Gruppo: I Feudatari

Componenti:
Giada Pica, Angelo Romano, Francesco Minotti

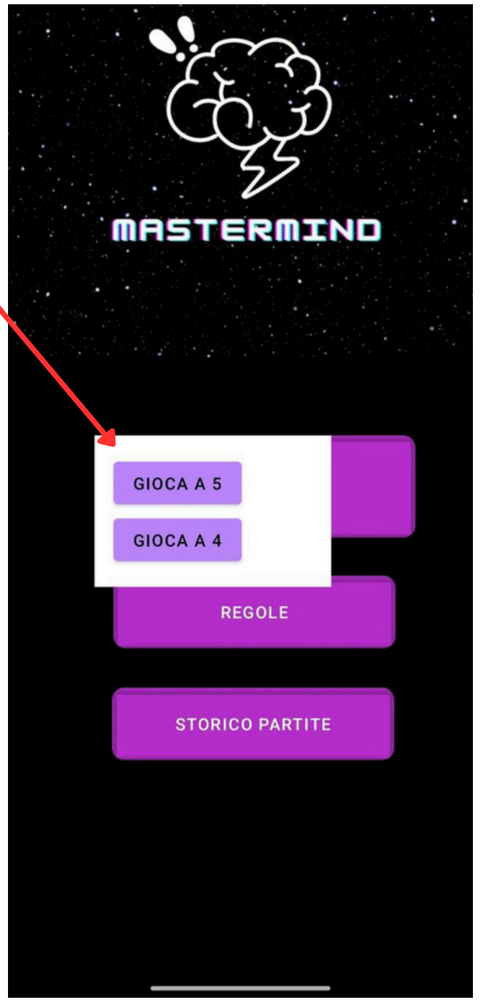
Menù iniziale:

Cliccando si aprirà un pop up per scegliere la modalità di gioco e iniziare una nuova partita

Qui sono presenti tutte le regole del gioco, con anche una breve spiegazione della plancia di gioco

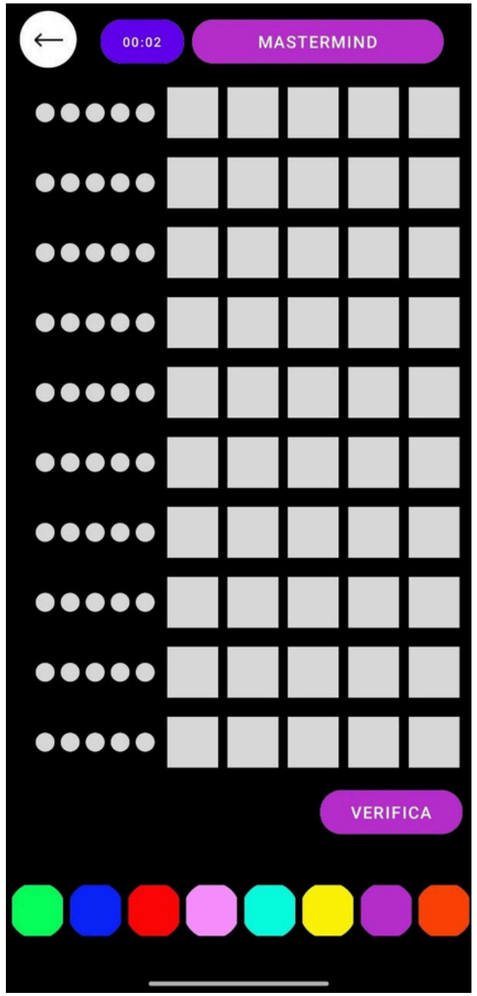
In questa sezione sono registrate tutte le vecchie partite giocate e concluse

Pop up di scelta



Plancia di Gioco:

