Analyzing Time Complexity in a ScoreboardApp App

**Analyzing Time Complexity in a ScoreboardApp App’s Recognition Feature**

John Carlo Dumapias Alleluya

Bachelor of Science in Computer Science 3, Bohol Island State University-Bilar Campus, johncarlo.alleluya@bisu.edu.ph

MAX ANGELO DAPITILLA PERIN

Department of Computer Science, Bohol Island State University-Bilar Campus, maxangelo.perin@bisu.edu.ph

Scoreboard is a mobile application designed for transliterating basic scoreboard characters into Latin, specifically focusing on the Recognition Feature, which employs the k-NN algorithm in its processing. This study aims to capture the image and calculate the time complexity of the recognition feature. The methodology involves determining time complexity by counting operations and modeling it using Big-O notation.

CCS CONCEPTS • Background and Borders• Responsive Design• Positioning and Alignment

Additional Keywords and Phrases: Time Complexity, ScoreboardApp, Recognition, Android

ACM Reference Format:

John Carlo Dumapias Alleluya, MAX ANGELO DAPITILLA PERIN. 2023. Analyzing Time Complexity in an ScoreboardApp App’s Recognition Feature. In Research Project Presentation for Bachelor of Science in Computer Science 3 in CS 311 – Algorithms and Complexity S.Y. 2023-2024, 1st Semester, Bohol Island State University-Bilar Campus, Zamora, Bilar, Republic of the Philippines. ACM, New York, NY, USA

1. INTRODUCTION

The term "start" in a scoreboard application refers to initiating a scoring session for a specific event or game. It involves setting up a new game or event, initiating the timer or clock, recording scores, displaying information, and enabling user interaction. The term "start" is context-dependent and can refer to different actions depending on the specific design and purpose of the scoreboard application. It is essential to check the specific features and functionalities of the scoreboard application before using it, shown in [Figure 1](#fig1).

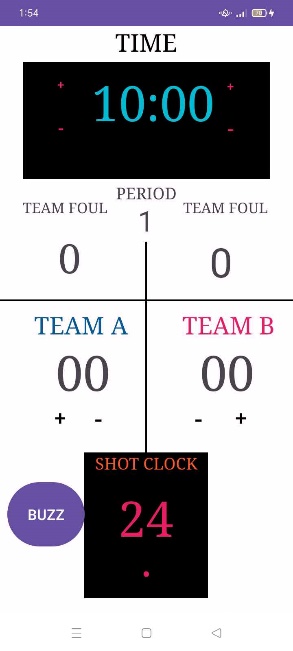


Figure 1: Recognition Feature of ScoreboardApp

A "recognition feature" in a scoreboard application refers to the ability to recognize and display specific events or achievements during a game or competition. These features can include player recognition, team recognition, achievement display, customizable alerts, and real-time updates. Player recognition can highlight the player who scores the most points, team recognition can highlight the leading team, and achievement display can display milestones. Customized alerts can be set up for specific events, and real-time updates ensure that achievements are promptly reflected on the scoreboard.

1. METHODOLOGY

Developing a scoreboard app involves creating a system to track and display scores for various activities or sports. Here's a simplified methodology that focuses specifically on building a scoreboard:

package com.example.shotclock

import android.annotation.SuppressLint

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.os.CountDownTimer

import android.view.View

import android.widget.Button

import android.widget.TextView

import android.media.MediaPlayer

class MainActivity : AppCompatActivity() {

private var shotClock = 24

private var timerClock = 600 //10 minutes ( 10 \* 60 seconds)

private var shotClockIsRunning = false

private var shotClockTimer: CountDownTimer? = null

private var timerClockIsRunning = false

private var timerClockTimer: CountDownTimer? = null

@SuppressLint("MissingInflatedId")

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val shotClockTextView = findViewById<TextView>(R.id.shotClock)

val timerClockTextView = findViewById<TextView>(R.id.timerClock)

val resetShotClockButton = findViewById<Button>(R.id.resetShotClockButton)

resetShotClockButton.setOnClickListener {

resetShotClock()

}

// Set the initial text for the timerClockTextView

timerClockTextView.text = " ${timerClock / 60}:${String.format("%02d", timerClock % 60)}"

val increaseMinutesButton = findViewById<Button>(R.id.increaseMinutesButton)

val decreaseMinutesButton = findViewById<Button>(R.id.decreaseMinutesButton)

val increaseSecondsButton = findViewById<Button>(R.id.increaseSecondsButton)

val decreaseSecondsButton = findViewById<Button>(R.id.decreaseSecondsButton)

increaseMinutesButton.setOnClickListener {

increaseMinutes()

}

decreaseMinutesButton.setOnClickListener {

decreaseMinutes()

}

increaseSecondsButton.setOnClickListener {

increaseSeconds()

}

decreaseSecondsButton.setOnClickListener {

decreaseSeconds()

