БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Ханты-Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт Кафедра автоматики и компьютерных систем

ОТЧЕТ

по лабораторной работе

дисциплина Теория Вычислительных Процессов

Тема: Модель Светофора с Кнопкой

Выполнил: студент группы 609-21

Шумилов Илья Денисович

Принял: доцент кафедры АиКС

Брагинский М. Я.

Сургут 2025 г.

**Задание:**

1. Описать работу ЦА и программную модель ЦА, управляющего двухсекционным светофором с кнопкой и индикатором ожидания

**Граф работы (рисунок 1):**

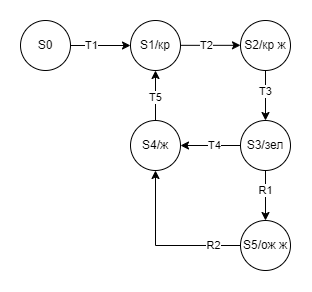


Рисунок 1 - Граф работы ЦА.

**Таблица переходов (таблица 1):**

Таблица 1 – Таблица переходов ЦА

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Входные Сигналы** | **Состояния** | | | | | |
| S0 | S1 | S2 | S3 | S4 | S5 |
| T1 | S1 | - | - | - | - | - |
| T2 | - | S2 | - | - | - | - |
| T3 | - | - | S3 | - | - | - |
| T4 | - | - | - | S4 | - | - |
| T5 | - | - | - | - | S1 | - |
| R1 | - | - | - | S5 | - | - |
| R2 | - | - | - | - | - | S4 |

**Входные сигналы:**

T1 – Таймер был запущен;

T2 – Таймер насчитал 10 секунд;

T3 – Таймер насчитал 2 секунды;

T4 – Таймер насчитал 10 секунд;

T5 – Таймер насчитал 2 секунды;

R1 – Была нажата кнопка запроса на переход;

R2 – Прошло 2 секунды после нажатия кнопки запроса на переход.

**Состояния:**

S0 – Светофор выключен, не горит ни один сигнал светофора;

S1 – Горит красный сигнал светофора;

S2 – Горят красный и желтый сигналы светофора;

S3 – Горит зеленый сигнал светофора;

S4 – Горит желтый сигнал светофора;

S5 – Горит зеленый сигнал светофора и индикатор ожидания перехода.

**Код программного обеспечения (листинг 1):**

Код был написан на языке программирования Typescript в среде разработки Visual Studio Code.

Листинг 1. Код программного обеспечения:

enum State {

    OFF,        // S0

    RED,        // S1

    YELLOW\_RED, // S2

    GREEN,      // S3

    YELLOW,     // S4

    WAIT        // S5

}

enum IncomingSignal {

    RED,            // T1

    YELLOW\_RED,     // T2

    GREEN,          // T3

    YELLOW,         // T4

    REQUEST\_WALK    // R1

}

class TrafficLightStateMachine {

    private \_state: State;

    public get state(): State {

        return this.\_state;

    }

    constructor() {

        this.\_state = State.OFF;

    }

    public receiveSignal(signal: IncomingSignal): State {

        switch (signal) {

            case IncomingSignal.RED:

                this.handleRedSignal();

                break;

            case IncomingSignal.YELLOW\_RED:

                this.handleYellowRedSignal();

                break;

            case IncomingSignal.GREEN:

                this.handleGreenSignal();

                break;

            case IncomingSignal.YELLOW:

                this.handleYellowSignal();

                break;

            case IncomingSignal.REQUEST\_WALK:

                this.handleRequestWalkSignal();

                break;

        }

        return this.state;

    }

    private handleRedSignal(): void {

        if (this.\_state == State.OFF || this.\_state == State.YELLOW) {

            this.\_state = State.RED;

        }

    }

    private handleYellowRedSignal(): void {

        if (this.\_state == State.RED) {

            this.\_state = State.YELLOW\_RED;

        }

    }

    private handleGreenSignal(): void {

        if (this.\_state == State.YELLOW\_RED) {

            this.\_state = State.GREEN;

        }

    }

    private handleYellowSignal(): void {

        if (this.\_state == State.GREEN || this.\_state == State.WAIT) {

            this.\_state = State.YELLOW;

        }

    }

    private handleRequestWalkSignal(): void {

        if (this.\_state == State.GREEN) {

            this.\_state = State.WAIT;

            setTimeout(() => {

                this.handleYellowSignal();

