```kotlin
class DataConversion {
    companion object {
        fun conversion(scoreData: Array<Array<Int>>, encampmentData: Array<Array<Int>>, agents: Map<String, MegurimasuSimulator.Agent>): String {
            var conversionStr = ""
            val width = scoreData[0].size
            val height = scoreData.size

            // ステージデータ(幅, 高さ)
            conversionStr = "${width.toString(36)}:${height.toString(36)}:"

            // 陣地データ
            encampmentData.forEach { array ->
                val binStr = array
                        .map{ String.format("%2s", it.toString(2)).replace(" ", "0") }
                        .reduce { s1, s2 -> s1 + s2 }
                conversionStr += Integer.parseInt(binStr, 2).toString(36) + ":"
            }

            // エージェントデータ
            conversionStr += agents
                    .map{ it.value.x.toString(36) + it.value.y.toString(36) + ":" }
                    .reduce { s1, s2 -> "$s1$s2" }

            return conversionStr
        }

        @Suppress("UNCHECKED_CAST")
        fun deconversion(target: String): Map<String, Any>{
            val splitTarget = target.split(":")

            // ステージ情報(幅, 高さ)
            val width = numAtoB(splitTarget[0], 36, 10)
            val height = numAtoB(splitTarget[1], 36, 10)

            // 陣地データ
            // 36進数を2進数に変換した後, 2個ずつ数字を連結してそれを10進数に直す
            val encampmentData = Array(height) { _ -> Array(width){0}}
            for(i in 0 until height){
                var binStr = Integer.parseInt(splitTarget[i+2], 36).toString(2)
                binStr = String.format("%"+(width*2)+"s", binStr).replace(" ", "0")

                for(charIdx in 0 until width*2 step 2){
                    encampmentData[i][charIdx/2] = "${binStr[charIdx]}${binStr[charIdx+1]}".toInt(2)
                }
            }

            // エージェントデータ
            // それぞれの対応桁を取り出す
            val agentPos = mutableMapOf<String, Array<Int>>()
            val agentNames = listOf("A_1", "A_2", "B_1", "B_2")
            for(i in 0 until 4){
                val agent = splitTarget[i+height+2]
                val agentX = numAtoB(agent[0].toString(), 36, 10)
                val agentY = numAtoB(agent[1].toString(), 36, 10)
                agentPos[agentNames[i]] = arrayOf(agentX, agentY)
            }

            return mapOf(
                    "width" to width,
                    "height" to height,
                    "encampmentData" to encampmentData,
                    "agentPos" to agentPos
            )
        }

        private fun numAtoB(numStr: String, A: Int, B:Int): Int{
            return Integer.parseInt(numStr, A).toString(B).toInt()
        }
    }
}
```

```kotlin
var qrData: String? = null
var posData: String? = null
var depth = 3
var probability = arrayOf(4, 1, 0)

fun main(args: Array<String>){
    TCPConnectionManager("localhost", 6666, ::tcpReceiver).receiveStart()

    // QRデータ待機
    println("Please Input QR Data")
    while(qrData == null){ Thread.sleep(5) }
    println("Received QR Data")

    // スコアデータとエージェント初期位置取得
    val qrParser = QRParser(qrData!!)
    val scoreData = qrParser.getScoreData()
    val agentPos = qrParser.getAgentPos()

    // MegurimasuSimulator初期化
    val megurimasu = MegurimasuSimulator(agentPos, scoreData)

    // 思考ループ
    val doLoop = true
    while(doLoop){
        // 最善手探索
        println("Searching Best Behavior...")
