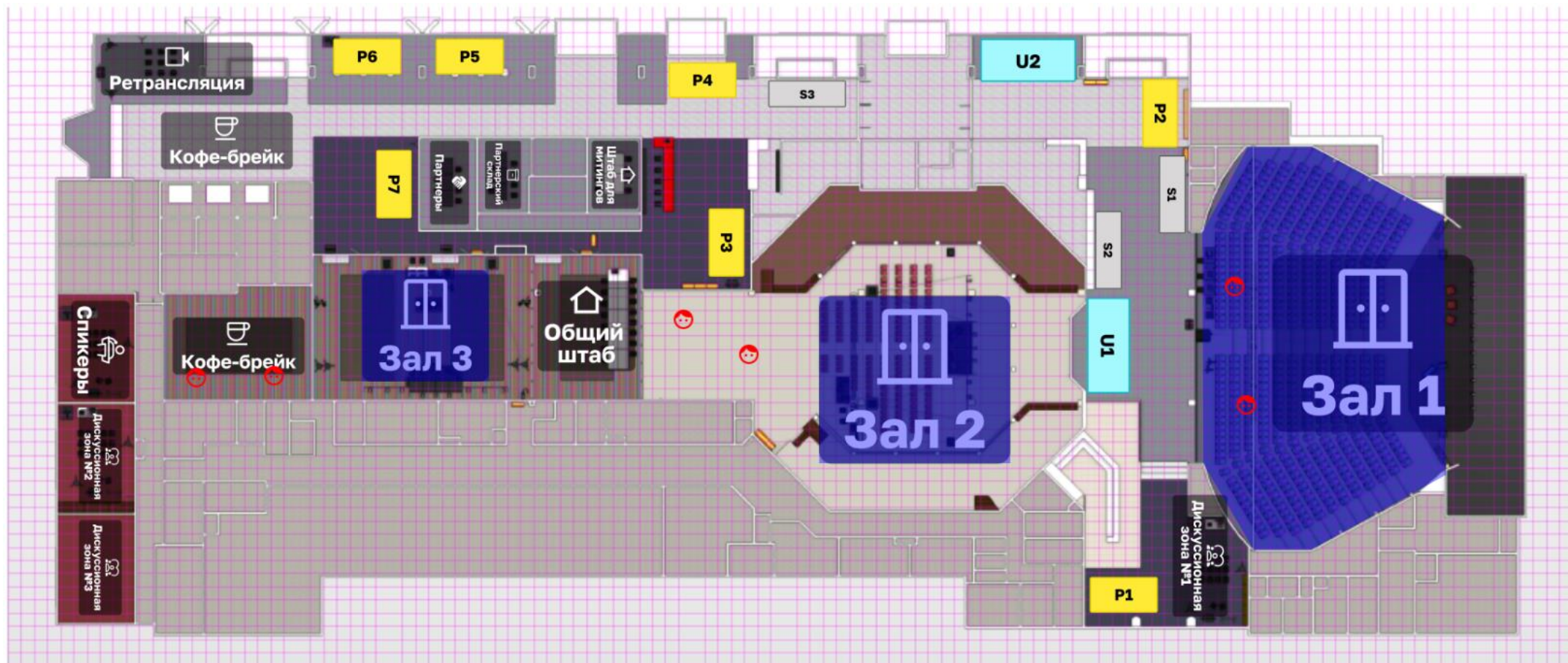
mutableStateMapOf()

Карта, которая триггерит рекомпозицию при изменении.

mutableStateListOf()

Список, который триггерит рекомпозицию при изменении.

Иерархический State: задача



Иерархический State: задача

```
background(1734f, 724f, id = "background") { painterResource("joker2023.png") }
```

```
group(id = "hall_2") {  
    rectanglePolygon(  
        left = 893, right = 1103,  
        bottom = 223, top = 406,  
    ).modifyAttributes {  
        ColorAttribute(Color.Blue)  
        AlphaAttribute(0.4f)  
    }.onClick {  
        println("hall_2")  
    }  
}
```

← Посмотрим, что происходит тут

```
group(id = "people") {  
    icon(XY(815.60535, 342.71313), Icons.Default.Face).color(Color.Red)  
    icon(XY(743.751, 381.09064), Icons.Default.Face).color(Color.Red)  
    icon(XY(1349.6648, 417.36014), Icons.Default.Face).color(Color.Red)  
    icon(XY(1362.4658, 287.21667), Icons.Default.Face).color(Color.Red)  
    icon(XY(208.24274, 317.08566), Icons.Default.Face).color(Color.Red)  
    icon(XY(293.5827, 319.21915), Icons.Default.Face).color(Color.Red)  
}
```

Иерархический State: реализация

```
public fun <T : Any, F : Feature<T>> FeatureRef<T, F>.modifyAttributes(  
    modify: AttributesBuilder.() -> Unit  
) : FeatureRef<T, F> {  
    parent.feature(  
        id,  
        resolve().withAttributes {  
            AttributesBuilder(this).apply(modify).build()  
        } as F  
    )  
    return this  
}
```

1. Резолвим существующую фичу.