            }, 2000);

        }

    }

}

class TrafficLightDomManager {

    readonly redCarLight: NodeListOf<HTMLElement>;

    readonly yellowCarLight: NodeListOf<HTMLElement>;

    readonly greenCarLight: NodeListOf<HTMLElement>;

    readonly redPedestrianLight: NodeListOf<HTMLElement>;

    readonly greenPedestrianLight: NodeListOf<HTMLElement>;

    readonly timerText: NodeListOf<HTMLElement>;

    readonly waitText: NodeListOf<HTMLElement>;

    readonly waitButton: NodeListOf<HTMLElement>;

    private elapsedTime: number;

    private STATE\_DURATIONS: Object = {

        [State.OFF]: 0,

        [State.RED]: 10,

        [State.YELLOW\_RED]: 2,

        [State.GREEN]: 10,

        [State.YELLOW]: 2

    }

    private stateMachine: TrafficLightStateMachine;

    constructor() {

        this.redCarLight = document.querySelectorAll('#redCarLight') as NodeListOf<HTMLElement>;

        this.yellowCarLight = document.querySelectorAll('#yellowCarLight') as NodeListOf<HTMLElement>;

        this.greenCarLight = document.querySelectorAll('#greenCarLight') as NodeListOf<HTMLElement>;

        this.redPedestrianLight = document.querySelectorAll('#redPedestrianLight') as NodeListOf<HTMLElement>;

        this.greenPedestrianLight = document.querySelectorAll('#greenPedestrianLight') as NodeListOf<HTMLElement>;

        this.timerText = document.querySelectorAll('#timerText') as NodeListOf<HTMLElement>;

        this.waitText = document.querySelectorAll('#waitText') as NodeListOf<HTMLElement>;

        this.waitButton = document.querySelectorAll('#waitButton') as NodeListOf<HTMLElement>;

        if (!this.redCarLight ||

            !this.yellowCarLight ||

            !this.greenCarLight ||

            !this.redPedestrianLight ||

            !this.greenPedestrianLight ||

            !this.timerText ||

            !this.waitText ||

            !this.waitButton) {

            alert('Not all required elements are present on page!');

        }

        this.waitButton.forEach(btn => {

            btn.addEventListener('click', () => {

                this.sendSignalToStateMachine(IncomingSignal.REQUEST\_WALK);

            });

        });

        this.stateMachine = new TrafficLightStateMachine();

        this.handleState();

        this.elapsedTime = 0;

    }

    public work(): void {

        setInterval((): void => {

            this.handleTimer();

            this.handleState();

        }, 1000);

    }

    private handleTimer(): void {

        if (this.stateMachine.state != State.WAIT) {

            this.elapsedTime++;

            this.timerText.forEach(timer => {

                timer.innerHTML = `Time: ${this.elapsedTime}`;

            });

            if (this.elapsedTime >= this.STATE\_DURATIONS[this.stateMachine.state]) {

                this.transitionToNextState();

            }

        } else {

            this.elapsedTime = 0;

            this.timerText.forEach(timer => {

                timer.innerHTML = 'Time: ...';

            });

        }

    }

    private transitionToNextState(): void {

        this.elapsedTime = 0;

        switch (this.stateMachine.state) {

            case State.OFF:

                this.sendSignalToStateMachine(IncomingSignal.RED);

                break;

            case State.RED:

                this.sendSignalToStateMachine(IncomingSignal.YELLOW\_RED);

                break;

            case State.YELLOW\_RED:

                this.sendSignalToStateMachine(IncomingSignal.GREEN);

                break;

            case State.GREEN:

                this.sendSignalToStateMachine(IncomingSignal.YELLOW);

                break;

            case State.YELLOW:

                this.sendSignalToStateMachine(IncomingSignal.RED);

                break;

        }

    }

    private sendSignalToStateMachine(signal: IncomingSignal): void {

        this.stateMachine.receiveSignal(signal);

    }

    private handleState(): void {

        switch (this.stateMachine.state) {

            case State.OFF:

                this.handleOffState();

                break;

            case State.RED:

                this.handleRedState();

                break;

            case State.YELLOW\_RED:

                this.handleYellowRedState();

                break;

            case State.GREEN:

                this.handleGreenState();

                break;

            case State.YELLOW:

                this.handleYellowState();

                break;

            case State.WAIT:

                this.handleWaitState();

                break;

