Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 6

Дисциплина: Моделирование

Тема: Моделирование работы подъемников горнолыжного склона

Студент: Платонова О. С.

Группа: ИУ7-75Б

Оценка (баллы): _____

Преподаватель: Рудаков И. В.

Москва, 2021 г.

Цель работы: моделирование работы подъемников горнолыжного склона.

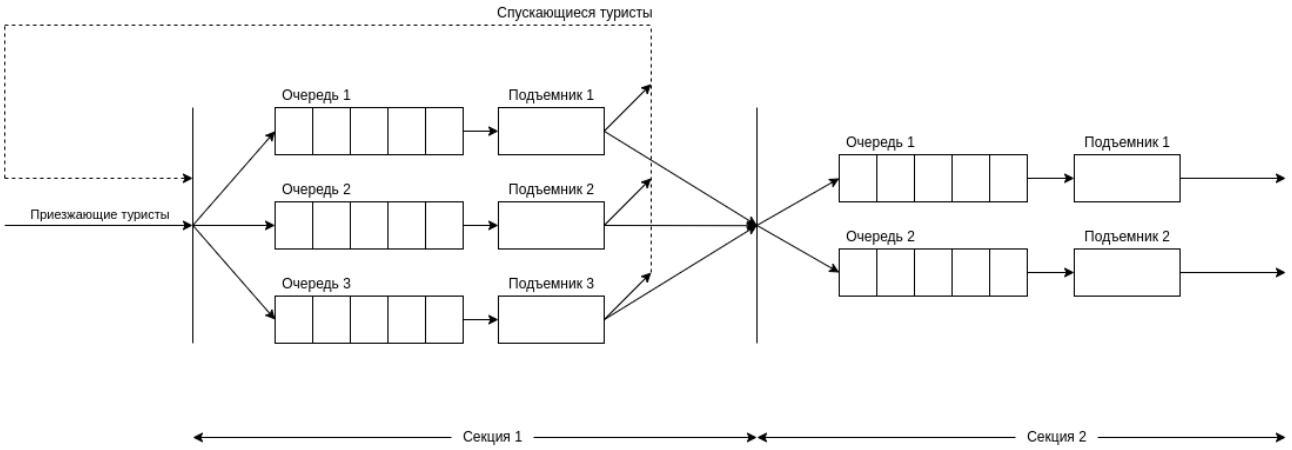
Описание модели

Горнолыжный склон Эльбруса разделен на три секции. Рассмотрим две верхние из них. Секция, расположенная посередине (секция 1) имеет 3 подъемника: два расположены с одной стороны, третий — с другой. Верхняя секция (секция 2) имеет только два подъемника.

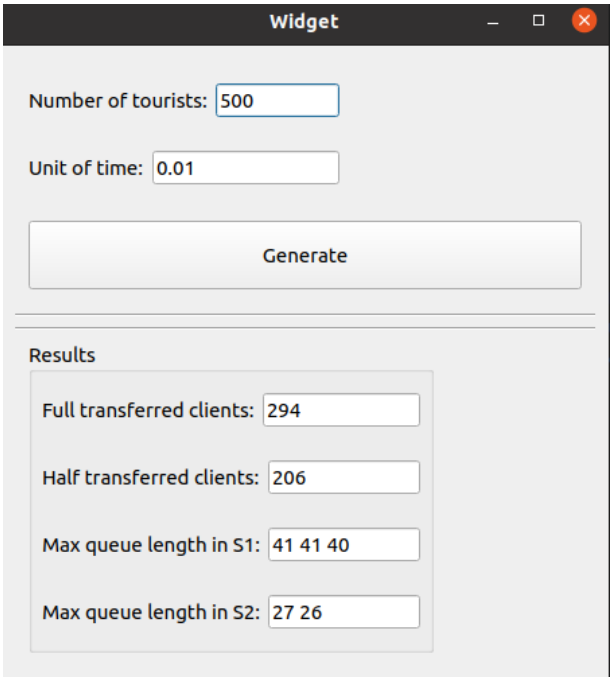
Отдыхающие подъезжают к секции 1 с интервалом 30 ± 5 секунд. Они встают в наименьшую очередь на один из трех подъемников. Скорость работы подъемников разная: 120 ± 20 секунд, 150 ± 30 секунд, 100 ± 10 секунд.