2. Делаем копию с новым набором атрибутов.

3. Заменяем фичу по индексу новой фичей.

Ключевой вопрос: откуда
Compose знает, что надо
рекомпозироваться?

Иерархический State: реализация

Все волшебство тут!

```
public data class FeatureGroup<T : Any>(  
    override val space: CoordinateSpace<T>,  
    public val featureMap: SnapshotStateMap<String, Feature<T>> = mutableStateMapOf(),  
) : CoordinateSpace<T> by space, Feature<T> {...}
```

Пример динамического использования:

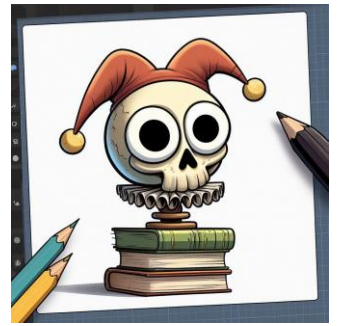
```
//remember feature ref  
val circleId = circle(centerCoordinates = pointTwo)  
  
scope.launch {  
    while (isActive) {  
        delay(200)  
        circleId.color(Color(Random.nextFloat(), Random.nextFloat(), Random.nextFloat()))  
    }  
}
```

Фичи иммутабельны, мы тут создаем копию

В канве



Канва или не канва, вот в чем вопрос?



Канва

- Все объекты отрисованы с на пиксельной канве.
- Менеджмент стейта и динамика на стороне приложения.
- Можно делать кастомный рендер (например SVG или JS библиотеку).

Не канва

- Все рисуется Composable функциями.
- Стейт менеджится самим Compose.
- Можно использовать только Compose.

Канва изнутри

Не композиabl

```
@Composable
fun Canvas(modifier: Modifier, onDraw: DrawScope.() -> Unit)
```

```
@DrawScopeMarker
@JvmDefaultWithCompatibility
interface DrawScope : Density {
```

```
/**
 * The current [DrawContext] that contains the dependencies
 * needed to create the drawing environment
 */
val drawContext: DrawContext
```

```
interface DrawContext {
```

```
    var size: Size
```

```
    /**
     * The target canvas to issue drawing commands
     */
    val canvas: Canvas
```

```
    val transform: DrawTransform
```

```
}
```

Канва изнутри

Canvas

- save(): Unit
- restore(): Unit
- saveLayer(Rect, Paint): Unit
- translate(Float, Float): Unit
- scale(Float, Float = ...): Unit
- rotate(Float): Unit
- skew(Float, Float): Unit
- skewRad(Float, Float): Unit
- concat(Matrix): Unit
- clipRect(Rect, ClipOp = ...): Unit
- clipRect(Float, Float, Float, Float, ClipOp = ...): Unit
- clipPath(Path, ClipOp = ...): Unit
- drawLine(Offset, Offset, Paint): Unit
- drawRect(Rect, Paint): Unit
- drawRect(Float, Float, Float, Float, Paint): Unit
- drawRoundRect(Float, Float, Float, Float, Float, Float, Paint): Unit
- drawOval(Rect, Paint): Unit
- drawOval(Float, Float, Float, Float, Paint): Unit
- drawCircle(Offset, Float, Paint): Unit
- drawArc(Rect, Float, Float, Boolean, Paint): Unit
- drawArc(Float, Float, Float, Float, Float, Float, Boolean, Paint): Unit
- drawArcRad(Rect, Float, Float, Boolean, Paint): Unit
- drawPath(Path, Paint): Unit
- drawImage(ImageBitmap, Offset, Paint): Unit

DrawTransform

- size: Size
- center: Offset
- inset(Float, Float, Float, Float): Unit
- clipRect(Float = ..., Float = ..., Float = ..., Float = ..., ClipOp = ...): Unit
- clipPath(Path, ClipOp = ...): Unit
- translate(Float = ..., Float = ...): Unit
- rotate(Float, Offset = ...): Unit
- scale(Float, Float, Offset = ...): Unit
- transform(Matrix): Unit

Single responsibility? Нет, не слышали.

