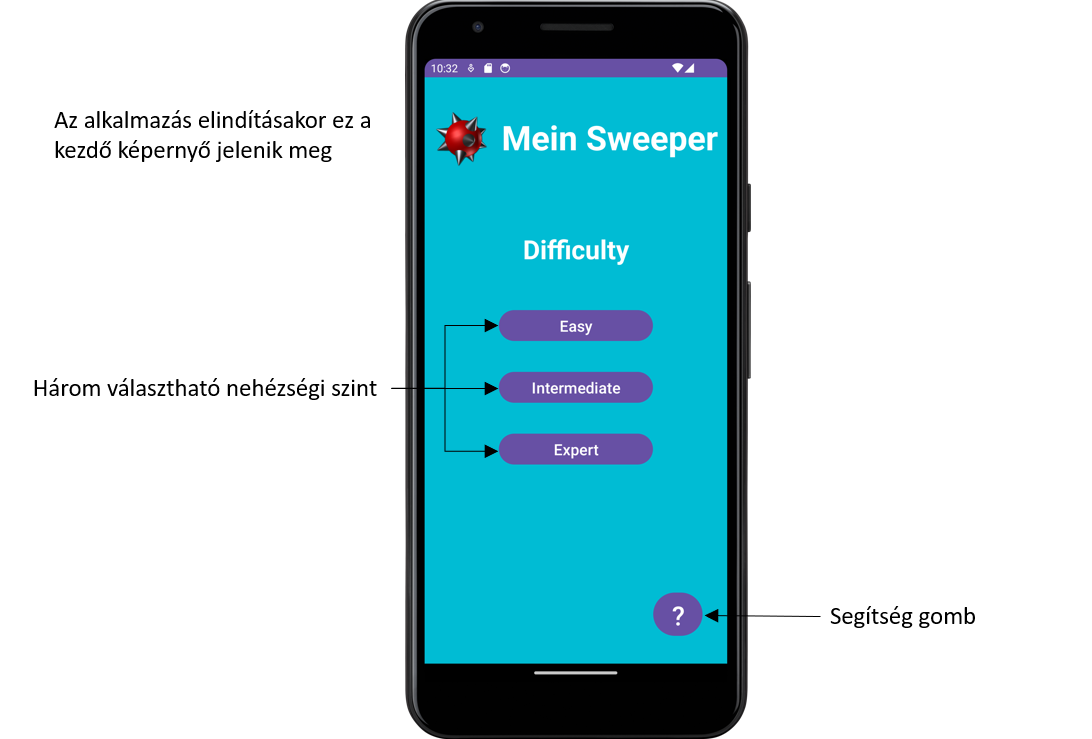
Aknakereső Játék Android Alkalmazás

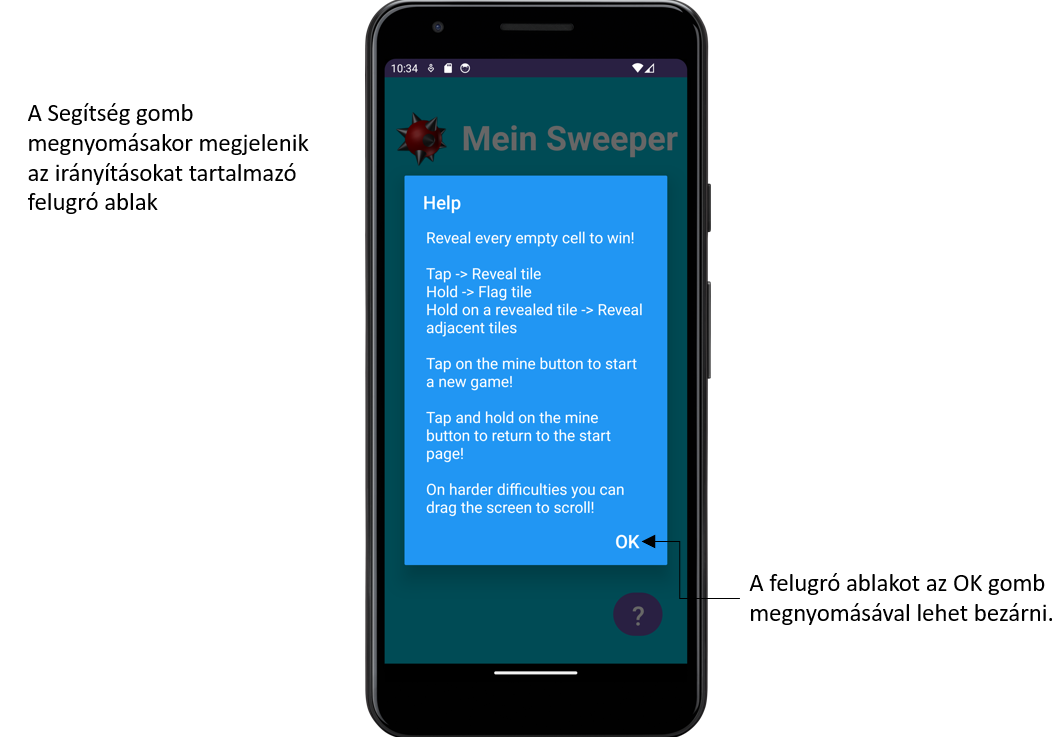
Mészáros Ákos

BFNA2X