        val (maxScore, bestBehavior) = searchBestBehavior(megurimasu, depth, probability)
        printInfo(maxScore, bestBehavior, megurimasu)

        // 相手の行動が入力されるのを待機
        println("Please Input Opponent Action")
        posData = "Waiting"
        while(posData == "Waiting"){ Thread.sleep(5) }
        println("Received Opponent Action Data")

        // 相手の行動を取得
        val agentB1Action = posData!!.split(":")[0].toInt()
        val agentB2Action = posData!!.split(":")[1].toInt()
        posData = null

        println("$agentB1Action, $agentB2Action")

        // 場面更新
        val behavior = mapOf(
                "A_1" to bestBehavior["A_1"]!!, "A_2" to bestBehavior["A_2"]!!,
                "B_1" to agentB1Action, "B_2" to agentB2Action
        )
        megurimasu.action(behavior)
    }
}

fun printInfo( maxScore: Int, bestBehavior: Map<String, Int>, megurimasu: MegurimasuSimulator){
    println()
    println("---")
    println("BestBehavior: A -> ${bestBehavior["A_1"]}, B -> ${bestBehavior["A_2"]}")
    println("MaxScore: $maxScore")
    println("EncampmentData: ")
    megurimasu.encampmentData.forEach { it.forEach { print("$it ") }; println() }
    println("AgentPos: ")
    megurimasu.agents.forEach { key, pos -> println("$key -> (${pos.x}, ${pos.y})") }
    println("Score: A ${megurimasu.calScore()["A"]} vs ${megurimasu.calScore()[
"B"]} B")
    println("---")
}

fun tcpReceiver(text: String) {
    if(text == "close"){ System.exit(0) }

    val dividedText = text.split("@")
    val type = dividedText[0]
    val data = dividedText[1]

    when(type){
        "QRData" -> qrData = data
        "OpponentPos" ->{
            if(posData != "Waiting"){ return }
            posData = data
            println("Input")
        }
    }
}
```

```kotlin
import kotlin.math.abs

class MegurimasuSimulator(agentInitPos: Map<String, Array<Int>>, var scoreData: Array<Array<Int>>){
    private var width = scoreData[0].size
    private var height = scoreData.size
    var agents = agentInit(agentInitPos)
    var encampmentData = arrayOf<Array<Int>>()

    inner class Agent(private val agentName: String, var x: Int, var y: Int) {
        fun action(type: Int): Boolean {
            if(!canAction(type)) return false

            when(type){
                // 移動
                in 0..7 ->{
                    val movedValues = takeActionPos(type)
                    x = movedValues["x"]!!
                    y = movedValues["y"]!!
                    encampmentData[y][x] = getTeamID(agentName)
                }
                // パネル除去
                in 10..17 ->{
                    val movedValues = takeActionPos(type)
                    encampmentData[movedValues["y"]!!][movedValues["x"]!!] = 0
                }
            }

            return true
        }

        private fun canAction(type: Int): Boolean {
            if(type !in 0..8 && type !in 10..18) return false

            val (xCopy, yCopy) = getActionPos(x, y, type%10)

            if(!isWithInRange(xCopy, yCopy)){ return false }
            val encampment = encampmentData[yCopy][xCopy]

            when(type){
                // 移動の場合: 移動先が敵の陣地であれば(=自分の陣地でないかつ空白ではない)場合は移動不許可
                in 0..8 -> {
                    if(encampment != getTeamID(agentName) && encampment != 0){ return false }
                }

                // パネル除去の場合: 移動先が空白の場合は除去不許可
                else -> if(encampment == 0){ return false }
            }

            return true
        }

        fun takeActionPos(type: Int): Map<String, Int>{
            // typeが範囲外であったり行動できなかったりする場合は計算せずに返す
            if(type !in 0..8 && type !in 10..18){ return mapOf("x" to 0, "y" to 0) }
            if(!canAction(type)){ return mapOf("x" to x, "y" to y)}

            val (retX, retY) = getActionPos(x, y, type%10)
            return mapOf("x" to retX, "y" to retY)
        }
    }

    init{
        // 盤面初期化
        encampmentData = Array(scoreData.size) { _ -> Array(scoreData[0].size){0}}
        agents.forEach { key, pos ->
            encampmentData[pos.y][pos.x] = getTeamID(key)
        }
    }

    private fun agentInit(agentInitPos: Map<String, Array<Int>>): Map<String, Agent>{
        val agents = mutableMapOf<String, Agent>()
        agentInitPos.forEach { key, pos ->
            agents[key] = Agent(key, pos[0], pos[1])
        }

        return agents
    }

    fun action(behavior: Map<String, Int>){
        // 行動後の座標を取得する
        val takeActionPositions = mutableMapOf<String, Int>()
        actionSimulation(behavior).forEach { agentName, pos ->
            takeActionPositions[agentName] = pos["x"]!!*100 + pos["y"]!!