Сделаем реализацию канвы для SVG

Canvas SVG backend

```
internal class SvgDrawContext(val graphics: SVGGraphics2D, override var size: Size) : DrawContext {  
    override val canvas: Canvas = SvgCanvas(graphics)  
  
    override val transform: DrawTransform = asDrawTransform()  
}
```

Исправляем косяки проектирования

```
internal class SvgCanvas(val graphics: SVGGraphics2D) : Canvas{
```

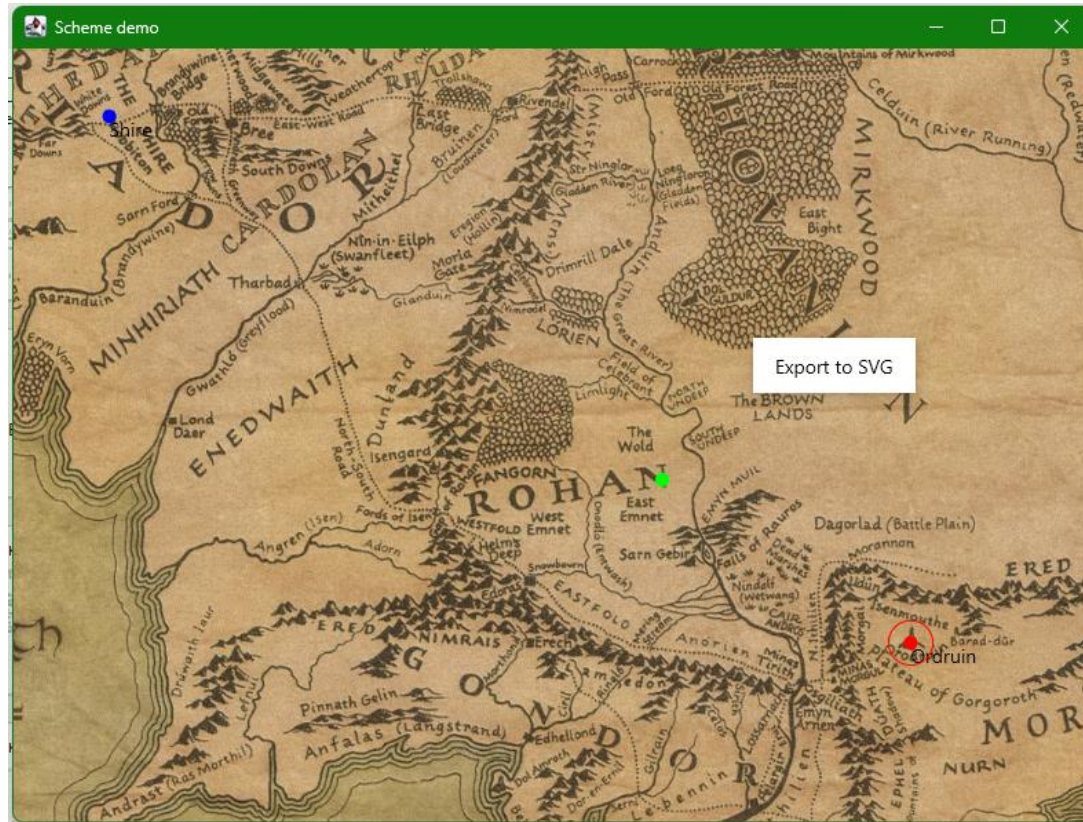
```
public fun FeatureStateSnapshot<XY>.exportToSvg(  
    viewPoint: ViewPoint<XY>,  
    width: Double,  
    height: Double,  
    path: java.nio.file.Path,  
) {  
    val svgString: String = generateSvg(viewPoint, width, height)  
    SVGUtils.writeToSVG(pathToFile(), svgString)  
}
```

<https://www.jfree.org/jfreesvg/>

<https://github.com/SciProgCentre/maps-kt/tree/dev/maps-kt-scheme/src/jvmMain/kotlin/center/sciprogram/maps/svg>

Canvas SVG backend

Screenshot



SVG



А теперь картинки!

```
public fun <T : Any> FeatureDrawScope<T>.drawFeature(  
    feature: Feature<T>,  
) : Unit {  
    val color = feature.color ?: Color.Red  
    val alpha = feature.attributes[AlphaAttribute] ?: 1f  
  
    when (feature) {  
        ...  
  
        is ScalableImageFeature -> {  
            val rect = feature.rectangle.toDpRect().toRect()  
            val offset = rect.topLeft  
  
            translate(offset.x, offset.y) {  
                with(painter) {  
                    draw(rect.size)  
                }  
            }  
        }  
    }  
}
```

Вот такая конструкция
используется для рисовки
картинок.