        }

    }

    private handleOffState(): void {

        this.redCarLight.forEach(el => el.classList.add('inactive'))

        this.yellowCarLight.forEach(el => el.classList.add('inactive'));

        this.greenCarLight.forEach(el => el.classList.add('inactive'));

        this.redPedestrianLight.forEach(el => el.classList.add('inactive'));

        this.greenPedestrianLight.forEach(el => el.classList.add('inactive'));

    }

    private handleRedState(): void {

        this.redCarLight.forEach(el => el.classList.remove('inactive'));

        this.yellowCarLight.forEach(el => el.classList.add('inactive'));

        this.redPedestrianLight.forEach(el => el.classList.add('inactive'));

        this.greenPedestrianLight.forEach(el => el.classList.remove('inactive'));

        this.waitText.forEach(el => el.classList.remove('active'));

    }

    private handleYellowRedState(): void {

        this.yellowCarLight.forEach(el => el.classList.remove('inactive'));

        this.redPedestrianLight.forEach(el => el.classList.remove('inactive'));

        this.greenPedestrianLight.forEach(el => el.classList.add('inactive'));

    }

    private handleGreenState(): void {

        this.redCarLight.forEach(el => el.classList.add('inactive'));

        this.yellowCarLight.forEach(el => el.classList.add('inactive'));

        this.greenCarLight.forEach(el => el.classList.remove('inactive'));

    }

    private handleYellowState(): void {

        this.yellowCarLight.forEach(el => el.classList.remove('inactive'));

        this.greenCarLight.forEach(el => el.classList.add('inactive'));

    }

    private handleWaitState(): void {

        this.waitText.forEach(el => el.classList.add('active'));

    }

}

let domManager = new TrafficLightDomManager();

domManager.work();