Modalità 5 buchi

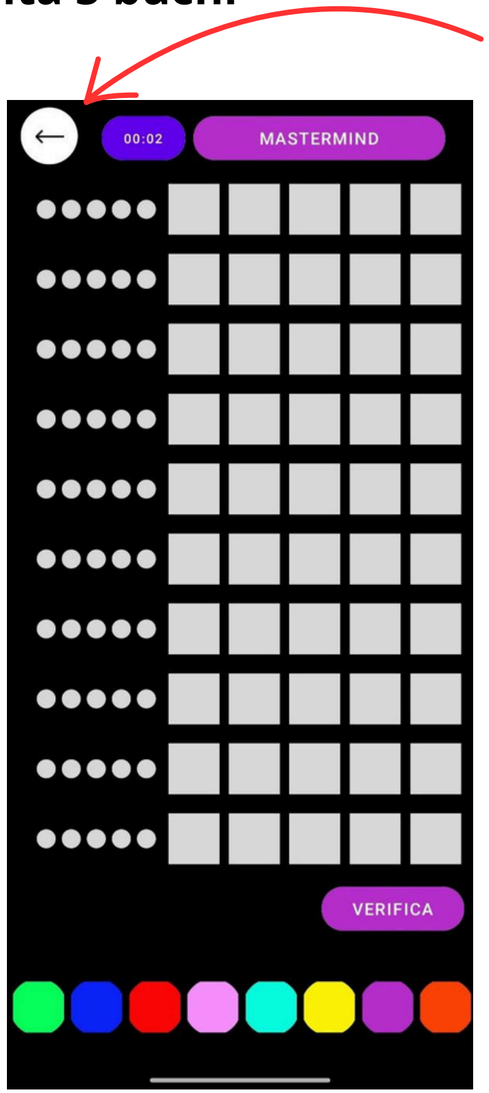



```
LazyColumn(state = listState) {
    items(numberOfRows) { rowIndex ->
        val rowKey = "Row #$rowIndex"
        LazyRow(
            modifier = Modifier
                .padding(
                    horizontal = 16.dp,
                    vertical = 8.dp
                )
        ) { items(numberOfBoxesPerRow) { index ->
                //Questo è l'identificatore univoco di ogni box
                val boxId = "Box #${(rowIndex * numberOfBoxesPerRow) + index}"
                //Questo è il colore di ogni box, all'inizio è default grigio
                var boxColor by remember { mutableStateOf(Color( color: 0xffd9d9d9)) }
                boxColor = boxColors[boxId] ?: Color( color: 0xffd9d9d9)
                Box(
                    modifier = Modifier
                        .padding(end = if (index < numberOfBoxesPerRow - 1) spacingBetweenBoxes else 0.dp)
                        .requiredWidth(width = 43.dp)
                        .requiredHeight(height = 43.dp)
                        .background(boxColor)
                        .clickable {
                            val boxPosition = (rowIndex * numberOfBoxesPerRow) + index
                            val startRange = row.value * numberOfBoxesPerRow
                            val endRange =
                                row.value * numberOfBoxesPerRow + numberOfBoxesPerRow

                            if (boxPosition in startRange ≤.. ≤ endRange) {
                                specialColorChange(boxId)
```

Plancia di Gioco:

Modalità 5 buchi



Tasto per andare indietro, sia questo bottone programmato che il tasto di sistema mostreranno a schermo un Alert per non permettere all'utente di uscire dalla partita senza essere cosciente che se la partita non è finita essa non verrà salvata

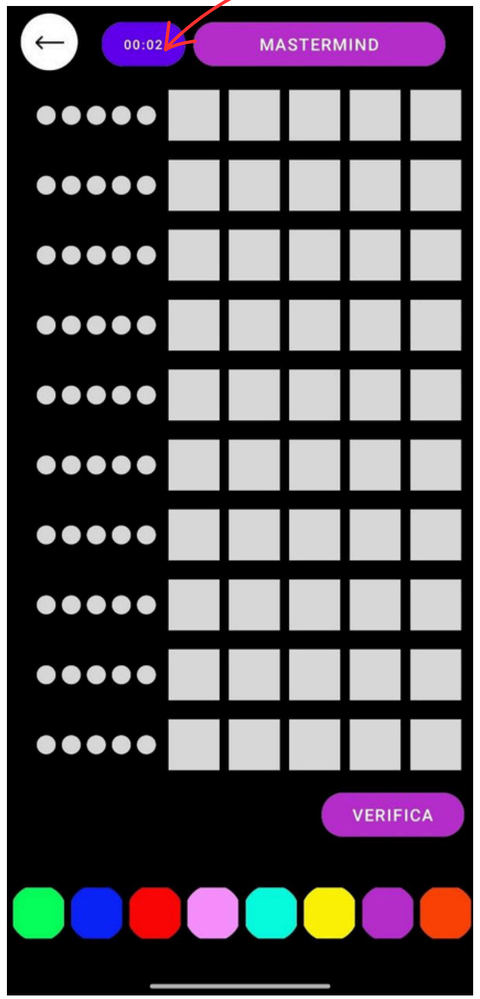
```
override fun onBackPressed() {
    val message = getString(R.string.exit_dialog)
    val yes = getString(R.string.yes)
    val no = getString(R.string.no)

    AlertDialog.Builder(this)
        .setMessage(message)
        .setPositiveButton(yes) { _, _ ->
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
            finish()
        }
        .setNegativeButton(no) { _, _ -> }
        .show()
}
```