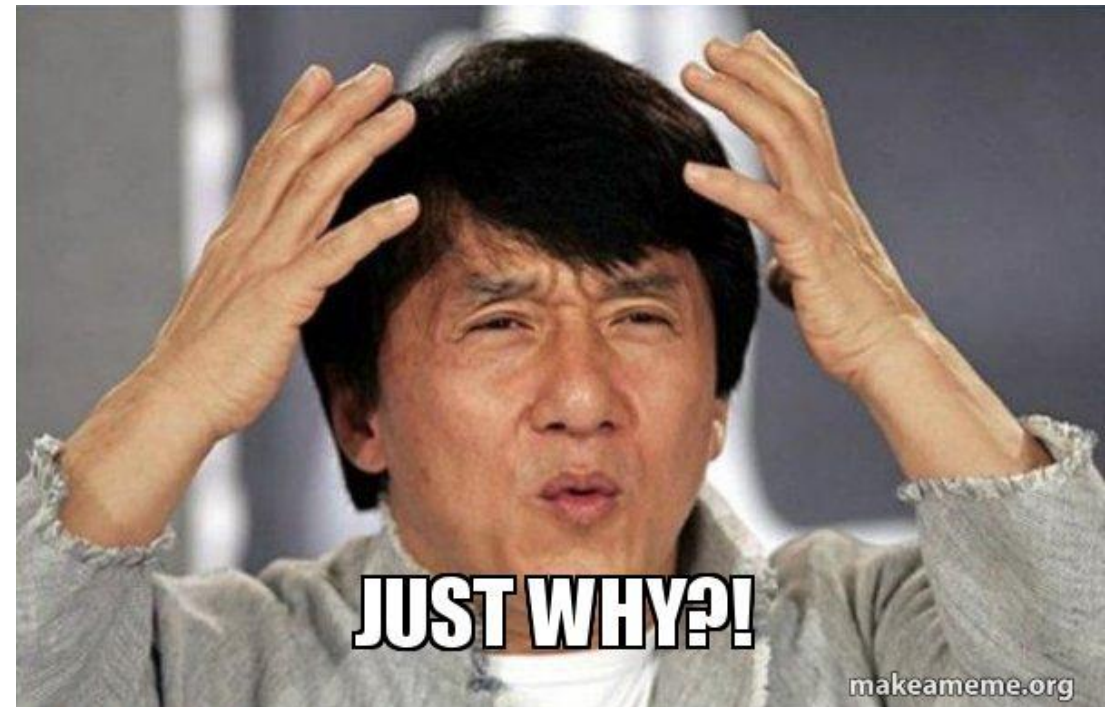
Откуда берется Painter?

```
@OptIn(ExperimentalComposeUiApi::class)  
@Composable  
fun painterResource(  
    resourcePath: String  
) : Painter = painterResource(  
    resourcePath,  
    ResourceLoader.Default  
)
```

Кто видит проблему?

А теперь картинки!

- Painter можно получить только из Composable функции
- А используется он в не Composable



```
val painterCache: Map<PainterFeature<T>, Painter> = features.features.flatMap {  
    if (it is FeatureGroup) it.features else listOf(it)  
}.filterIsInstance<PainterFeature<T>>().associateWith { it.getPainter() }
```

Про клики

```
public fun <T : Any> Modifier.canvasControls(
    state: CanvasState<T>,
    features: FeatureGroup<T>,
): Modifier = with(state){
    pointerInput(Unit) {
        detectClicks(
            onDoubleClick = if (viewConfig.zoomOnDoubleClick) {
                ""
            } else null,
            onClick = { event ->
                ""
            }
        )
    }.pointerInput(Unit) {
        awaitPointerEventScope {
            while (true) {
                val event: PointerEvent = awaitPointerEvent()
                event.changes.forEach { change ->
                    ""
                }
            }
        }
    }
}
}
```

- + Встроенная поддержка корутин.
- + Готовые обработчики для разных типов движений.

- Если нет готового, то беда...