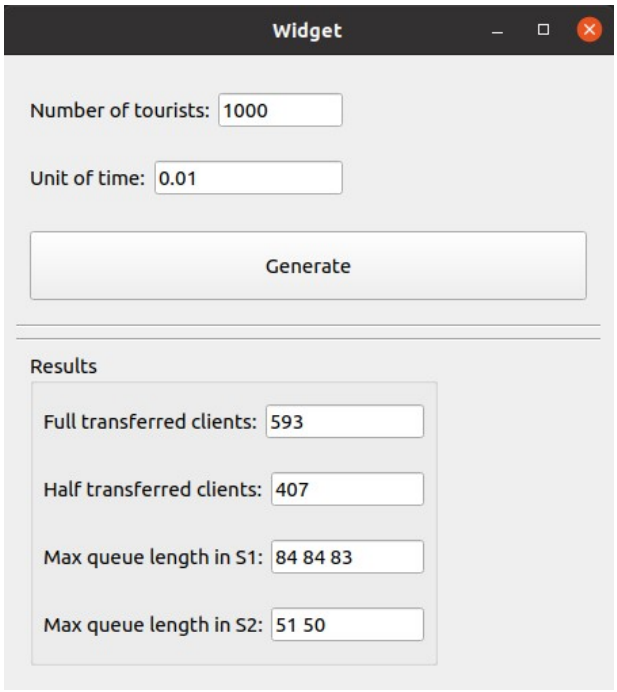
Поднявшись до верхней секции 40% отдыхающих спускаются; остальные — встают в очередь на следующий подъемник. Скорость верхних подъемников: 170 ± 15 секунд, 160 ± 10 секунд.

Схема СМО

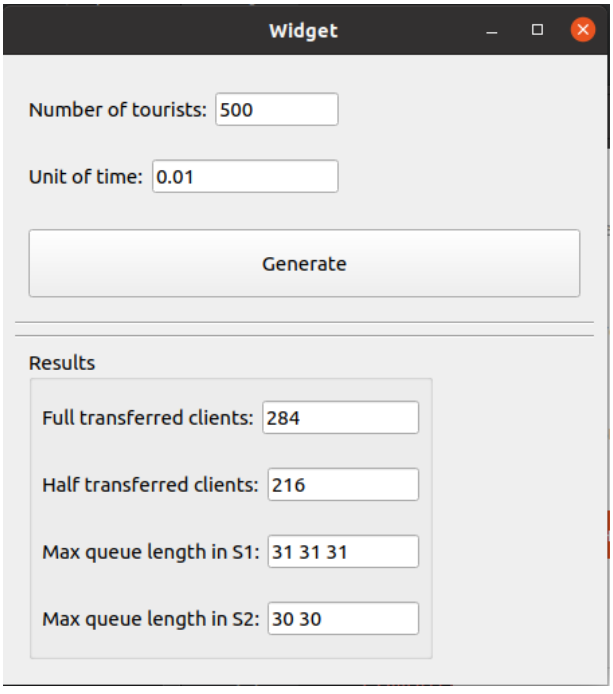


Результаты





С увеличением числа отдыхающих (разгар сезона) максимальная длина очереди становится неприемлемой для функционирования курорта. Однако увеличения скорости работы подъемников секции 1 на 10 секунд приводит к сокращению очереди примерно на 40%. Ее размер становится приближенным к реальным условиям.



Листинг

Computer (секция 2)

```h
class Computer
{
public:
    Computer(std::default_random_engine *re, std::vector<int> *_queue);

    void setEvenDistribution(const float _a, const float _b);
    double even();
    bool liftTourist(double dt);
    int getMaxQueueLen();

private:
    std::default_random_engine *gnt;
    std::vector<int> *queue;
    int maxQueueLen = 0;
    double timer;
    float a, b;
    bool busy;

    void updateMaxQueueLen();
};
```

```cpp
bool Computer::liftTourist(double dt)
{
    timer -= dt;

    updateMaxQueueLen();

    if (busy && timer <= 0) {
        busy = false;
        return true;
    }

    if (!busy && queue->size() != 0) {
        (*queue).erase((*queue).begin());
        timer = even();
        busy = true;
        return false;
    }

    return false;
}

void Computer::updateMaxQueueLen()
{
    if (static_cast<int>(queue->size()) > maxQueueLen) {
        maxQueueLen = queue->size();
    }
}
```