Plancia di Gioco:

Modalità 5 buchi

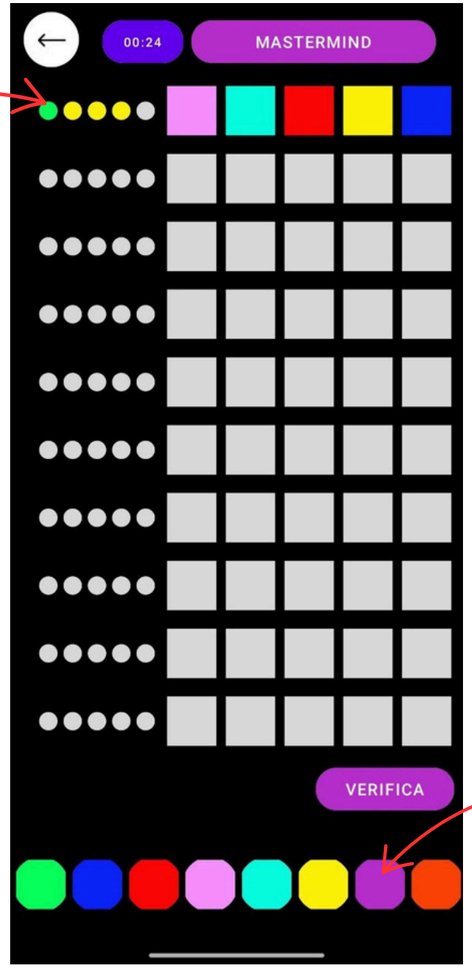
Tempo trascorso, se cliccato la partita va in pausa




```
@Composable
fun TimerScreen() {
    var minutes by remember { mutableStateOf( value: 0) }
    var seconds by remember { mutableStateOf( value: 0) }
    val lifecycleOwner = LocalLifecycleOwner.current
    val observer = rememberUpdatedState(LifecycleEventObserver { _, event ->
        when (event) {
            Lifecycle.Event.ON_PAUSE -> isPaused.value = true
            Lifecycle.Event.ON_RESUME -> isPaused.value = false
            else -> Unit } })
    DisposableEffect(lifecycleOwner) { this: DisposableEffectScope
        val lifecycle = lifecycleOwner.lifecycle
        lifecycle.addObserver(observer.value)
        onDispose {
            lifecycle.removeObserver(observer.value) } ^DisposableEffect }
    LaunchedEffect(isPaused.value) { this: CoroutineScope
        snapshotFlow { isPaused.value}
            .collect { paused ->
                if (!isPaused.value) {
                    while (true) {
                        delay( timeMillis: 1000)
                        seconds++
                        if (seconds >= 60) {
                            minutes++
                            seconds = 0 }
                        timerState.value = String.format("%02d:%02d", minutes, seconds) } } } }
```

Plancia di Gioco:

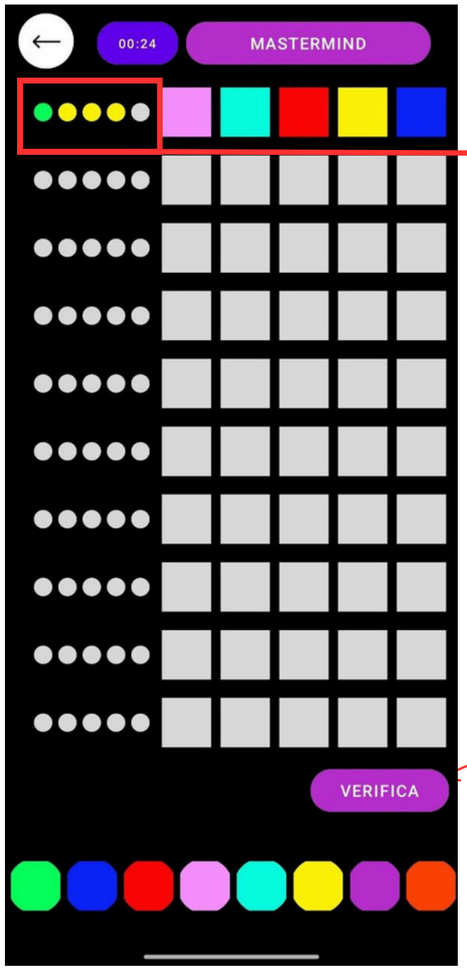
Quando si verifica una linea verranno mostrati qui i suggerimenti, per ogni pallino verde corrisponde un colore giusto e posizione giusta, per ogni pallino giallo solo il colore è giusto



Se cliccati questi bottoni andranno ad inserire il colore nel quadrato al momento libero, nella linea di gioco

Plancia di Gioco:

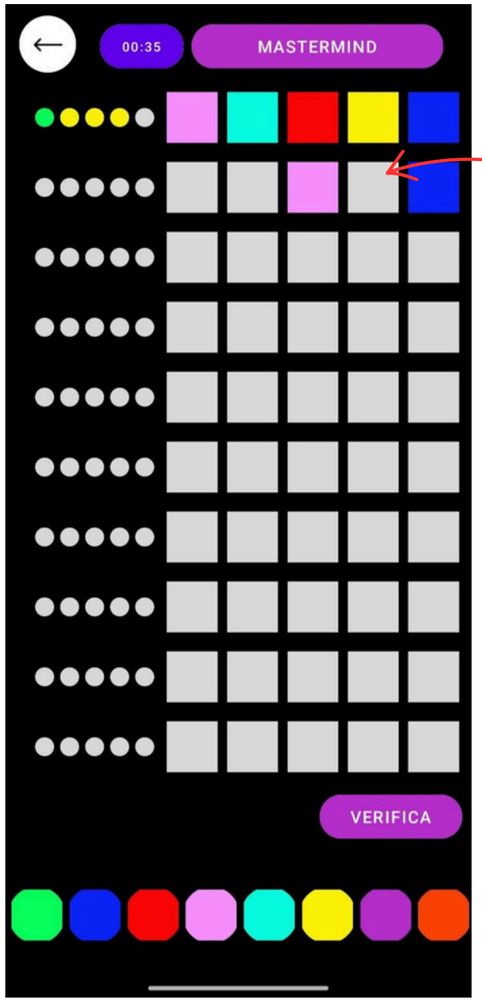
Modalità 5 buchi



Bottone di verifica linea

```
private fun checkMastermind() {
    if (box.value != 0 && box.value % numberOfBoxesPerRow == 0 && checkLine.value == false) {
        checkLine.value = true
        var mySet: MutableList<String> = mutableListOf()
        for (element in randomColors) {
            mySet.add(getColorHexFromName(element.second))}
        for (i in 1 ≤ until < numberOfBoxesPerRow +1) {
            val boxKey = "Box #${box.value - i}"
            val randomColorHex = getColorHexFromName(randomColors[numberOfBoxesPerRow - i].second)
            if (boxColors[boxKey]?.let { colorToHex(it) } == randomColorHex) {
                mySet.remove(boxColors[boxKey]?.let { colorToHex(it) })
                checkColors[appoggioCheckColors.value] = Color( color: 0xFF07FF5C)
                appoggioCheckColors.value += 1
                // Verifica se tutti i box sono verdi
                if (appoggioCheckColors.value % numberOfBoxesPerRow == 0 && appoggioCheckColors.value != 0) {
                    _allBoxesAreGreen.value = true}}}
        for (i in 1 ≤ until < numberOfBoxesPerRow +1) {
            val boxKey = "Box #${box.value - i}"
            val randomColorHex = getColorHexFromName(randomColors[numberOfBoxesPerRow - i].second)
            if (box.value - i >= 0 && boxColors[boxKey]?.let { colorToHex(it) } != randomColorHex && boxColors[boxKey]?.let { it: Color
                    colorToHex(it)
                } in mySet) {
                mySet.remove(boxColors[boxKey]?.let { colorToHex(it) })
                checkColors[appoggioCheckColors.value] = Color( color: 0xFFFBF207)
                appoggioCheckColors.value += 1}}
        row.value += 1 }
    //Qui si arrotonda appoggio per farlo passare alla riga successiva(QUI C ERA IL BUG DI GIADA DEL 02/06
    if (appoggioCheckColors.value != row.value * numberOfBoxesPerRow) {
        appoggioCheckColors.value += numberOfBoxesPerRow - appoggioCheckColors.value % numberOfBoxesPerRow
        println("Ho aggiunto ${appoggioCheckColors.value}")
```

Plancia di Gioco:



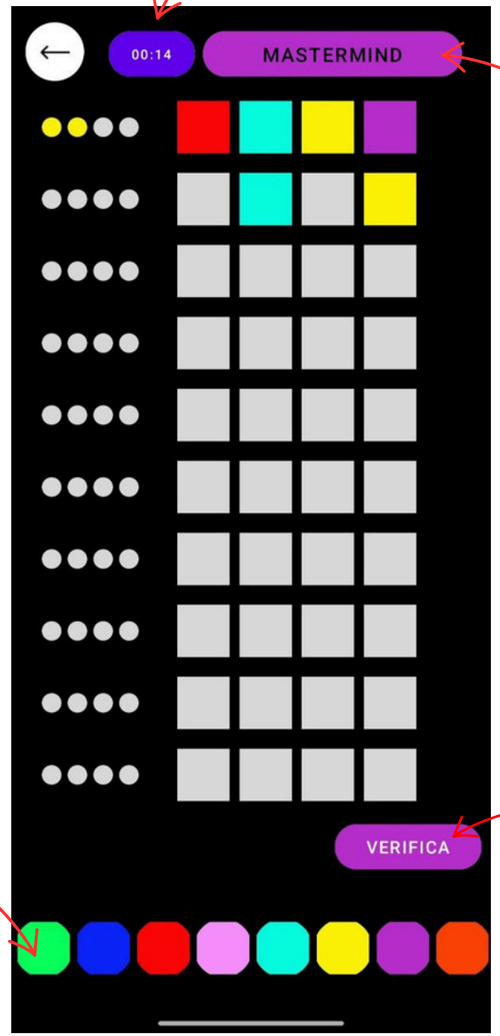
Se durante l'inserimento si sbaglia e si vuole cambiare il colore inserito si può cliccare il colore che si vuole modificare e inserirlo nuovamente

```
private fun specialColorChange(s: String) {
    checkLine.value = true
    if (boxColors[s] != Color( color: 0xffd9d9d9)) {
        boxColors[s] = Color( color: 0xffd9d9d9)
        if (s.substringAfter( delimiter: "#").toIntOrNull()!! < box.value) {
            if (box.value != row.value * numberOfBoxesPerRow && row.value < 2) {
                box.value = s[s.length - 1].digitToInt()
            } else if (box.value != row.value * numberOfBoxesPerRow && box.value != row.value * numberOfBoxesPerRow + numberOfBoxesPerRow && row.value >= 2 && (box.value
                if(s.substringAfter( delimiter: "#").toIntOrNull()!! < 10) {
                    box.value = row.value * numberOfBoxesPerRow + s[s.length - 1].digitToInt() % numberOfBoxesPerRow }
                else {
                    box.value = row.value * numberOfBoxesPerRow + (s[s.length - 2].digitToInt()*10 + s[s.length - 1].digitToInt()) % numberOfBoxesPerRow } }
            else if (box.value != row.value * numberOfBoxesPerRow && box.value != row.value * numberOfBoxesPerRow && row.value >= 2 && (box.value / numberOfBoxesPerRow) %
                println("3.sono entrato per boxvalue ${box.value}")
                if(numberOfBoxesPerRow == 5)
                    box.value = row.value * numberOfBoxesPerRow + s[s.length - 1].digitToInt()
                else if(numberOfBoxesPerRow == 4)
                    if(s.substringAfter( delimiter: "#").toIntOrNull()!! < 10) {
                        box.value = row.value * numberOfBoxesPerRow + s[s.length - 1].digitToInt() % numberOfBoxesPerRow }
                    else box.value = row.value * numberOfBoxesPerRow + (s[s.length - 2].digitToInt()*10 + s[s.length - 1].digitToInt()) % numberOfBoxesPerRow
            } else if (box.value != row.value * numberOfBoxesPerRow && row.value >= 2 && (box.value / numberOfBoxesPerRow) % 2 == 0) {
                if(numberOfBoxesPerRow == 5)
                    box.value = row.value * numberOfBoxesPerRow + s[s.length - 1].digitToInt() % numberOfBoxesPerRow
                else if(numberOfBoxesPerRow == 4)
                    if(s.substringAfter( delimiter: "#").toIntOrNull()!! < 10) {
                        box.value = row.value * numberOfBoxesPerRow + s[s.length - 1].digitToInt() % numberOfBoxesPerRow }
                    else box.value = row.value * numberOfBoxesPerRow + (s[s.length - 2].digitToInt() * 10 + s[s.length - 1].digitToInt()) % numberOfBoxesPerRow
```