Вся промежуточная логика
internal, так что
переиспользовать ее нельзя

```

31 internal suspend fun PointerInputScope.detectClicks(
32     onDoubleClick: (Density.(PointerEvent) -> Unit)? = null,
33     onLongClick: (Density.(PointerEvent) -> Unit)? = null,
34     onPress: suspend PressGestureScope.(event: PointerEvent) -> Unit = NoPressGesture,
35     onClick: (Density.(PointerEvent) -> Unit)? = null,
36 ): Unit = coroutineScope {
37     // special signal to indicate to the sending side that it shouldn't intercept and consume
38     // cancel/up events as we're only require down events
39     val pressScope = PressGestureScopeImpl( density: this@detectClicks)
40
41     awaitEachGesture {
42         val down : PointerEvent = awaitFirstDownEvent()
43         down.consume()
44
45         pressScope.reset()
46         if (onPress !== NoPressGesture) launch {
47             pressScope.onPress(down)
48         }
49         val longPressTimeout : Long = onLongClick?.let {
50             viewConfiguration.longPressTimeoutMillis
51         } ?: (Long.MAX_VALUE / 2)
52         var upOrCancel: PointerEvent? = null
53         try {
54             // wait for first tap up or long press
55             upOrCancel = withTimeout(longPressTimeout) {
56                 waitForUpOrCancellation()
57             }
58             if (upOrCancel == null) {
59                 pressScope.cancel() // tap-up was canceled
60             } else {
61                 upOrCancel.consume()
62                 pressScope.release()
63             }
64         } catch (_: PointerEventTimeoutCancellationException) {
65             onLongClick?.invoke(this, down)
66             consumeUntilUp()
67             pressScope.release()
68         }
69
70         if (upOrCancel != null) {
71             // tap was successful.
72             if (onDoubleClick == null) {

```

```

92 suspend fun PointerInputScope.detectTapGestures(
93     onDoubleTap: ((Offset) -> Unit)? = null,
94     onLongPress: ((Offset) -> Unit)? = null,
95     onPress: suspend PressGestureScope.(Offset) -> Unit = NoPressGesture,
96     onTap: ((Offset) -> Unit)? = null
97 ) = coroutineScope {
98     // special signal to indicate to the sending side that it shouldn't intercept and consume
99     // cancel/up events as we're only require down events
100     val pressScope = PressGestureScopeImpl( density: this@detectTapGestures)
101
102     awaitEachGesture {
103         val down : PointerInputChange = awaitFirstDown()
104         down.consume()
105         launch {
106             pressScope.reset()
107         }
108         if (onPress !== NoPressGesture) launch {
109             pressScope.onPress(down.position)
110         }
111         val longPressTimeout : Long = onLongPress?.let {
112             viewConfiguration.longPressTimeoutMillis
113         } ?: (Long.MAX_VALUE / 2)
114         var upOrCancel: PointerInputChange? = null
115         try {
116             // wait for first tap up or long press
117             upOrCancel = withTimeout(longPressTimeout) {
118                 waitForUpOrCancellation()
119             }
120             if (upOrCancel == null) {
121                 pressScope.cancel() // tap-up was canceled
122             } else {
123                 upOrCancel.consume()
124                 launch {
125                     pressScope.release()
126                 }
127             }
128         } catch (_: PointerEventTimeoutCancellationException) {
129             onLongPress?.invoke(down.position)
130             consumeUntilUp()
131             launch {
132                 onLongPress?.invoke(down.position)
133                 consumeUntilUp()
134             }

```

Видите разницу?

Я тоже не вижу

Но приходится писать

Приложения

Схемы

Точка входа в приложение

```
fun main() = application {  
    Window(onCloseRequest = ::exitApplication, title = "Joker2023 demo", icon = painterResource("SPC-logo.png")) {  
        MaterialTheme {
```

Desktop – специфичное окружение

Компонент
схемы

```
        SchemeView(  
            initialRectangle = Rectangle(XY(0f, 0f), XY(1734f, 724f)),  
            config = ViewConfig(  
                onClick = { _, pointer -> println("(${pointer.focus.x}, ${pointer.focus.y})") }  
            )  
        ) {  
            background(1734f, 724f, id = "background") { painterResource("joker2023.png") }  
            group(id = "hall_1") {  
                polygon(  
                    listOf(  
                        XY(1582.0042, 210.29636),  
                        XY(1433.7021, 127.79796),  
                        XY(1370.7639, 127.79796),  
                        XY(1315.293, 222.73865),  
                        XY(1314.2262, 476.625),  
                        XY(1364.3635, 570.4984),  
                        XY(1434.7689, 570.4984),  
                        XY(1579.8469, 493.69244),  
                    )  
                ).modifyAttributes {  
                    ColorAttribute(Color.Blue)  
                    AlphaAttribute(0.4f)  
                }.onClick {  
                    println("hall_1")  
                }  
            }  
        }  
    }  
}
```

