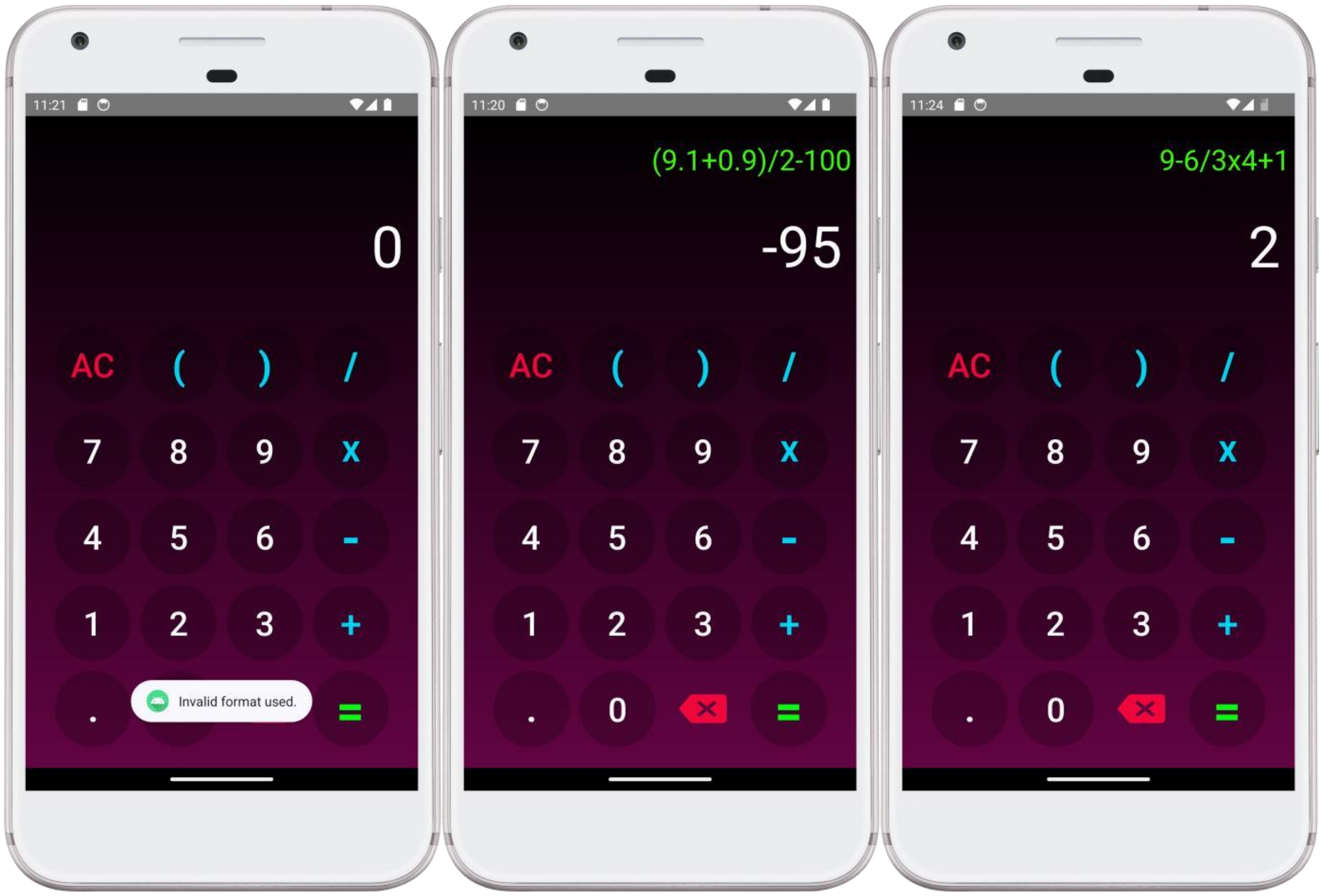
5. Calculator With Chain of Operations

5.1. Overview

The **Calculator app** is designed to provide users with a simple and efficient tool for performing basic arithmetic calculations. The initial idea was to create a calculator that just supports a chain of operations, but I have made it also such that it follows the BODMAS rule.

5.2. User Experience

Efficient and user-friendly, this simple calculator features a clean user interface with easy-to-use buttons for numeric input, basic operations, and more. Moreover, the evaluation engine follows the BODMAS rule. A reliable tool for everyday math needs. Vector assets and gradient backgrounds (Evangelidis, 2020) were used to design the interface.



5.3. Back-End Development

The code structure consists of one MainActivity.

5.3.1 Overview

The **MainActivity** is responsible for all evaluations, input handling, displaying warnings on wrong inputs and many more. The calculation uses a stack-based approach to evaluate expressions.

5.3.2 Evaluation Engine

1. **evaluate_string() method:**
   - The method uses two stacks: one for operands (**operandStack**) and another for operators (**operatorStack**). The algorithm used is Shunting Yard Algorithm (Brilliant.org, n.d.).
   - It iterates through each character of the input expression (**exp**) using a while loop.
   - If the character is a digit or a decimal point, it builds a number and pushes it onto the operand stack.
   - For operators (+, -, x, /), it handles unary and binary operators, considering precedence rules. It uses a helper method **hasPrecedence()** to check the precedence of operators.
   - It handles parentheses by pushing them onto the operator stack and ensuring correct evaluation order.
   - After processing all characters, it performs remaining operations on the stacks until the result is obtained.

2. **hasPrecedence() method:**
   - Determines if the precedence of **op1** is higher than, equal to, or lower than **op2**.
   - Handles different cases for addition/subtraction and multiplication/division.

3. **applyOperator() method:**
   - Takes an operator and two operands from the stack, performs the corresponding operation, and pushes the result back onto the operand stack.
   - Includes checks to handle division by zero and unknown operators.

4. **Additional Functions (CheckLastOp, getLastChar, etc.):**
   - **CheckLastOp()**: Checks if the last character entered was an operator based on specific conditions.
   - **getLastChar()**: Retrieves the last character from a given string.
   - **setDecimalEnabled()**: Manages whether entering a decimal point is allowed based on certain conditions.
   - **lastNumberHasDecimal()** and **getLastNumber()**: Check if the last number in the expression contains a decimal point and retrieve the last number, respectively.
   - **handleParenthesis()**: Handles the input of parentheses based on specific conditions, updating the expression and displaying warnings.

5.4. Known Issues and Future Improvements

- **Known Issues:** Currently, there are no known issues.
- **Future Improvements:** Consider adding scientific calculator functions, memory features, and improving user interface elements.