        }

        // エージェントを行動させる(重複してないかつ条件を満たしたものだけ)
        duplicateDetection(takeActionPositions)
                .forEach { agentName, isDuplicate ->
                    if(isDuplicate || !agents.containsKey(agentName) || !behavior.containsKey(agentName)) {
                        return@forEach
                    }
                    agents[agentName]!!.action(behavior[agentName]!!)
                }
    }

    private fun actionSimulation(behavior: Map<String, Int>): Map<String, Map<String, Int>>{
        val takeActionPositions = mutableMapOf<String, Map<String, Int>>()
        behavior.forEach { agentName, type ->
            if(!agents.containsKey(agentName)){ return@forEach }
            takeActionPositions[agentName] = agents[agentName]!!.takeActionPos(type)
        }

        return takeActionPositions
    }

    private fun duplicateDetection(target: Map<String, Int>): Map<String, Boolean>{
        // 重複があればduplicateCheckMapの値がtrueになる
        val duplicateCheckMap = mutableMapOf<String, Boolean>()
        target.forEach { agentName, value ->
            duplicateCheckMap[agentName] = target.count { it.value == value} >= 2
        }

        return duplicateCheckMap
    }

    fun calScore(): Map<String, Int>{
        val score = mutableMapOf("A" to 0, "B" to 0)
        var flatScoreDara = scoreData.flatten().toIntArray()

        // パネルスコア
        score.forEach { key, _ ->
```

```kotlin
            val teamID = getTeamID(key)
            score[key] = flatScoreDara
                    .filterIndexed { idx, _ -> encampmentData[idx/width][idx%width] == teamID; }
                    .sum()
        }

        // 陣地スコア
        flatScoreDara = flatScoreDara.map { abs(it) }.toIntArray()
        arrayOf("A", "B").forEach { teamIDStr ->
            var fillEncampment: Array<Array<Int>>? = Array(height) { _ -> Array(width){0} }
            val teamID = getTeamID(teamIDStr)

            // 外周を除いた全ての座標を起点として陣地探索をする(再帰)
            for(y in 1 until height-1){
                for(x in 1 until width-1) {
                    if(fillEncampment!![y][x] == 1 || encampmentData[y][x] == teamID){ continue }

                    // 探索結果がnullなら探索失敗，fillEncampmentを元に戻す
                    val copyFillEncampment = fillEncampment.map{ it.clone() }.toTypedArray()
                    fillEncampment = recursionSearch(x, y, teamID, fillEncampment)?: copyFillEncampment
                }
            }

            // 探索結果をスコアに反映
            val encScore = flatScoreDara
                    .filterIndexed { idx, _ -> fillEncampment!![idx/width][idx%width] == 1 }
                    .sum()
            score[teamIDStr] = score[teamIDStr]!!.plus(encScore)
        }

        return score
    }

    private fun recursionSearch(x: Int, y: Int, teamID: Int, argFillEncampment: Array<Array<Int>>?): Array<Array<Int>>?{
        if(x == 0 || x == width-1 || y == 0 || y == height-1 || argFillEncampment == null){
            return null
        }

        // 探索済みにする
        argFillEncampment[y][x] = 1

        var fillEncampment = argFillEncampment
        val moveXList = listOf(x, x, x-1, x+1)
        val moveYList = listOf(y-1, y+1, y, y)

        for(i in 0 until 4){
            val _x = moveXList[i]
            val _y = moveYList[i]

            // 移動先がステージ内 and 探索先の場所が自分の陣地でない and すでに探索済みでなければ探索続行
            // nullが返ってきたらそのまま返す
            if(isWithInRange(_x, _y) && encampmentData[_y][_x] != teamID && fillEncampment!![_y][_x] == 0){
                fillEncampment = recursionSearch(_x, _y, teamID, fillEncampment)
                if(fillEncampment == null){ return null }
            }
        }

        return fillEncampment
    }

    private fun isWithInRange(x: Int, y: Int): Boolean{
        return (x in 0..(width - 1)) && (y in 0..(height-1))
    }

    fun conversion(): String{
        return DataConversion.conversion(scoreData, encampmentData, agents)
    }

    @Suppress("UNCHECKED_CAST")
    fun deconversion(target: String){
        val stageData = DataConversion.deconversion(target)