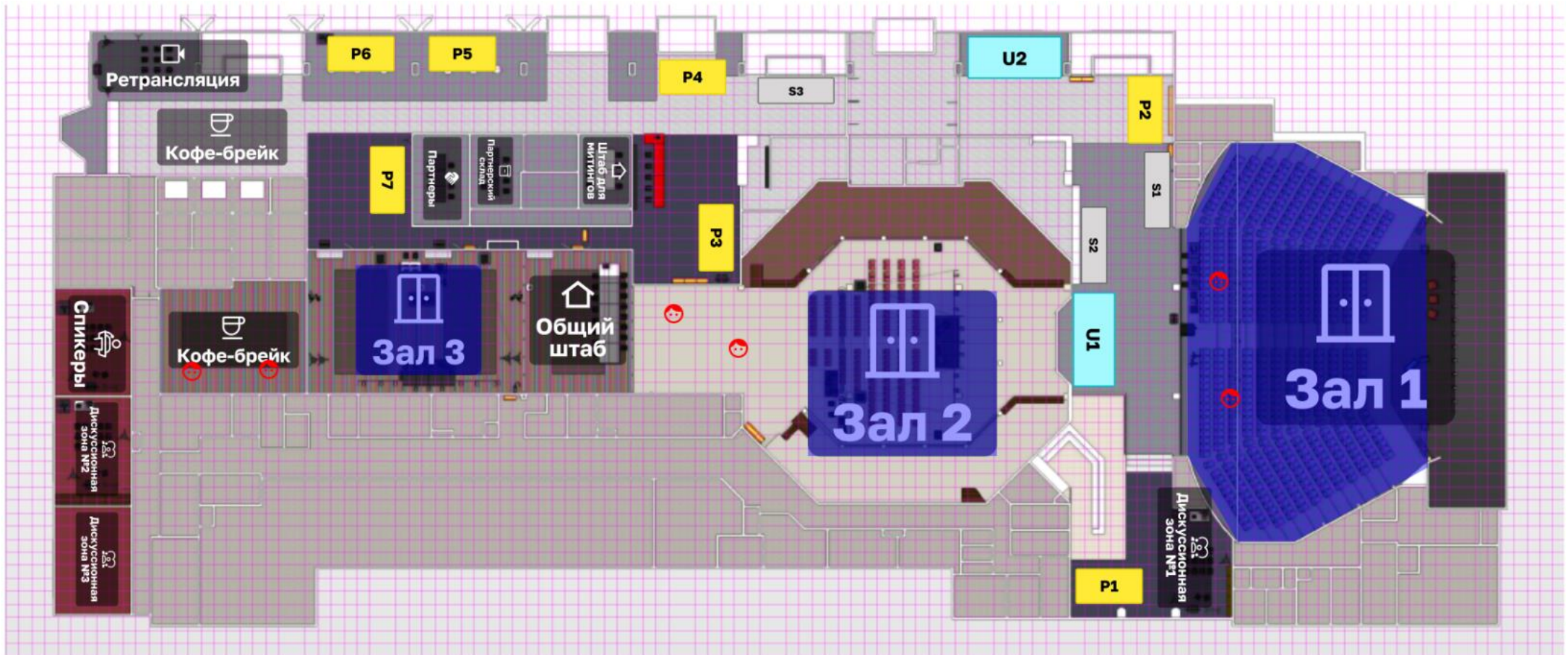
Здесь Compose
мир кончился,
началась канва

Набрасываем
фичи или группы
фич на карту.

Фичи можно собирать в
отдельных модулях или
десериализовывать.