**Пример работы программы (рисунки 2-7):**

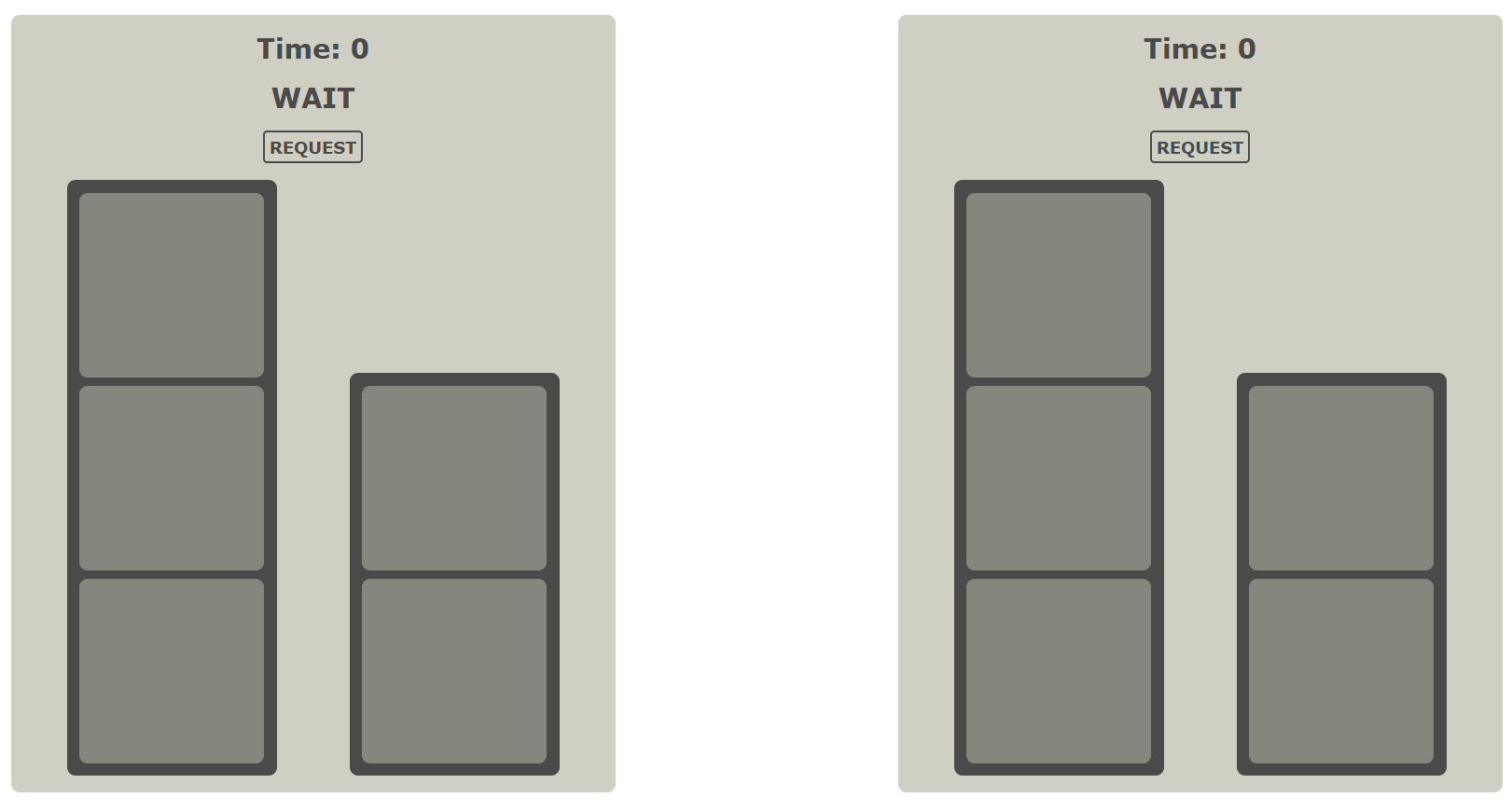


Рисунок 2 – Светофор в состоянии S0