        width = stageData["width"] as Int
        height = stageData["height"] as Int
        encampmentData = stageData["encampmentData"] as Array<Array<Int>>
        agents = agentInit(stageData["agentPos"] as Map<String, Array<Int>>)
    }
}
```

```kotlin
class QRParser(qrText: String){
    private val qrData = qrText.split(":")
    private val stageSizeInfo = qrData[0].split(" ")
    private val height = stageSizeInfo[0].toInt()

    fun getAgentPos(): Map<String, Array<Int>>{
        // わかりやすさを優先してループを使わない
        val agentA1Pos = qrData[height+1].split(" ")
        val agentA1PosY = agentA1Pos[0].toInt() - 1
        val agentA1PosX = agentA1Pos[1].toInt() - 1

        val agentA2Pos = qrData[height+2].split(" ")
        val agentA2PosY = agentA2Pos[0].toInt() - 1
        val agentA2PosX = agentA2Pos[1].toInt() - 1

        return mapOf(
                "A_1" to arrayOf(agentA1PosX, agentA1PosY),
                "A_2" to arrayOf(agentA2PosX, agentA2PosY),
                "B_1" to arrayOf(agentA1PosX, agentA2PosY),
                "B_2" to arrayOf(agentA2PosX, agentA1PosY)
        )
    }

    fun getScoreData(): Array<Array<Int>>{
        val scoreData = arrayListOf<Array<Int>>()

        qrData.forEachIndexed{ idx, line ->
            if(idx == 0 || height < idx){ return@forEachIndexed }

            val scoreLine = line
                    .split(" ")
                    .map { it -> it.toInt() }
                    .toTypedArray()
            scoreData.add(scoreLine)
        }

        return scoreData.toTypedArray()
    }
}
```

```
import java.lang.IndexOutOfBoundsException
import java.util.Random;
import kotlin.math.max
import kotlin.system.measureTimeMillis

val random = Random()

// 再帰でより良い手を探す
fun searchBestBehavior(megurimasu: MegurimasuSimulator, depth: Int, probability: Array<Int>): Pair<Int, Map<String, Int>>{
    // 葉ならスコアを計算して返す
    if(depth == 0){
        val score = megurimasu.calScore()
        return Pair(score["A"]!! - score["B"]!!, mapOf())
    }

    // 次の手を列挙(A)
    val agentsActionA = listOf("A_1", "A_2")
            .map{ agentName ->
                val bruteforce = strategyOfBruteForce(megurimasu, agentName, probability[0])
                val stalker = strategyOfStalker(megurimasu, agentName, probability[1])
                val prayToGod = strategyOfPrayToGod(probability[2])

                agentName to bruteforce + stalker + prayToGod
            }
            .toMap()

    // 次の手を列挙(B)
    val agentsActionB = listOf("B_1", "B_2")
            .map{ agentName ->
                val randBrute = random.nextInt(probability.sum())
                val randStalker = random.nextInt(probability.sum() - randBrute)
                val randGod = probability.sum() - randBrute - randStalker

                val bruteforce = strategyOfBruteForce(megurimasu, agentName, randBrute)
                val stalker = strategyOfStalker(megurimasu, agentName, randStalker)
                val prayToGod = strategyOfPrayToGod(randGod)

                agentName to bruteforce + stalker + prayToGod
            }
            .toMap()

    // それぞれのエージェントが選択した手を合わせて次の盤面を決める
    val agentsAction = agentsActionA + agentsActionB
    val nextBehaviors = arrayListOf<Map<String, Int>>()
    val total = probability.sum()
    for(i: Int in 0 until total * total){
        nextBehaviors.add(mapOf(
                "A_1" to agentsAction["A_1"]!![i / total],
                "A_2" to agentsAction["A_2"]!![i % total],
                "B_1" to agentsAction["B_1"]!![i / total],
                "B_2" to agentsAction["B_2"]!![i % total]
        ))
    }

    // リードが一番大きくなるような手を見つける
    val nowBoard = megurimasu.conversion()
    var maxScore = -99
    val bestBehavior = nextBehaviors
            .asSequence()
            .maxBy { it ->
                megurimasu.action(it)
                val (score, _) = searchBestBehavior(megurimasu, depth - 1, probability)
                megurimasu.deconversion(nowBoard)

                maxScore = max(score, maxScore)
                score
            }!!