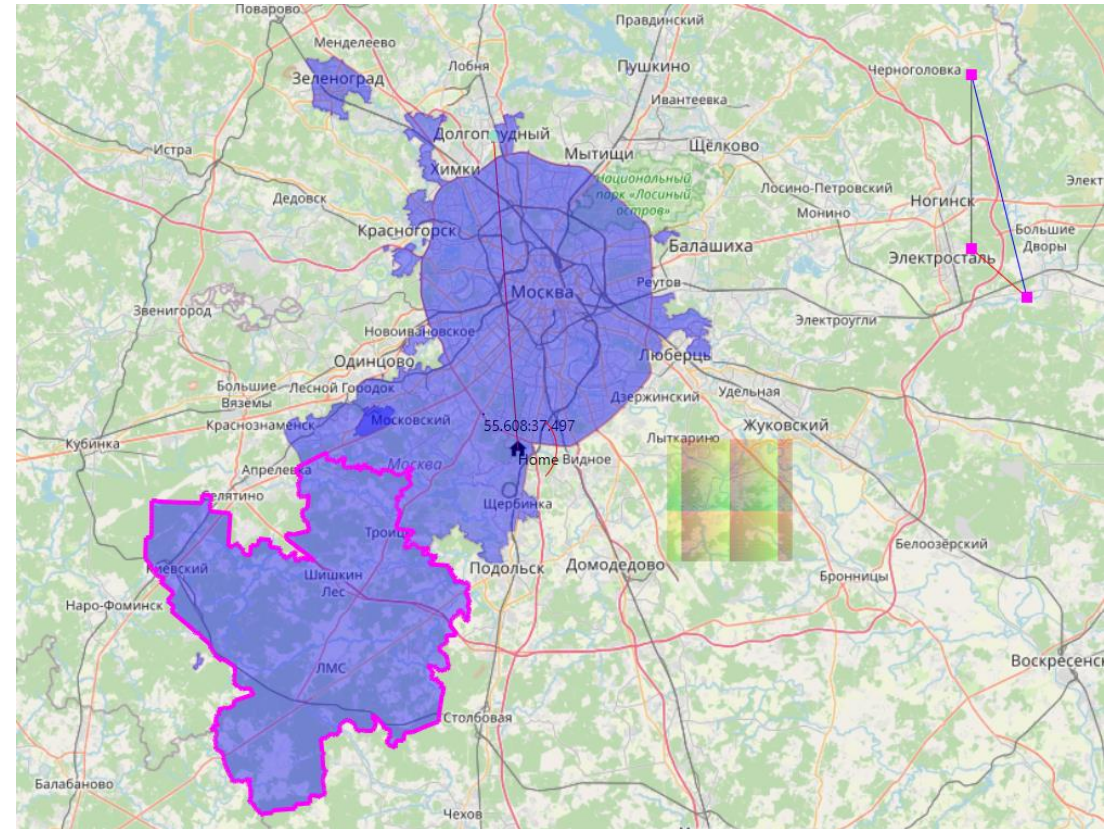
Схемы

<https://github.com/SciProgCentre/maps-kt/blob/dev/demo/scheme/src/jvmMain/kotlin/joker2023.kt>



Карты

- Базовые примитивы (линии, маркеры, дуги).
- GeoJson.
- Пиксельные карты.
- Интерактивные элементы