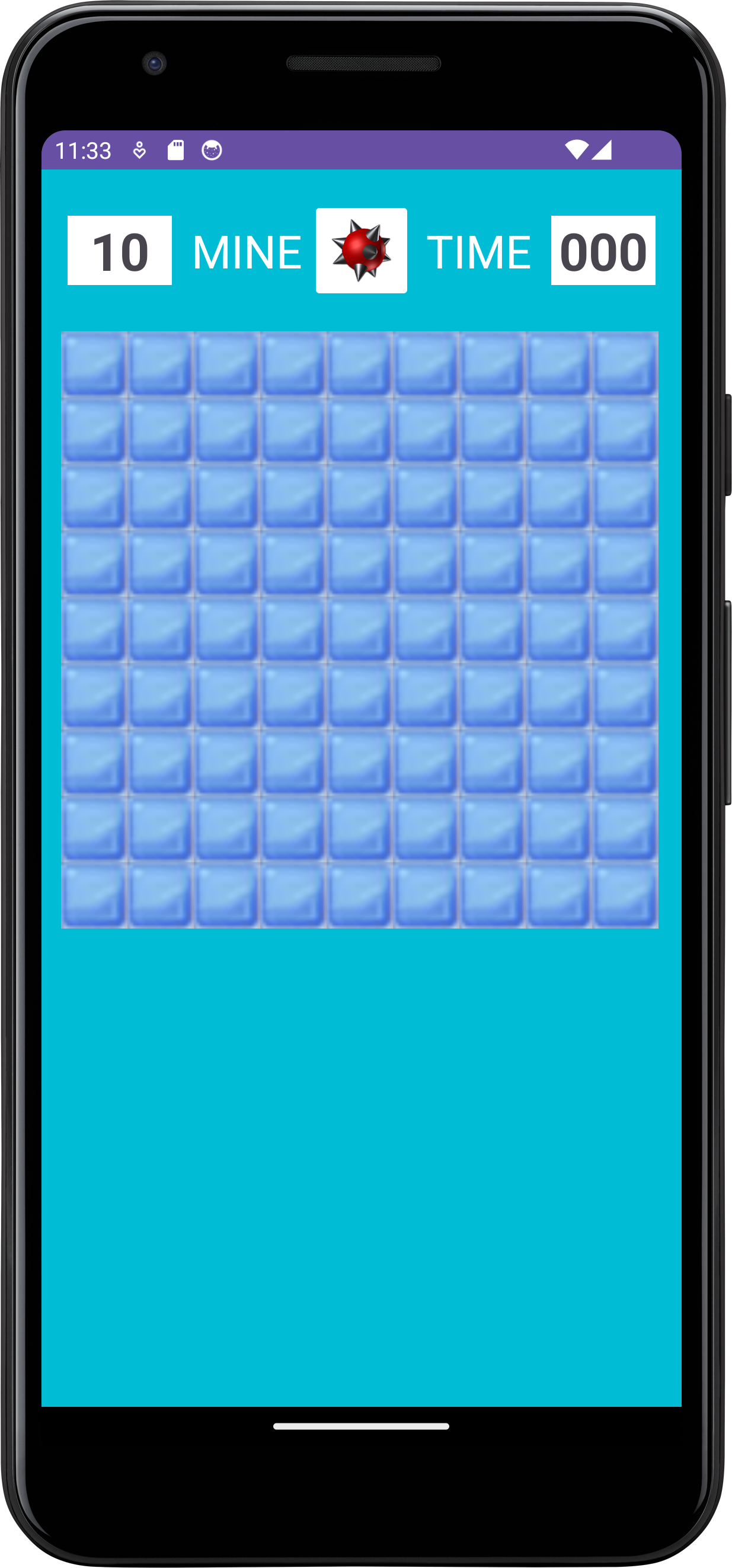
# Kezdőképernyő



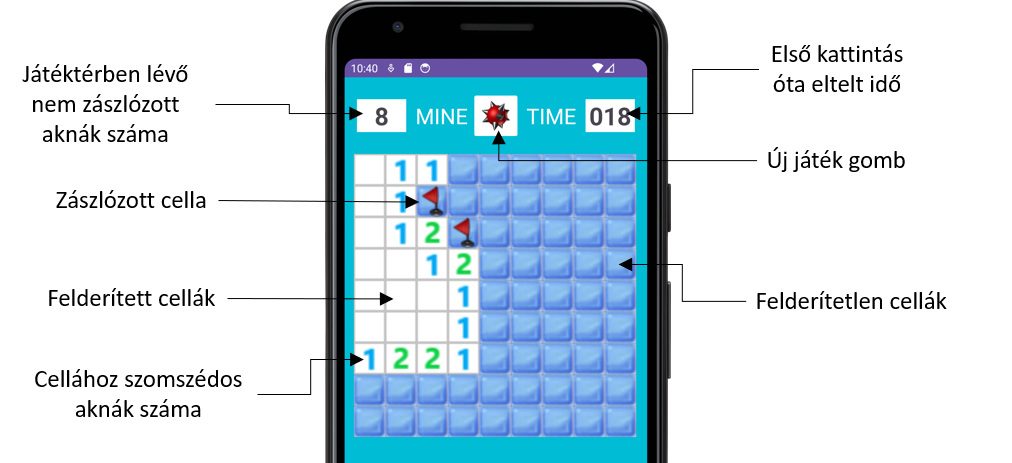
A kezdőképernyőn a nehézségi szint gombok egyikének megnyomásával lehet elindítani a játékot, valamint a segítség gomb megnyomásával előhozhatjuk az irányításokat tartalmazó felugró ablakot.