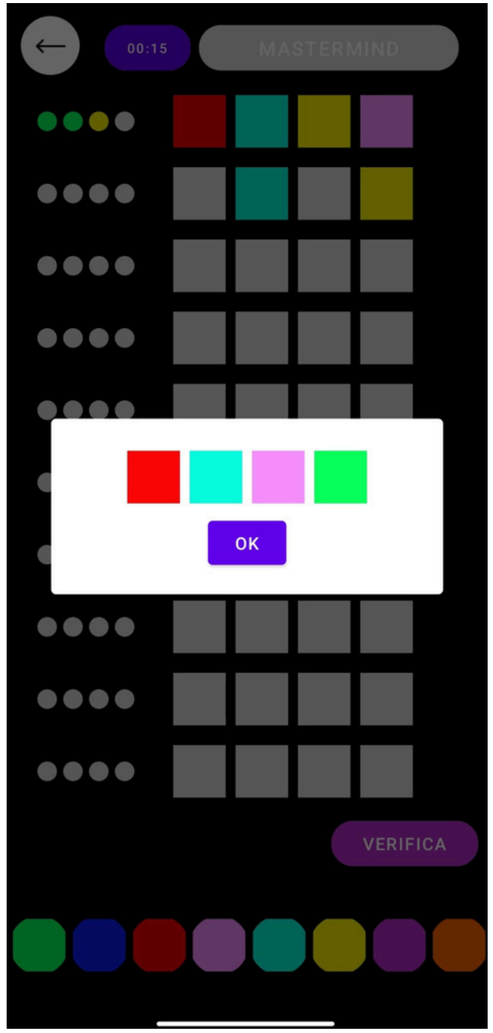
Plancia di Gioco:

Modalità 4 buchi

Tempo trascorso, se cliccato la partita va in pausa



Premendo questo bottone è sempre possibile vedere il codice generato dal computer



Se cliccati questi bottoni andranno ad inserire il colore nel quadrato al momento libero, nella linea di gioco