    return Pair(maxScore, mapOf("A_1" to bestBehavior["A_1"]!!, "A_2" to bestBehavior["A_2"]!!))
}

fun strategyOfBruteForce(megurimasu: MegurimasuSimulator, agentName: String, num: Int): List<Int>{
    val actionedScoreList = arrayListOf<Array<Int>>()
    for(i in 0..7){
        var _i = i
        val movableList = listOf(0, 1, 2, 3, 4, 5, 6, 7).filter { it -> it != (i+4)%8 }

        // 現在の盤面から1つ手を選択した時，それに対して新たに手を選択した合計2手のスコアを計算して集計する
        // 必要なのは1手後の情報だけなので，2手後の選択については特に選択した手の保持などをしない
        val maxValue = arrayOf(-99, 0)
        movableList.forEach{ type ->
            // 必要な座標を取得
            val agentX = megurimasu.agents[agentName]!!.x
            val agentY = megurimasu.agents[agentName]!!.y
            val (actionX, actionY) = getActionPos(agentX, agentY, i)
            val (actionXTwo, actionYTwo) = getActionPos(actionX, actionY, type)

            // 範囲外
            try { megurimasu.encampmentData[actionY][actionX]; megurimasu.encampmentData[actionYTwo][actionXTwo]}
            catch (e: ArrayIndexOutOfBoundsException){ return@forEach }

            // 既に自分の陣地であるか敵の陣地だった場合は負の評価を与えたのちに集計する
            var score = megurimasu.scoreData[actionY][actionX] + megurimasu.scoreData[actionYTwo][actionXTwo]
            when(megurimasu.encampmentData[actionY][actionX]){
                0 -> { }
                getTeamID(agentName) -> score = 0
                else -> {score -= 3; _i += 10}
            }

            // 最大値更新
            if(maxValue[0] < score){
                maxValue[0] = score
                maxValue[1] = _i
            }

            if(_i > 10){ _i %= 10 }
        }

        actionedScoreList.add(maxValue)
    }

    // スコアを降順にソートして指定数だけ選択してそのidxを返す
    return actionedScoreList
            .asSequence()
            .sortedByDescending { ( score, _) -> score }
            .take(num)
            .map { it[1] }
            .toList()
```

```kotlin
}

fun strategyOfStalker(megurimasu: MegurimasuSimulator, agentName: String, num: Int): List<Int>{
    // 存在しないエージェントの名前が引数で与えられたとき時は全てが8のListを返す
    if(agentName !in megurimasu.agents.keys){
        return Array(num){ _ -> 8}.toList()
    }

    // 一番近い敵エージェントを探す
    val enemyTeam = if("A" in agentName) "B" else "A"
    val agent = megurimasu.agents[agentName]!!
    val enemyAgents = arrayOf(megurimasu.agents["${enemyTeam}_1"]!!, megurimasu.agents["${enemyTeam}_2"]!!)
    val minDistAgent = enemyAgents
            .minBy { calDist(agent.x, agent.y, it.x, it.y) }!!