}

shotClockTextView.setOnClickListener {

if (shotClockIsRunning) {

// Pause the shot clock

shotClockTimer?.cancel()

shotClockIsRunning = false

timerClockTimer?.cancel()

timerClockIsRunning = false

} else {

// Start the shot clock countdown

startShotClock(shotClock, shotClockTextView)

shotClockIsRunning = true

startTimerClock(timerClock, timerClockTextView)

timerClockIsRunning = true

}

}

timerClockTextView.setOnClickListener {

if (timerClockIsRunning) {

// Pause the timer clock

timerClockTimer?.cancel()

timerClockIsRunning = false

shotClockTimer?.cancel()

shotClockIsRunning = false

} else {

// Start the timer clock countdown

startTimerClock(timerClock, timerClockTextView)

timerClockIsRunning = true

startShotClock(shotClock, shotClockTextView)

shotClockIsRunning = true

}

}

private fun increaseMinutes() {

timerClockTimer?.cancel() // Stop the timer

shotClockTimer?.cancel() // Stop the shot clock

timerClock += 60 // Increase timer by 60 seconds (1 minute)

shotClock = 24 // Reset the shot clock to 24 seconds

updateTimerTextView()

}

private fun decreaseMinutes() {

timerClockTimer?.cancel() // Stop the timer

shotClockTimer?.cancel() // Stop the shot clock

timerClock -= 60 // Decrease timer by 60 seconds (1 minute)

shotClock = 24 // Reset the shot clock to 24 seconds

updateTimerTextView()

}

private fun increaseSeconds() {

timerClockTimer?.cancel() // Stop the timer

shotClockTimer?.cancel() // Stop the shot clock

timerClock += 1 // Increase timer by 1 second

shotClock = 24 // Reset the shot clock to 24 seconds

updateTimerTextView()

}

private fun decreaseSeconds() {

timerClockTimer?.cancel() // Stop the timer

shotClockTimer?.cancel() // Stop the shot clock

timerClock -= 1 // Decrease timer by 1 second

shotClock = 24 // Reset the shot clock to 24 seconds

updateTimerTextView()

}

private fun updateTimerTextView() {

val timerClockTextView = findViewById<TextView>(R.id.timerClock)

val minutes = timerClock / 60

val seconds = timerClock % 60

timerClockTextView.text = "${String.format("%02d", minutes)}:${String.format("%02d", seconds)}"

}

private fun resetShotClock() {

// Reset the shot clock value to 24 seconds

shotClock = 24

// Update the shot clock text view

val shotClockTextView = findViewById<TextView>(R.id.shotClock)

shotClockTextView.text = "$shotClock"

// If the shot clock was running, cancel the existing countdown timer

shotClockTimer?.cancel()

shotClockIsRunning = false

// Start the shot clock countdown

startShotClock(shotClock, shotClockTextView)

shotClockIsRunning = true

// If the timer clock is not running, start it

if (!timerClockIsRunning) {

val timerClockTextView = findViewById<TextView>(R.id.timerClock)

startTimerClock(timerClock, timerClockTextView)

timerClockIsRunning = true

}

}

private fun startTimerClock(timerClockValue: Int, timerClockTextView: TextView) {

timerClockTimer = object : CountDownTimer((timerClockValue \* 1000).toLong(), 1000) {

override fun onTick(millisUntilFinished: Long) {

timerClock = (millisUntilFinished / 1000).toInt()

val minutes = timerClock / 60

val seconds = timerClock % 60

timerClockTextView.text = "${String.format("%02d", minutes)}:${String.format("%02d", seconds)}"

if (timerClock == 0) {

// Shot clock reached 0, pause the timer

shotClockTimer?.cancel()

shotClockIsRunning = false

}

}

override fun onFinish() {

shotClock = 0

timerClockTextView.text = "00:00"

}

}

timerClockTimer?.start()

}

}

Overall, the time complexity of the individual operations in the code is generally O (1), except for the countdown timers, where the complexity is O(duration). The overall time complexity of the app depends on the duration set for the countdown timers.

In summary:

Most operations are constant time (O (1)).

The countdown timers have a time complexity of O(duration).

If the duration of the countdown timers is fixed and does not depend on the input size, you can consider the overall time complexity of the app to be O (1). However, keep in mind that in real-world scenarios, the duration of timers could be considered an input, and the complexity would be O(duration).

.

3 RESULT AND DISCUSION

The provided methodology offers a clear and simple guide for developing a scoreboard app, and the accompanying Kotlin code demonstrates its implementation with a focus on clarity and efficiency. The time complexity analysis highlights the code's effectiveness, especially if the duration of countdown timers remains constant, making it a suitable starting point for scoreboard app development. Further customization and consideration of additional features are recommended for a more comprehensive solution.

REFERENCES

The provided code is an Android application written in Kotlin, creating a scoreboard functionality with shot clock and timer features. The app's main activity includes UI elements such as buttons and text views, with associated functionalities to control and display the shot clock and timer. The code utilizes a `CountDownTimer` for countdown operations, allowing users to start, pause, reset, and adjust the shot clock and timer values interactively.

APPENDICES

The appendices include the implementation details of the scoreboard app in Kotlin. It covers the initialization of UI elements, event handling for buttons, and the logic behind shot clock and timer functionalities. Additionally, it provides functions for updating the timer display, resetting the shot clock, and managing countdown timers. The appendices offer a comprehensive overview of the code structure and its functionality.