# Játéktér



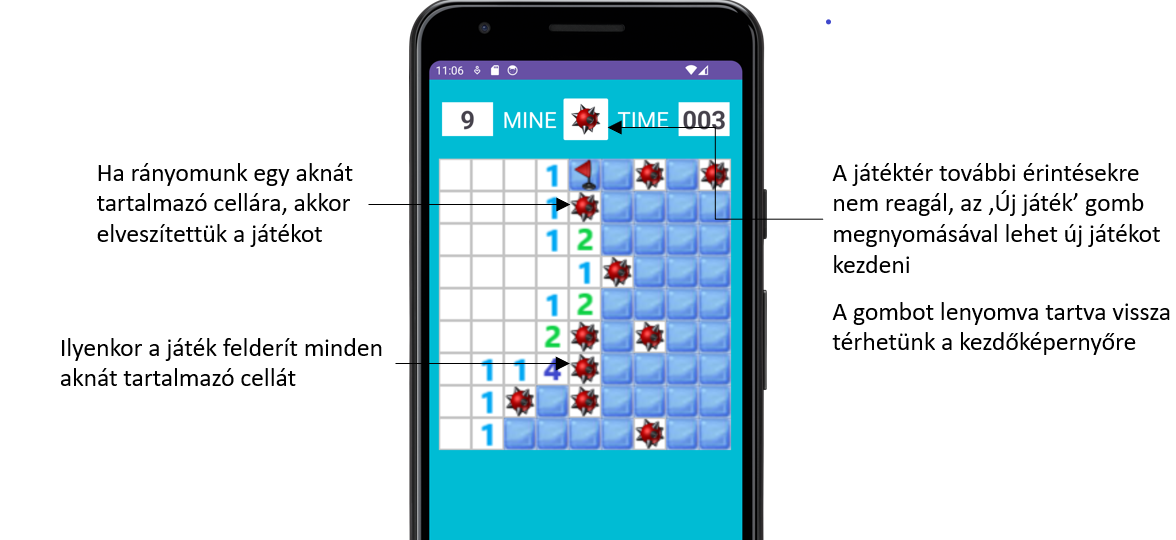
A játék elindításakor egy felderítetlen cellákat tartalmazó rács jelenik meg. Bármelyik cellára rányomva megkezdhetjük a játékot. (Az első kattintással sosem találhatunk aknára.)



A játéktérben végrehajtható akciók listája:

* Felderítetlen cellára nyomva az adott cellát felderítjük
* Felderítetlen cellára hosszú lenyomással zászlót rakhatunk (vagy levehetünk róla) amivel megjelöljük, hogy szerintünk aknát tartalmaz.
* Zászlót tartalmazó cellát nem derítünk fel, ha azt megnyomjuk
* Ha a cella amit felderítettünk egy szomszédja sem tartalmaz aknát, akkor az összes szomszédos cellát autómatikusan felderítjük
* Számmal (szomszédos aknákkal) jelzett cellák szomszédos celláit is fel lehet fedezni hosszú lenyomással, ha kellő mennyiségű szomszédos cellát zászlóztunk
* Új játék gomb megnyomásával a játékot újra kezdhetjük
* Az új játék gombot hosszan lenyomva vissza térhetünk a kezdőképernyőre







# Kódrészletek

## Játék indítása

fun newGameEasy(view: View) {  
 val intent = Intent(this, MainActivity::class.*java*)  
 intent.putExtra("gameSettings",easyGame) //{9,9,10}}  
  
 startActivity(intent)  
}  
  
fun newGameMed(view: View) {  
 val intent = Intent(this, MainActivity::class.*java*)  
 intent.putExtra("gameSettings",medGame) //{16,16,40}  
  
 startActivity(intent)  
}  
fun newGameExpert(view: View) {  
 val intent = Intent(this, MainActivity::class.*java*)  
 intent.putExtra("gameSettings", expertGame) //{16,30,99}  
  
 startActivity(intent)  
}

Új játék kezdetekor Intent segítségével adjuk át a játéktérnek a kiválasztott nehézség beállításait.