Operator (секция 1)

```h
class Operator
{
public:
    Operator(std::default_random_engine *re,
             std::vector<int> *_queue, std::vector<std::vector<int>> *_queueGroup);

    void setEvenDistribution(float _a, float _b);
    double even();
    bool liftTourist(double step);
    int getMaxQueueLen();

private:
    std::default_random_engine *gnt;
    std::vector<int> *queue; int maxQueueLen;
    std::vector<std::vector<int>> *queueGroup;
    double timer;
    float a, b;
    bool busy;

    void queueTourist();
    void updateMaxQueueLen();
};
```

```cpp
bool Operator::liftTourist(double step)
{
    timer -= step;

    updateMaxQueueLen();

    if (!busy && queue->size() != 0) {
        busy = true;
        timer = even();
    }

    if (busy && timer <= 0) {
        if (random() % 100 <= 40) {
            (*queue).erase((*queue).begin());
            busy = false;
            return false;
        }
        else {
            (*queue).erase((*queue).begin());
            queueTourist();
            busy = false;
            return true;
        }
    }
    return false;
}

void Operator::queueTourist()
{
    size_t minLen = (*queueGroup)[0].size();
    int minIdx = 0;

    for (size_t i = 1; i < queueGroup->size(); i++) {
        if ((*queueGroup)[i].size() < minLen) {
            minLen = (*queueGroup)[i].size();
            minIdx = i;
        }
    }
    (*queueGroup)[minIdx].push_back(1);
}
```

Generator

```cpp
class Generator
{
public:
    Generator();

    void setRnd(std::default_random_engine *re);
    void setEvenDistribution(const float _a, const float _b);
    void setSection(std::vector<std::vector<int>> *_section);

    double even();
    bool produceTourist(double dt);

private:
    float a, b;
    double timer;
    std::vector<std::vector<int>> *section;
    std::default_random_engine *gnt;

    void queueTourist();
};
```

```cpp
bool Generator::produceTourist(double dt)
{
    timer -= dt;

    if (timer <= 0) {
        timer = even();
        queueTourist();
        return true;
    }
    return false;
}

void Generator::queueTourist()
{
    size_t minLen = (*section)[0].size();
    int minIdx = 0;

    for (size_t i = 1; i < section->size(); i++) {
        if ((*section)[i].size() < minLen) {
            minLen = (*section)[i].size();
            minIdx = i;
        }
    }
    (*section)[minIdx].push_back(1);
}
```

Model

```cpp
class Model
{
public:
    Model();
    Result generate(const int numTourists, double step);

private:
    Generator generator;
    std::vector<Operator> lowLifts;
    std::vector<Computer> highLifts;
    std::vector<std::vector<int>> section1, section2;

    std::random_device rd;
    std::default_random_engine gnt;

    int lowTransfer(const double step);
    int highTransfer(const double step);

    std::vector<int> getLowMaxLen();
    std::vector<int> getHighMaxLen();
};
```

```cpp
Model::Model() :
    gnt(rd())
{
    section1 = vector<vector<int>>(3, vector<int>());
    section2 = vector<vector<int>>(2, vector<int>());

    generator.setEvenDistribution(25, 35);
    generator.setRnd(&gnt);
    generator.setSection(&section1);

    Operator lowLift1(&gnt, &section1[0], &section2),
             lowLift2(&gnt, &section1[1], &section2),
             lowLift3(&gnt, &section1[2], &section2);

    lowLift1.setEvenDistribution(90, 130);
    lowLift2.setEvenDistribution(110, 170);
    lowLift3.setEvenDistribution(80, 100);
    lowLifts = {lowLift1, lowLift2, lowLift3};

    Computer highLift1(&gnt, &section2[0]),
             highLift2(&gnt, &section2[1]);

    highLift1.setEvenDistribution(155, 185);
    highLift2.setEvenDistribution(150, 170);
    highLifts = {highLift1, highLift2};
}
```

```cpp
Result Model::generate(const int numTourists, double step)
{
    int curTourists = 0, procTourists = 0;
    int fullTransf = 0, halfTransf = numTourists;

    while (curTourists < numTourists) {
        bool tourist = generator.produceTourist(step);
        if (tourist) {
            curTourists++;
        }

        int lowTransferred = lowTransfer(step);
        fullTransf += lowTransferred;
        halfTransf -= lowTransferred;
        procTourists += highTransfer(step);
    }

    while (procTourists < fullTransf) {
        int lowTransferred = lowTransfer(step);
        fullTransf += lowTransferred;
        halfTransf -= lowTransferred;
        procTourists += highTransfer(step);
    }

    Result res;
    res.FTransf = fullTransf;
    res.HTransf = halfTransf;
    res.LMaxLen = getLowMaxLen();
    res.HMaxLen = getHighMaxLen();

    return res;
}
```