Group 2
Ronas Baran, Hector Gonzalez Campos, Emil Ståhl

# Integrated Prototype

A research project in collaboration with KTH and Stockholms University

## 1. Introduction

This week the focus was on delivering the Android prototype. In order to do this we needed to implement the given cognitive task specified by the research team at Stockholm University. The cognitive task consists of a visuospatial memory task measuring working memory performance.

## 2. Visuospatial Memory Task

The test consists of encoding, distraction and recall phases.

**Encoding phase** (10 second duration): 9 symbols are displayed in a 3x3 grid, the symbols are pseudo randomly chosen and displayed from a set of icons with the limitation that none of the symbols was presented during the past 5 trials.

**Distraction phase** (8 second duration): The participant has to touch all F's as fast as possible on the 5x9 grid, distribution 30% F's, 70%. When a F is pressed it disappears from the grid.

**Recall phase** (15 second duration): The participant has to touch the location of the symbol shown on top. The symbol shown is selected randomly from the previous set of 9 displayed symbols. If the participant does not respond within 15 seconds, a beeping tone is presented to alert the participant for 1 second and the test ends.

# 3. Implementation

Each phase was implemented in its own file, named Encoding.kt, Distraction,kt and Recall.kt respectively.

## 3.1 Encoding

Encoding.kt contains code for displaying the encoding activity and setting the timeout. Something not shown in the code below is the mutable list of the drawable symbols, this list is later used for getting a random symbol to display on the grid. To do this a random number is generated, the imageSource is set and the symbol is removed from the list to prevent copies on the grid.

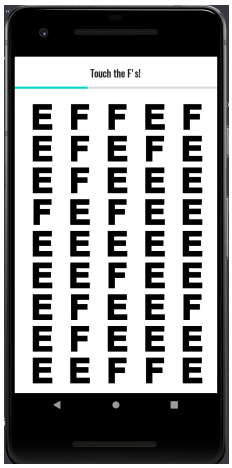

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    getWindow().setFlags(
        WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN
    );
    setContentView(R.layout.activity_encoding)

    //----------TIMER----------
    //changes activity after 10 seconds
    val pb: ProgressBar = findViewById(R.id.progressBar)
    val animation = ObjectAnimator.ofInt(pb, "progress", 0, 100)
    animation.duration = 10000
    animation.interpolator = DecelerateInterpolator()
    animation.addListener(object : Animator.AnimatorListener {
        override fun onAnimationStart(animator: Animator) {}

        //changes activity after 10 seconds
        override fun onAnimationEnd(animator: Animator) {
            val i = Intent(this@Encoding, Distraction::class.java)
            startActivity(i)
            finish()
        }

        override fun onAnimationCancel(animator: Animator) {}
        override fun onAnimationRepeat(animator: Animator) {}
    })
    animation.start()
```

## 3.2 Distraction

The file for implementing the distraction phase looks pretty much the same. The list however only consists of two symbols, an E and a F. In order to generate a distribution of ~70% E's the following code snippet was used:

```
private fun rollDie(): Any {
    val rand = Random()
    var roll = rand.nextInt(100);
    return roll < 70;
}
```

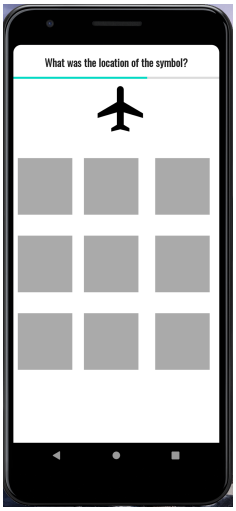
This does not give the exact same distribution on each run but it fluctuates a bit. This can be interpreted as both positive and negative since it does not result in a recurring number of F's which the user can use for completing the test in an easier way. At the same time it might make it harder to evaluate the performance on this step since the number of F's needs to be considered.




### 3.3 Recall

The file for implementing the recall phase looks pretty much the same as the Encoding. The main difference is the code snippet for determining the symbol to guess, which looks like:

```
val  guessImageView : ImageView =  findViewById(R.id.symbolToGuess)
val guessImage = images[usedSymbols1?.random()!!]
guessImageView.setImageResource(guessImage)
for ((counter, i) in symbols.withIndex()){
   if(guessImage==images[usedSymbols1!!?.get(counter)]) {
        success = true
        i?.setOnClickListener {
            i?.setImageResource(images[usedSymbols1!!?.get(counter)])
        }
```



## 4. User feedback

During development of the cognitive task external user feedback was documented and taken into consideration while developing new features. Due to the current situation a video of the app was recorded and sent out to users for feedback. The video is available here: https://youtu.be/cinVmjFHJto

**Documented feedback:**

**User 1:** Looks good but the information icon is bigger than the others. Nice color of the background. When the user has guessed the correct location of the symbol in the recall phase maybe the test can finish instead of the user having to wait for it to end:

**User 2:** The cognitive test manages to be efficient and become a challenge, and it is a medium-difficult level. The layout is basic but it works perfectly. The symbols of the encoding phase are in a different position regarding the recall phase.

**User 3:** I think the design and ease of use is good because it's clear what it does and not complicated to use.

**User 4:** It looks cheap, feels like something a freshman in my High School could have come up with.

**User 5:** It's weird that the cognitive test just starts without any warning or notification, it's not clear why I should touch the F's more than as a distraction. It's strict timers that sum up to a total of 30 seconds for the cognitive test but it says on the start page that it takes 3 minutes to complete.

**User 6:** I found it to be user friendly. Easy good choice of symbols and the letter test not too complicated.

## 5. GitHub repository

The source code of the integrated Android prototype can be found here:

https://github.com/NellyFriman/SleepApp-AndriodPrototype