Міністерство освіти і науки України
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

Лабораторна робота №3.3

з дисципліни
«Інтелектуальні вбудовані системи»

на тему
«Дослідження генетичного алгоритму»

Виконав:
Василиненко Д.Д.

Студент групи ІП-84

Перевірив:

Регіда Павло Геннадійович

Київ 2021

Основні теоретичні відомості

Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку. Можна виділити наступні етапи генетичного алгоритму: (Початок циклу) Розмноження (схрещування) Мутація Обчислити значення цільової функції для всіх особин Формування нового покоління (селекція) Якщо виконуються умови зупинки, то (кінець циклу), інакше (початок циклу). Розглянемо приклад реалізації алгоритму для знаходження цілих коренів діофантового рівняння a+b+2c=15. Згенеруємо початкову популяцію випадковим чином, але з дотриманням умови – усі згенеровані значення знаходяться у проміжку від одиниці до y/2, тобто на відрізку [1;8] (узагалі, границі випадкового генерування можна вибирати на свій розсуд): (1,1,5); (2,3,1); (3,4,1); (3,6,4) Отриманий генотип оцінюється за допомогою функції пристосованості (fitness function). Згенеровані значення підставляються у рівняння, після чого обраховується різниця отриманої правої частини з початковим y. Після цього рахується ймовірність вибору генотипу для ставання батьком – зворотня дельта ділиться на сумму сумарних дельт усіх генотипів. Наступний етап включає в себе схрещування генотипів по методу кросоверу – у якості дітей виступають генотипи, отримані змішуванням коренів – частина йде від одного з батьків, частина від іншого. Лінія кросоверу може бути поставлена в будь-якому місці, кількість потомків також може вибиратися. Після отримання нових генотипів вони перевіряються функцією пристосованості та створюють власних потомків, тобто виконуються дії, описані вище. Ітерації алгоритму відбуваються, поки один з генотипів не отримає Δ=0, тобто його значення будуть розв’язками рівняння.

Завдання

Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки

Вихідний код:

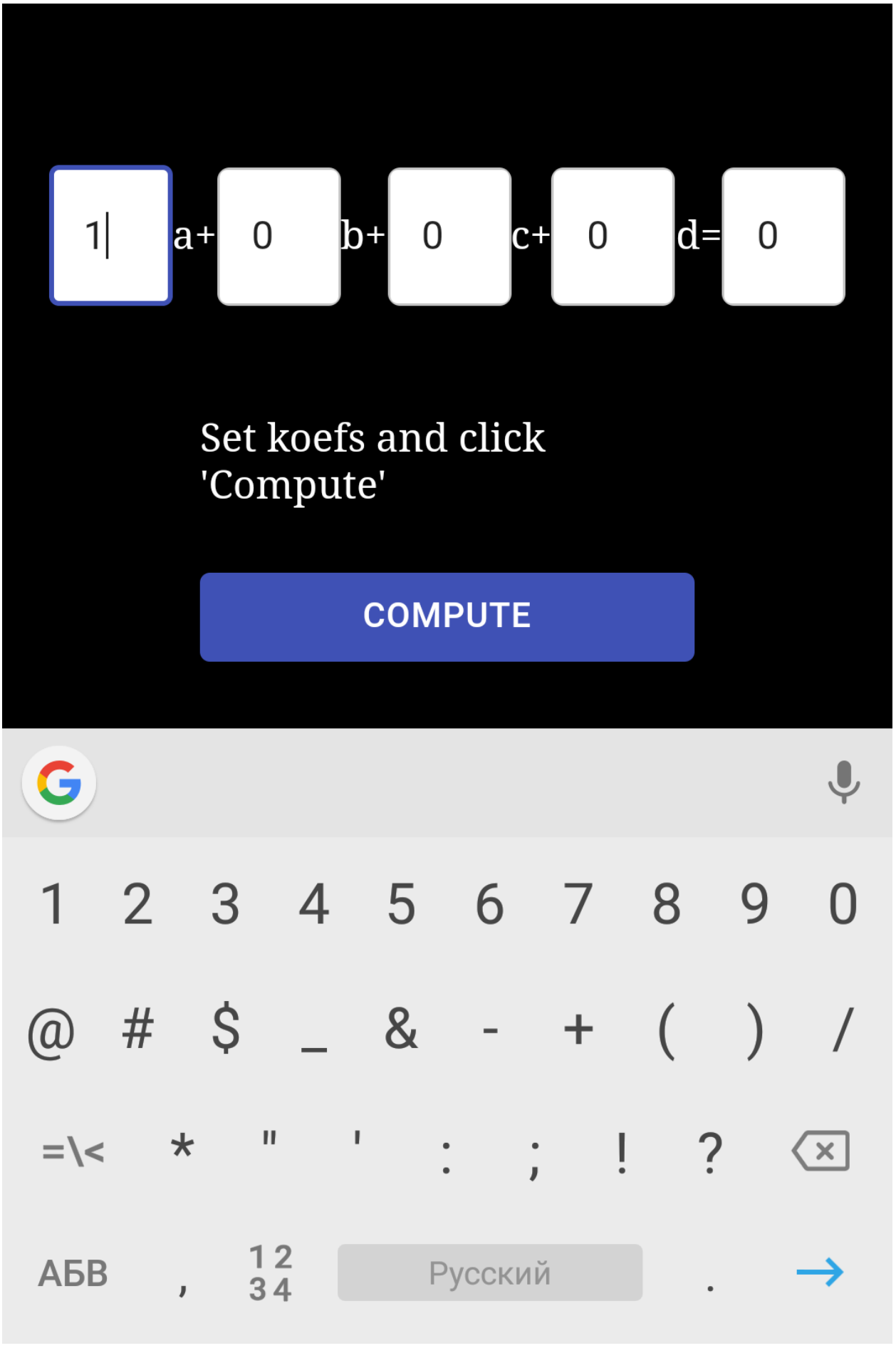
PerceptronLogic.ts

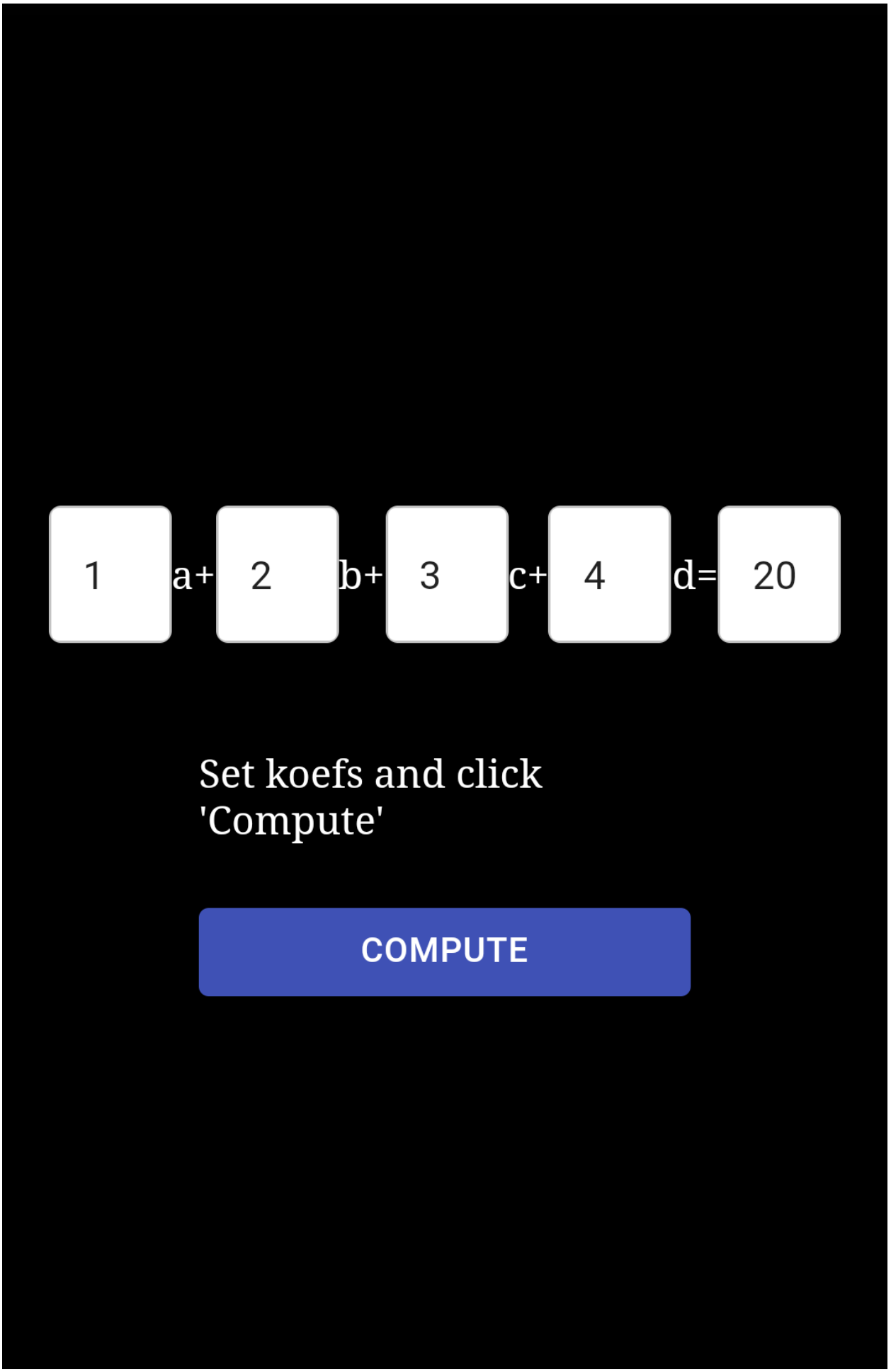
```
const random = (min: number, max: number): number =>
    ~~(Math.random() * (max - min) + min)

class Chromosome {
    public genes: number[] = []
    public fitness: number = Infinity

    protected task: number[] = []
    protected target: number = 0

    constructor(opts: {
        genes: number[],
        task: number[],
        target: number
    }) {
        Object.assign(this, opts);
        this.calcFitness()
    }

    public crossover(partner: Chromosome): Chromosome {
        const child = this.clone()

        const fromParent = child.genes.length / 2
        child.genes = [...child.genes.slice(0, fromParent),
...partner.genes.slice(child.genes.length - fromParent)]

        child.calcFitness()
```