## Játéktér inicializásál

override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 val settings = *intent*.getIntArrayExtra("gameSettings")!!  
  
 gm = GameManager(settings[0],settings[1],settings[2])

A kapott Intent-ből kicsomagoljuk a játék beállításait és létrehozzuk a játékvezérlőt.

grid = findViewById(R.id.*grid*)  
 mineCounter = findViewById(R.id.*mineCounter*)  
 timer = findViewById(R.id.*timer*)

Felületi elemek betöltése változókba.

updateTimer = *Runnable* **{** runOnUiThread **{** gameTime++  
 timer.*text* = String.*format*("%03d", gameTime)  
 **}  
 }** val button = findViewById<ImageButton>(R.id.*startButton*)  
 button.setOnClickListener**{** newGame()  
 **}** button.setOnLongClickListener **{** finish() //close activity, return to title screen  
 true  
 **}**

Időzítő és gombfunkciók hozzárendelése.

newGame()  
}

## Játék indítása

private fun newGame(){  
  
 //regenerate game board  
 gm.startGame()

A játékteret a játékvezérlő startGame() funkciója hozza létre

fun startGame(){  
 remainingCells.clear()  
 mineDisplayCount = mineCount  
 isFirstRound = true  
  
 for (row in 0 *until* rows) {  
 for(col in 0 *until* cols){  
 gameGrid[row][col] = Cell(row,col)  
 remainingCells.add(gameGrid[row][col]!!)  
 }  
 }  
  
 val rand = Random  
 for (i in 0 *until* mineCount){  
 placeMine(rand)  
 }  
   
 calcAdjacentMines()  
}

Az aknákat garantált elhelyezése érdekében az alábbi függvényt használjuk:

private fun placeMine(rand: Random) {  
 while (true) {  
 val row = rand.nextInt(rows)  
 val col = rand.nextInt(cols)  
  
 if (!gameGrid[row][col]!!.hasMine){  
 gameGrid[row][col]!!.hasMine = true  
 remainingCells.*remove*(gameGrid[row][col])  
 return  
 }  
 }

A calcAdjacentMines funkció védidmegy minden aknán, és megszámolja hány szomszédos cellája tartalmaz aknát

private fun calcAdjacentMines(){  
 gameGrid.*flatten*().*forEach* **{** val cell = **it**!!  
  
 if (cell.hasMine) {  
 return  
 }  
 val row = cell.row  
 val col = cell.col  
 var value = 0  
  
 for (i in row - 1..row + 1) {  
 for (j in col - 1..col + 1)  
 if (i in 0 *until* rows && j in 0 *until* cols) {  
 if (gameGrid[i][j]!!.hasMine) {  
 value++  
 }  
 }  
 }  
  
 cell.adjacentMines = value  
 **}**}

Ezt követően vissza térünk a MainActivity newGame() funkciójába

//cleanup previous game  
if(gameReset) {  
 grid.removeAllViews()  
  
 imageArray.*forEach* **{** row **->** //clear image array from previous game  
 row.*forEach* **{** imageView **->** imageView.setOnClickListener(null)  
 imageView.setOnLongClickListener(null)  
 imageView.setImageDrawable(null)  
 (imageView.*parent* as? ViewGroup)?.removeView(imageView)  
 **}  
 }** System.gc()  
}

Ez a szakasz első futáskor nem hajtódik végre, korábbi játékok cellaikonjait töröli teljesen, memory leak elkerülése érdekében.

//reset grid size and counters  
grid.*rowCount* = gm.rows  
grid.*columnCount* = gm.cols  
mineCounter.*text* = "${gm.mineDisplayCount}"  
timer.*text* = getString(R.string.*zeroCount*)  
gameTime = -1

A GridLayoutot a játéktérnek megfelelő méretűre állítjuk, és vissza állítjuk az időzítőt az alapértelmezett értékre.

//create image grid  
imageArray = Array(gm.rows) **{** Array(gm.cols)**{**ImageView(this)**} }**//populate image grid  
for(row in 0 *until* gm.rows) {  
 for (col in 0 *until* gm.cols) {  
 imageArray[row][col].setImageResource(R.drawable.*button*) //set image  
 imageArray[row][col].*layoutParams* = ViewGroup.LayoutParams(112, 112)  
  
 imageArray[row][col].setOnClickListener**{**//short tap  
 revealCell(row,col)  
 **}** imageArray[row][col].setOnLongClickListener**{**//tap and hold  
 flagCell(row,col)  
 true  
 **}** //add each ImageView to the display grid  
 grid.addView(imageArray[row][col])  
 }  
}

Létrehozzuk a cellák kirajzolására szolgáló ImageView-eket, majd hozzá rendeljük a kattintás és hosszú lenyomás funkciókat.