    // 一番近いエージェントに近づくための行動タイプを探す
    val meAgentDegree = calDegree2Points(agent.x, agent.y, minDistAgent.x, minDistAgent.y).toInt()
    val optimalActionType = (meAgentDegree % 360 / 45 + 2) % 8

    // 評価の高いものから順にListに放り込む
    val retList = mutableListOf(optimalActionType)
    for(i: Int in 1..4){
        retList.add((optimalActionType + i + 8) % 8)
        retList.add((optimalActionType + (i * -1) + 8) % 8)
    }

    // 敵陣地だった場合はパネル除去を行うように
    retList.forEachIndexed { idx, elem  ->
        val agentX = megurimasu.agents[agentName]!!.x
        val agentY = megurimasu.agents[agentName]!!.y
        val (actionX, actionY) = getActionPos(agentX, agentY, elem)

        // 範囲外
        try { megurimasu.encampmentData[actionY][actionX]}
        catch (e: IndexOutOfBoundsException){ return@forEachIndexed }

        if(!listOf(0, getTeamID(agentName)).contains(megurimasu.encampmentData[actionY][actionX])){
            retList[idx] += 10
        }
    }

    return retList.take(num)
}

fun strategyOfPrayToGod(num: Int): List<Int>{
    // ランダムに値を選択してListに詰めて返す
    val retList = mutableListOf<Int>()
    for(i in 0 until num){
        var randValue = 0
        do{
            randValue = random.nextInt(8) + random.nextInt(2) * 10
        }while(retList.contains(randValue))

        retList.add(randValue)
    }

    return retList
}
```

```kotlin
import com.sun.jdi.connect.spi.ClosedConnectionException
import java.io.BufferedReader
import java.io.InputStreamReader
import java.lang.Exception
import java.net.Socket
import kotlin.concurrent.thread

class TCPConnectionManager(private val hostAddress: String, private val hostPort: Int, private val receiver: (String) -> Unit){

    var socket: Socket? = null
    init { initSocket() }

    private fun initSocket(){
        try {
            socket = Socket(hostAddress, hostPort)
            println("Socket Open")
        }
        catch (e: Exception) {
            e.printStackTrace()
        }
    }

    // データ受信開始
    fun receiveStart(){
        thread { receiveData() }
    }

    private fun closeSocket(){
        if(socket == null){ return }

        socket!!.close()
        println("Socket Closed")
        receiver("close")
    }

    private fun receiveData(){
        if(socket == null){ return }

        // データ受信処理
        //  - 受信したデータはレシーバ関数へ
        //  - 切断時にはCloseConnectionExceptionを投げる
        try {
             val reader = BufferedReader(InputStreamReader(socket!!.getInputStream()))
             while (true) {
                 val text = reader.readLine() ?: throw ClosedConnectionException()
                 receiver(text)
             }
         } catch (e: ClosedConnectionException){
             closeSocket()
         } catch (e: Exception) {
             e.printStackTrace()
             closeSocket()
         }
    }
}
```

```kotlin
import kotlin.math.PI
import kotlin.math.sqrt
import kotlin.math.atan2

fun getActionPos(x: Int, y: Int, type: Int): Pair<Int, Int>{
    if(type%10 !in 0..8 && type%10 !in 10..18){ return Pair(0, 0) }

    return Pair(
            x + movementValues[type]!!["x"]!!,
            y + movementValues[type]!!["y"]!!
    )
}

fun getTeamID(agentName: String): Int{
    return when(agentName){
        "A_1", "A_2", "A" -> 1
        "B_1", "B_2", "B" -> 2
        else -> 0
    }
}

fun calDist(x: Int, y: Int, x_1: Int, y_1:Int): Double{
    return sqrt( ((x-x_1) * (x-x_1) + (y-y_1) * (y-y_1)).toDouble() )
}

fun calDegree2Points(x: Double, y: Double, x_2: Double, y_2: Double): Double{
    var degree = atan2(y_2-y, x_2-x) * 180 / PI
    if(degree < 0){
        degree += 360
    }

    return degree
}

fun calDegree2Points(x: Int, y: Int, x_2: Int, y_2: Int): Double{
    return calDegree2Points(x.toDouble(), y.toDouble(), x_2.toDouble(), y_2.toD
ouble())
}

val movementValues = mapOf(
        8 to mapOf("x" to 0,  "y" to 0),
        0 to mapOf("x" to 0,  "y" to -1),
        1 to mapOf("x" to 1,  "y" to -1),
        2 to mapOf("x" to 1,  "y" to 0),
        3 to mapOf("x" to 1,  "y" to 1),
        4 to mapOf("x" to 0,  "y" to 1),
        5 to mapOf("x" to -1, "y" to 1),
        6 to mapOf("x" to -1, "y" to 0),
        7 to mapOf("x" to -1, "y" to -1)
)
```