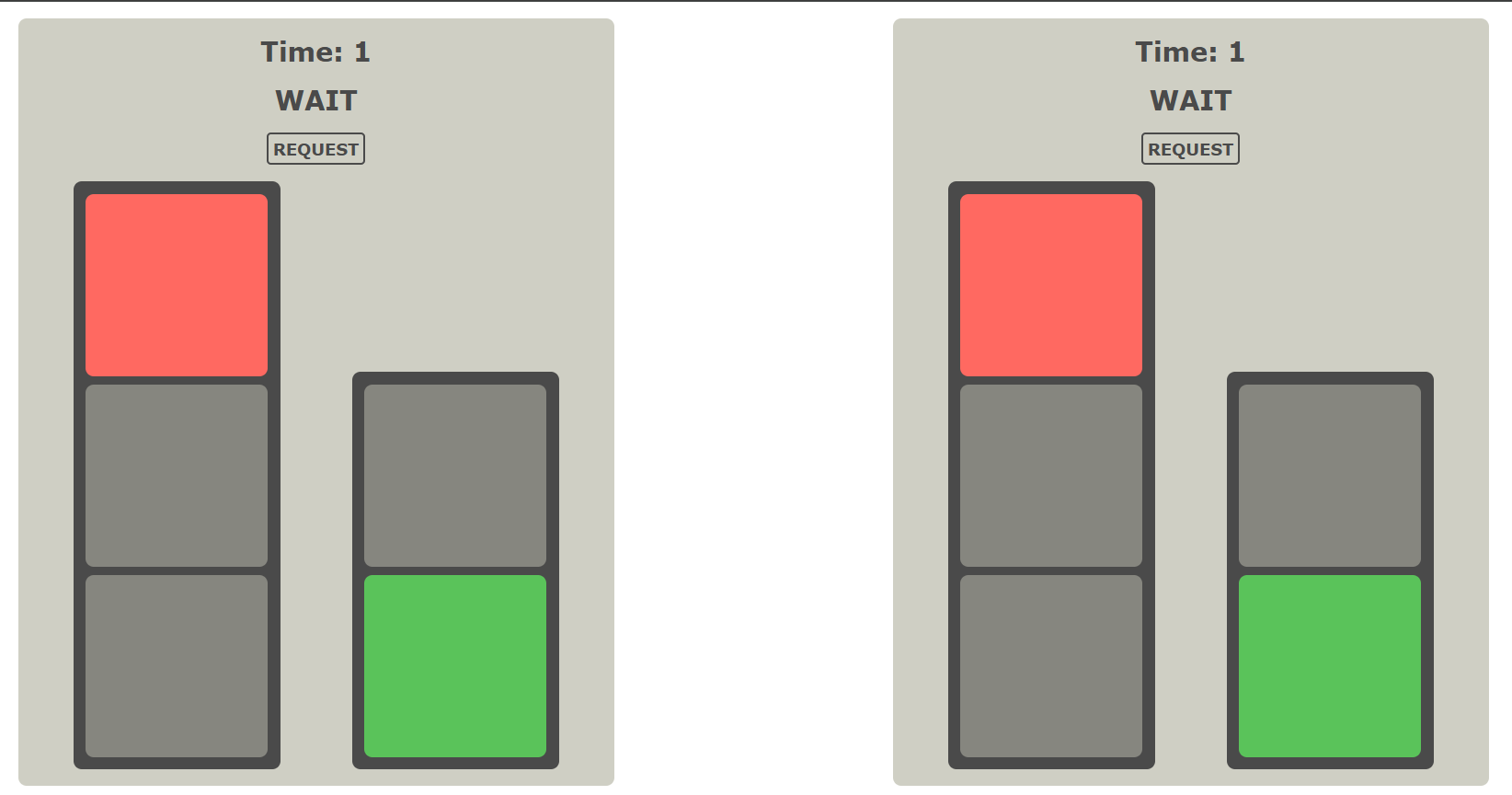


Рисунок 3 – Светофор в состоянии S1

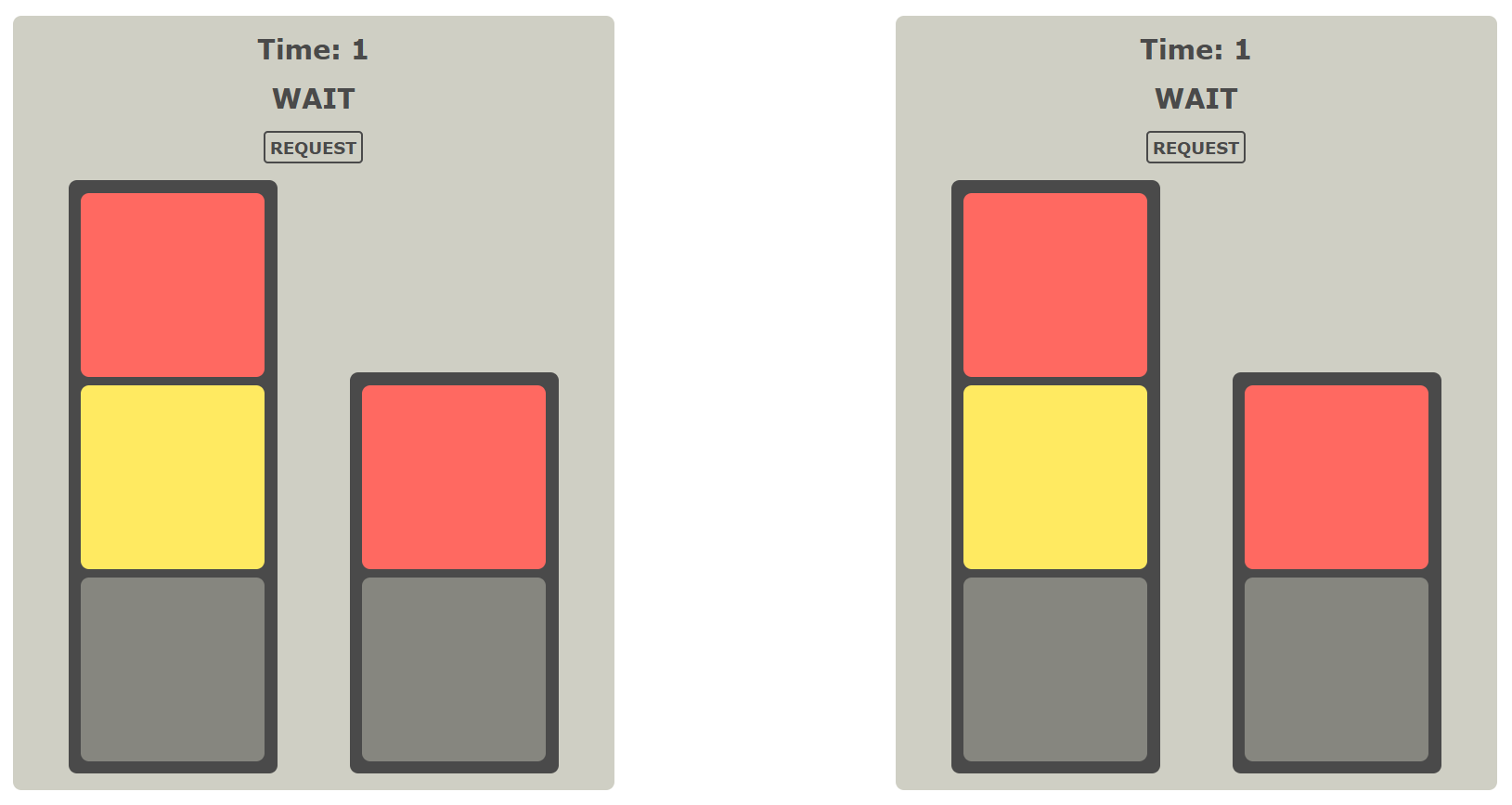