//set resetGame flag after first init  
 gameReset = true  
}

Jelezzük, hogy lefutott az első játékindítás, ezt követően futni fog a takarító szakasz.

## Cellák felfedése

private fun revealCell(row: Int, col: Int){  
  
 val thisCell = gm.gameGrid[row][col]!!  
  
 if(thisCell.isExposed)  
 return  
  
 if(thisCell.isFlagged)  
 return

Ha a cella már fel van fedve, vagy zászlózott, nem fogjuk felfedni.

if(gm.isFirstRound){  
 if(thisCell.hasMine)  
 gm.firstClickMine(thisCell)  
  
 executor.scheduleAtFixedRate(updateTimer, 0, 1, TimeUnit.*SECONDS*)  
 gm.isFirstRound = false  
}

Első kattintásnál, amennyiben aknára kattintottunk, áthelyezzünk más pozícióba az aknát, valamint elindítjuk az időzítőt.

if(thisCell.hasMine){  
 imageArray[row][col].setImageResource(R.drawable.*mine*)  
 endGame(false)  
 return  
}  
  
imageArray[row][col].setImageResource(cellImageArray[thisCell.adjacentMines])  
thisCell.isExposed = true  
gm.remainingCells.remove(thisCell)

Frissítjük a cella ikonját aknával, vagy a megfelelő számértékkel. Amennyiben cellára kattintottunk befejezzük a játékot.

if(gm.remainingCells.isEmpty()){  
 endGame(true)  
}

Ha ez volt az utolsó üres cella, befejezzük a játékot.

if(thisCell.adjacentMines == 0)  
 revealAdjacent(row,col)  
}

Ha a cella egy szomszédja sem tartalmaz aknát, felfedjük azokat is.

## Cella zászlózása (hosszú lenyomás)

private fun flagCell(row: Int, col: Int) {  
 val thisCell : Cell = gm.gameGrid[row][col]!!  
 if(thisCell.isExposed){  
 if(gm.allMinesFlagged(thisCell))  
 revealAdjacent(row,col)  
 return  
 }

Ha a cella már fel van fedve, nem zászlózzuk. Számot tartalmazó cella esetén felfedjük a szomszédokat, ha a helyes számú szomszédos cella zászlózva van.

thisCell.isFlagged = !thisCell.isFlagged //flip flag value  
  
 when(thisCell.isFlagged){ //redraw cell based on new flag state  
 true -> {  
 gm.mineDisplayCount--  
 imageArray[row][col].setImageResource(R.drawable.*flag*)  
 }  
 false -> {  
 gm.mineDisplayCount++  
 imageArray[row][col].setImageResource(R.drawable.*button*)  
 }  
 }  
 mineCounter.*text* = "${gm.mineDisplayCount}" //update remaining mines display  
}

Átváltjuk a zászló állapotát, frissítjük a cella ikonját és az akna számlálót.

## Szomszédos aknák felfedése

private fun revealAdjacent(row: Int, col: Int) {  
  
 for(i in row-1..row+1)  
 if(i in 0 *until* gm.rows){  
 for(j in col-1 .. col+1)  
 if(j in 0 *until* gm.cols){

Ellenőrzés, hogy a cella a játékteren belül helyezkedik-e el (NullPointerException elkerülése)

val thisCell = gm.gameGrid[i][j]!!  
 if(gm.remainingCells.contains(thisCell) || (!thisCell.isFlagged && thisCell.hasMine))  
 revealCell(i,j)  
 }  
 }  
}

Csak akkor fedjük fel a cellát, ha még nem volt felfedve. (infinite loop elkerülése)