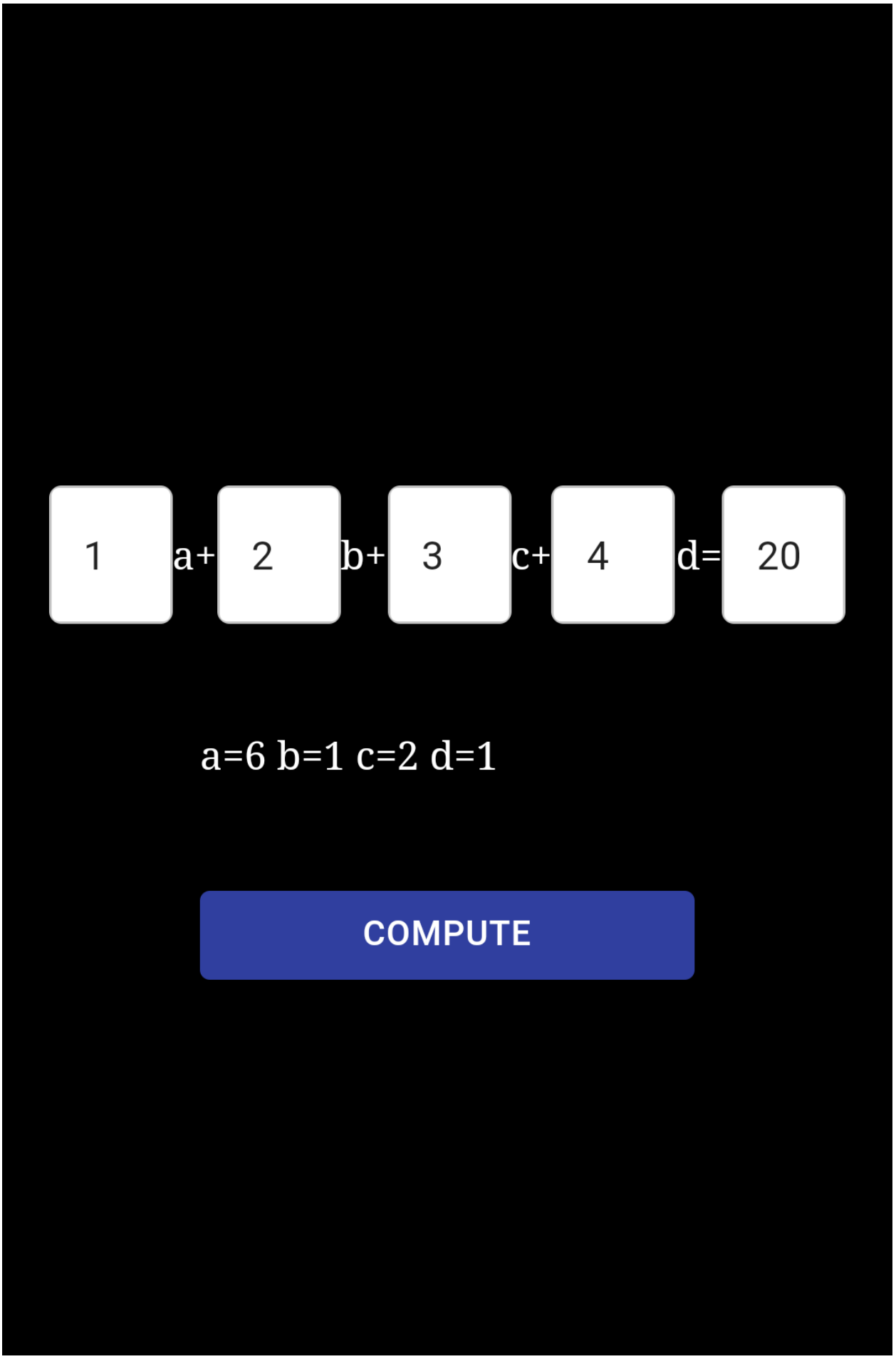
```
        return child
    }

    private calc = () =>
        this.genes.reduce((a, gene, i) => a + (gene * this.task[i]))

    private calcFitness() {
        this.fitness = Math.abs(this.target - this.calc())
    }

    private clone(): Chromosome {
        return Object.assign(Object.create(Object.getPrototypeOf(this)),
this);
    }
}

export class GeneticDiophantus {
    public population: Chromosome[] = []
    public iterations: number = 0

    constructor({task, target, populationSize}: { task: number[], target:
number, populationSize?: number }) {
        const {length} = task
        this.population =
            Array.from(
                {length: populationSize || length},
                () => new Chromosome({
                    genes: Array.from({length}, () => random(1, target /
2)),
                    task: task,
                    target: target,
                })
            )
    }

    public solve(solveFor: number = Infinity): number[] | void {
        while (solveFor--) {
            ++this.iterations
            const chromosome = this.crossover()
            if (chromosome)
                return chromosome.genes
        }
    }
```

```
    private crossover() {
        const children: Chromosome[] = []
        for (let i = 0; i < this.population.length; i++) {
            const parents = this.population
                .map((chromosome) => ({chromosome, probability:
Math.random() * (chromosome.fitness * 1000)}))
                .sort((a, b) => a.probability - b.probability)
            const parent = parents[0].chromosome
            const partner = parents[1].chromosome

            const child = parent.crossover(partner)

            if (child.fitness === 0)
                {
                    return child
                }

            children.push(child)
        }

        this.population = children
    }
}
```

Результати роботи програми


1
a+
0
b+
0
c+
0
d=
0
Set koefs and click
'Compute'
COMPUTE
1 2 3 4 5 6 7 8 9 0
@ # $ _ & - + ( ) /
=\< * " ' : ; ! ?
АБВ
,
Русский
.


1
a+
2
b+
3
c+
4
d=
20
Set koefs and click
'Compute'
COMPUTE


1
a+
2
b+
3
c+
4
d=
20
a=6 b=1 c=2 d=1
COMPUTE

Висновки

Під час виконання лабораторної роботи ми ознайомилися з принципами реалізації генетичного алгоритму, вивчили та дослідили особливості даного алгоритму. Було створено мобільний застосунок для пошуку коренів діофантового рівняння з використанням генетичного алгоритму. При цьому розмір опуляції було обрано як 10, ймовірність мутації – 10%, а метод вибору батьків для схрещування – рулетка. Перевірка отриманих результатів показала, що програма працює коректно, отже, мету лабораторної роботи можна вважати виконаною