Рисунок 4 – Светофор в состоянии S2

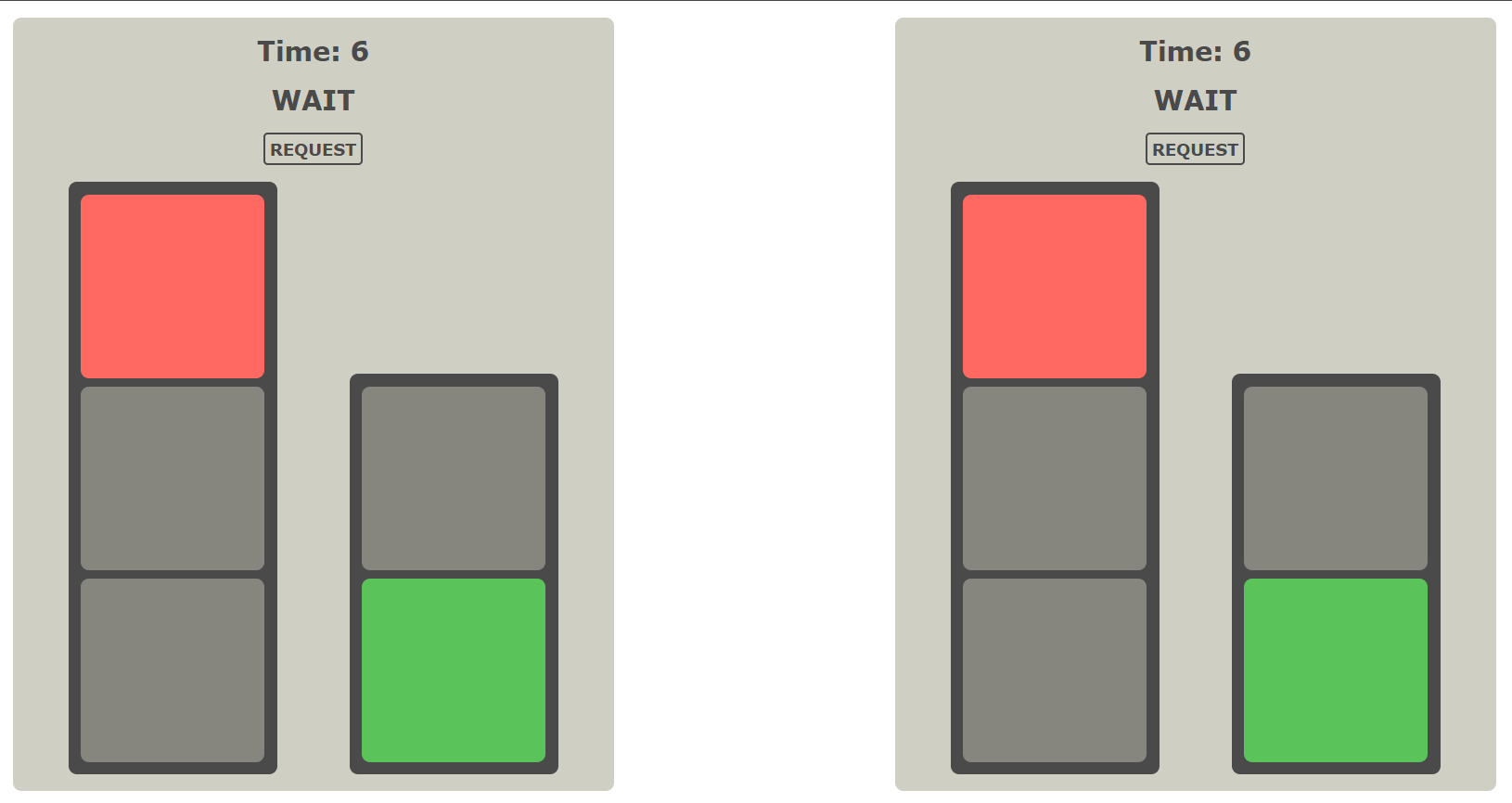


Рисунок 5 – Светофор в состоянии S3