Интерактивность

Берем существующие
маркеры

Автоматически смещаем
соединяющую линию

Добавляем наблюдателя за их
смещением

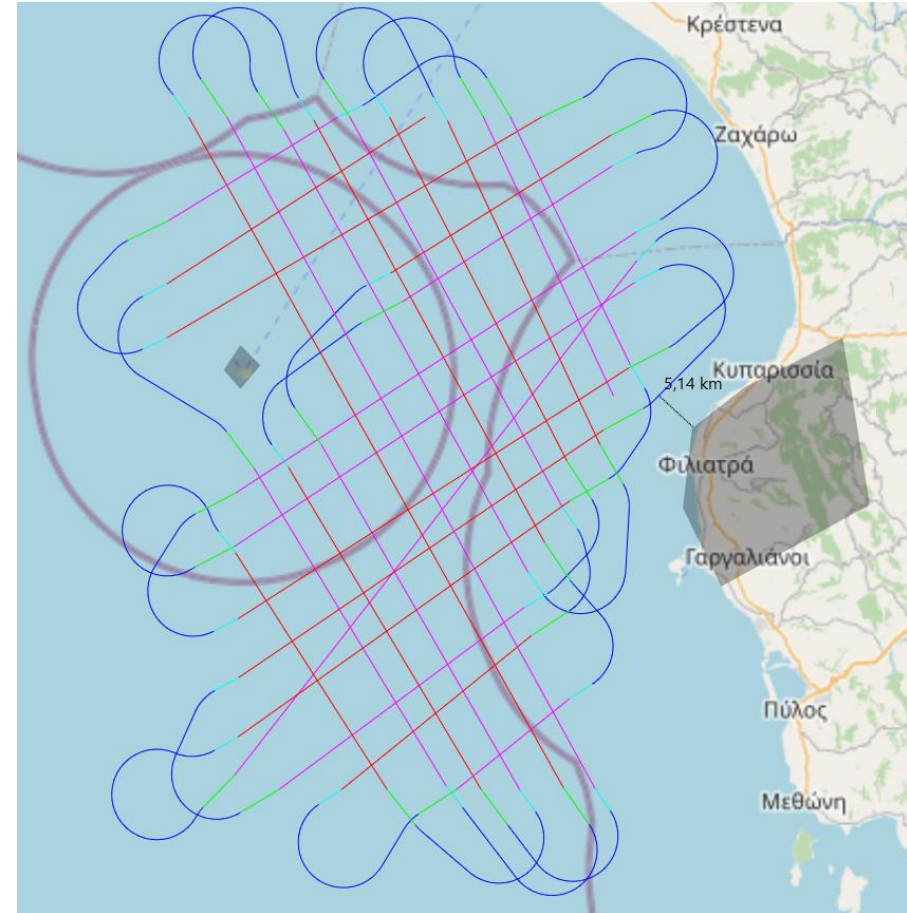
```
public fun <T : Any> FeatureGroup<T>.draggableLine(
    aId: FeatureRef<T, MarkerFeature<T>>,
    bId: FeatureRef<T, MarkerFeature<T>>,
    id: String? = null,
): FeatureRef<T, LineFeature<T>> {
    var lineId: FeatureRef<T, LineFeature<T>>? = null

    fun drawLine(): FeatureRef<T, LineFeature<T>> {
        val currentId = feature(
            lineId?.id ?: id,
            LineFeature(
                space,
                aId.resolve().center,
                bId.resolve().center,
                Attributes {
                    ZAttribute(-10f)
                    lineId?.attributes?.let { from(it) }
                }
            )
        )
        lineId = currentId
        return currentId
    }

    aId.draggable { _, _ -> drawLine() }
    bId.draggable { _, _ -> drawLine() }
    return drawLine()
}
```

LotSeis

- Оптимизация траекторий сейсморазведывательных кораблей.
- Динамическая отрисовка траекторий и полигонов.
- Пользовательский интерфейс.
- Все это на десктопе (на корабле может быть плохо с вебом).



Чего не хватает для счастья?

- Трансформация координат для произвольных EPSG.
- Импорт и экспорт форматов (например SHP – файлы).

<https://geotools.org/>



Java фреймворк для работы с системами координат и картами.

Что дальше?

Android

- Почему библиотека не поддерживает андроид?
- Потому что мне было лень настраивать под него сборку и тестирование.
- Потому что на андроиде уже и так они есть.



Web/Wasm

Compose-html

- <https://github.com/JetBrains/compose-multiplatform/tree/master/examples/html>
- Позволяет использовать стейт менеджмент Compose с html тэгами.
- Канва тоже есть, но глючная. В частности не работают API Google для работы с текстом.

Compose-web

- <https://github.com/Kotlin/kotlin-wasm-examples>
- Использует Skiko/Wasm для рендера компонентов напрямую (как Flutter).
- Для работы требуется добавить поддержку Wasm таргета в мультиплатформные библиотеки Kotlin.

Compose-html

<https://github.com/JetBrains/compose-multiplatform/blob/master/examples/html/with-react/src/jsMain/kotlin/ReactInComposeApp.kt>

```
fun reactInComposeAppExample() {  
    var videoUrl by mutableStateOf("")  
    renderComposable(rootElementId = "root") {  
        A(href = "${window.location.origin}?app=reactApp") {  
            Text("GO TO COMPOSE IN REACT EXAMPLE")  
        }  
        Div {  
            videos.forEachIndexed { ix, url ->  
                Button(  
                    attrs = {  
                        onClick { videoUrl = url }  
                        style {  
                            margin(10.px)  
                        }  
                    }  
                ) { Text("Video ${ix + 1}") }  
            }  
            YoutubeReactPlayerWrapper(videoUrl)  
        }  
    }  
}
```

- Drop-in replacement для React.
- Работает с Html тэгами, не требует Skia и дружит с SEO.
- Совместим с React. Можно вставлять React в Compose и обратно.
- Набор компонентов отличается от Compose Desktop.
- Зато можно использовать JS библиотеки.

Выводы

- Картография куда сложнее, чем может показаться.
- Но ничего невозможного.
- Compose полностью готов для использования на Desktop. И скоро будет можно для Wasm и Ios.
- Но иногда торчат уши андроида и Google API.

(Почти) весь код тут:

<https://github.com/SciProgCentre/maps-kt>