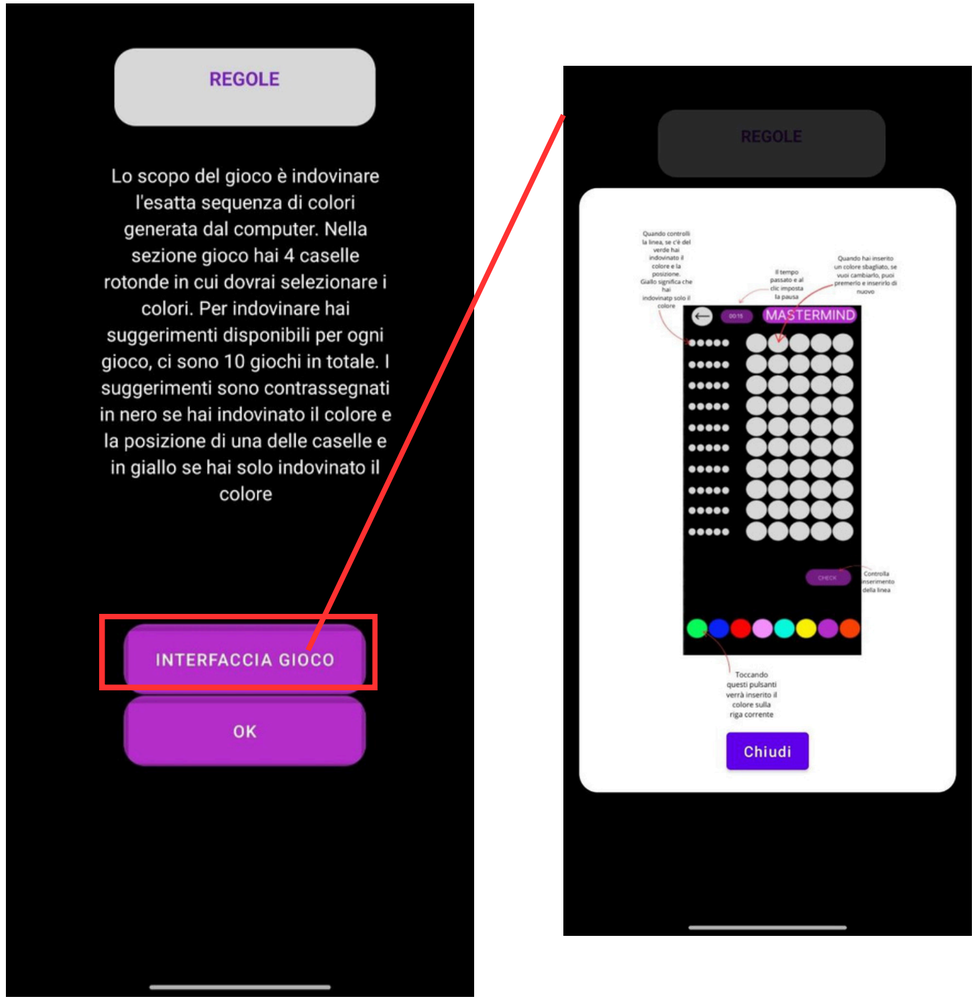
Bottone di verifica linea

Storico delle partite:

Qui vengono salvate tutte le partite giocate tramite un id unico per ogni partita, la configurazione della giocata, il risultato booleano visibile anche grazie al colore del box, quanti tentativi sono stati impiegati e il tempo

Regole:

In questa sezione vengono brevemente descritte le regole generali del gioco, e premendo su “Interfaccia Gioco” si può visualizzare la plancia di gioco e la spiegazione di ogni sua parte




```
if (showPopup) {
    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(color = Color.Black.copy(alpha = 0.8f))
            //.clickable { showPopup = false }
    ) { this: BoxScope
        Box(
            modifier = Modifier
                .align(Alignment.Center)
                .clip(RoundedCornerShape(18.dp))
                .background(Color.White)
                .padding(16.dp)
        ) { this: BoxScope
            Column(horizontalAlignment = Alignment.CenterHorizontally) { this: ColumnScope

                Image(
                    painter = painterResource(id = R.drawable.spiega_regole),
                    contentDescription = null,
                    modifier = Modifier
                        .height(500.dp)
                        .width(330.dp)
                )
                Spacer(modifier = Modifier.height(1.dp))
                Button(
                    onClick = { showPopup = false },
                ) { this: RowScope
                    Text(stringResource(id = "Close"))
                }
```