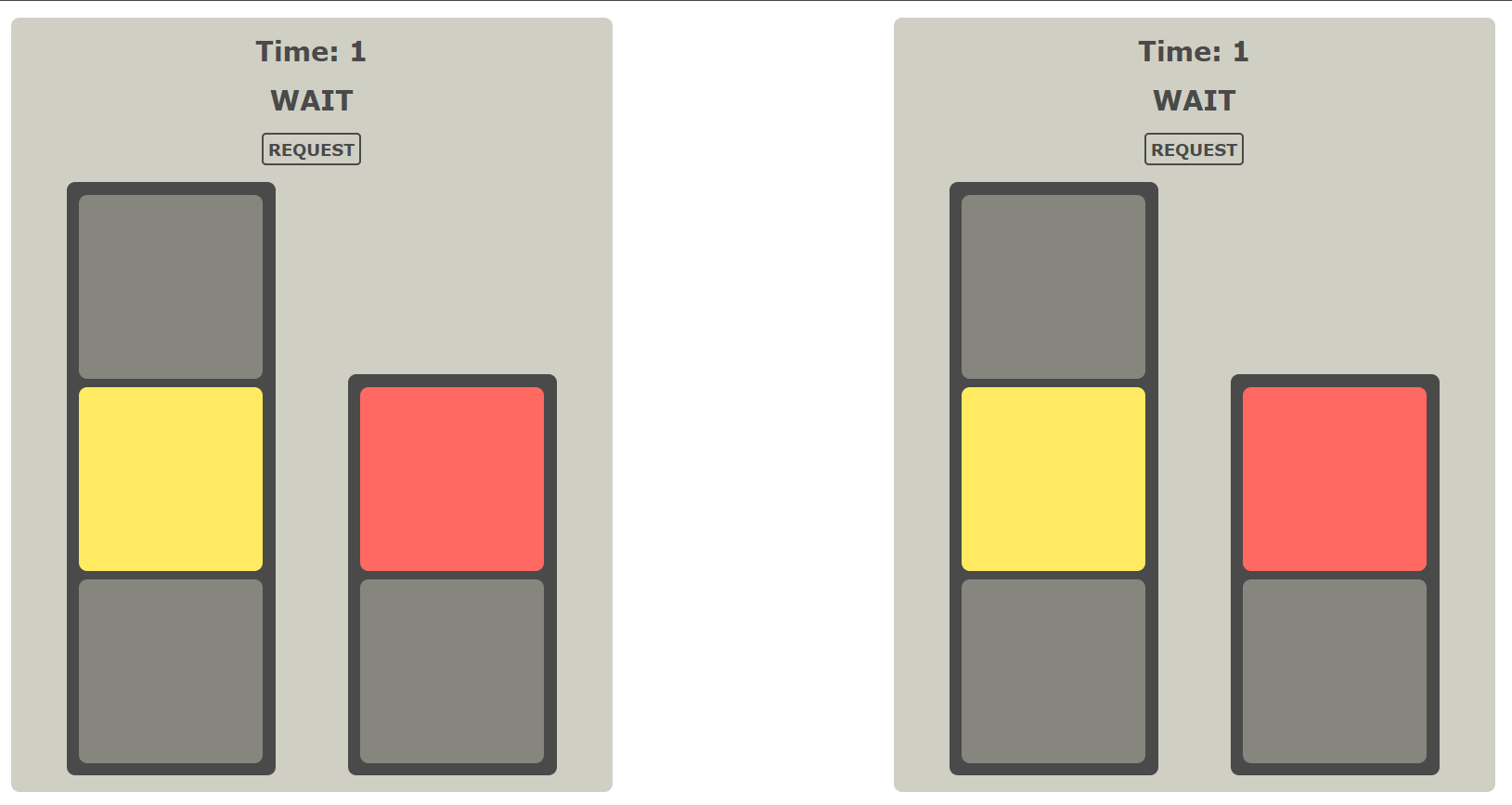


Рисунок 6 – Светофор в состоянии S4

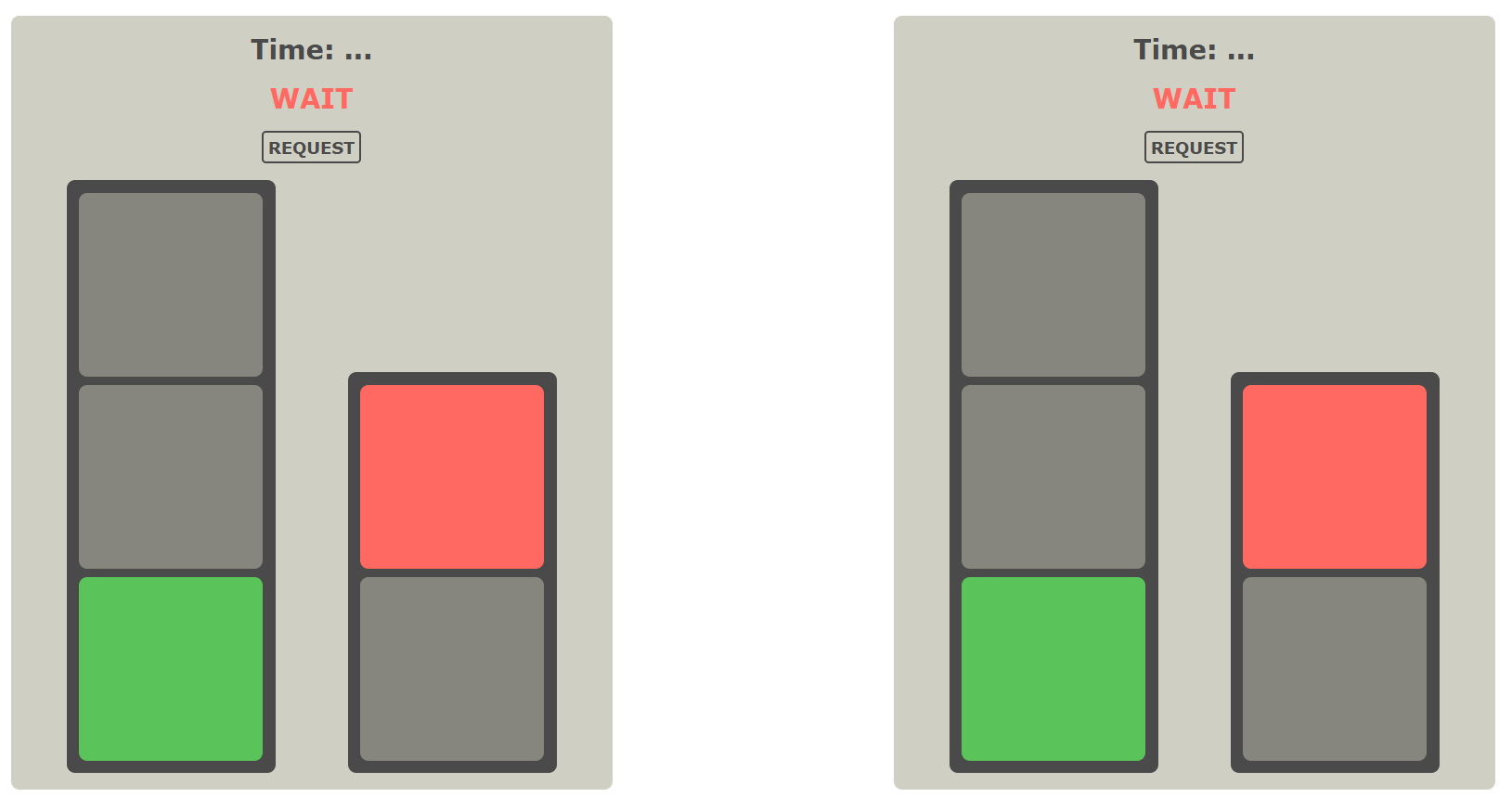


Рисунок 7 – Светофор в состоянии S5

Выводы: в ходе лабораторной работы была создана программная реализация цифрового автомата модели светофора.