Лабораторна Робота №4

Дудник Олексій КН-19-2

Рефакторинг програмного коду

Мета роботи: навчитися виконувати реорганізацію програмного коду на підставі шаблонів рефакторінгу.

Хід виконання роботи

1. Вивчити теоретичні відомості.
2. Виконати аналіз програмного коду розроблюваного ПО і модульних тестів з метою виявлення погано організованого код.
3. Використовуючи шаблони рефакторінгу, виконати реорганізацію програмного коду розроблюваного ПО і модульних тестів.
4. Перевірити успішність виконання всіх модульних тестів.
5. Виконати опис вироблених операцій рефакторінгу (було-стало-шаблон рефакторінгу).
6. В разі необхідності коректувати проектну документацію (діаграми класів, послідовностей).
7. В разі необхідності коректувати проектну документацію (діаграми класів, послідовностей).

Підстави для проведення рефакторінгу:

1. Код дублюється. Це призводить до необхідності вносити зміни в однакові ділянки коду
2. Розширення програми (було додано 2 генератори)
3. Класс має багато обов'язків
4. Було використано паттерн рефакторінгу Extract class

Було

```
double Voice::getSample(double time, WaveType *type)
{
    double amplitude = m_adsrEnvelope.getAmplitude(time);
    double sample = 0.0f;
    switch (*type) {
        case SINE:
            sample = std::sin(M_PI * 2 * m_frequency * time);
            break;
        case SQUARE:
            sample = 0.8 * (std::sin(2 * M_PI * m_frequency * time) > 0 ? 1.0 : -1.0);
            break;
        case SAW:
        {
            // analog saw
            //            float output = 0.0f;
            //
            //            for (float n = 1.0f; n < 40.0f; n++) {
            //                output += (std::sin(n * 2 * M_PI * m_frequency * time)) / n;
            //            }
            //            sample = 0.9 * output * (2.0f / M_PI);

            // digital saw
            sample = (2.0 / M_PI) * (m_frequency * M_PI * std::fmod(time, 1.0 / m_frequency) - (M_PI / 2.0));
            break;
        }
        case TRIANGLE:
            sample = std::asin(std::sin(2 * M_PI * m_frequency * time) * (2.0 / M_PI));
            break;
        default:
            sample = 0.0;
    }

    if (m_adsrEnvelope.isNoteOff() && m_adsrEnvelope.getCurrentAmplitude() == 0.0f)
        m_active = false;
    return amplitude * sample;
//    return sample;
}
```

Рис. 1: Розрахунок значення генератора до

```
// OSC 1
ImGui::Begin("Oscillator 1");
ImGuiKnobs::Knob("Gain", &m_osc1Gain, 0.0f, 1.0f, 0.05f);
ImGui::SameLine();
ImGuiKnobs::KnobInt("Offset", &m_osc1NoteOffset, 0, 48, 12);
ImGui::SameLine();
ImGui::Checkbox("Active", &m_isOsc1Active);
if (ImGui::Selectable("Sine", m_osc1Type == SINE)) m_osc1Type = SINE;
if (ImGui::Selectable("Square", m_osc1Type == SQUARE)) m_osc1Type = SQUARE;
if (ImGui::Selectable("Saw", m_osc1Type == SAW)) m_osc1Type = SAW;
if (ImGui::Selectable("Triangle", m_osc1Type == TRIANGLE)) m_osc1Type = TRIANGLE;
if (ImGui::Selectable("Noise", m_osc1Type == NOISE)) m_osc1Type = NOISE;
ImGui::End();

// OSC2
ImGui::Begin("Oscillator 2");
ImGuiKnobs::Knob("Gain", &m_osc2Gain, 0.0f, 1.0f, 0.05f);
ImGui::SameLine();
ImGuiKnobs::KnobInt("Offset", &m_osc2NoteOffset, 0, 48, 12);
ImGui::SameLine();
ImGui::Checkbox("Active", &m_isOsc2Active);
if (ImGui::Selectable("Sine", m_osc2Type == SINE)) m_osc2Type = SINE;
if (ImGui::Selectable("Square", m_osc2Type == SQUARE)) m_osc2Type = SQUARE;
if (ImGui::Selectable("Saw", m_osc2Type == SAW)) m_osc2Type = SAW;
if (ImGui::Selectable("Triangle", m_osc2Type == TRIANGLE)) m_osc2Type = TRIANGLE;
if (ImGui::Selectable("Noise", m_osc2Type == NOISE)) m_osc2Type = NOISE;
ImGui::End();
```

Рис. 2: UI генераторів до

Стало

```cpp
double Voice::getSample(double time)
{
    double amplitude = m_adsr->getAmplitude(time);

    double sample = 0.0;
    double lfoFreq = 0.0;

    for (int i = 0; i < m_oscillators.size(); i++) {
        if (i == 0 && m_lfo->isActive()) lfoFreq = m_lfo->getFrequency();
        sample += m_oscillators[i]->getSample(m_key, time, lfoFreq);
    }

    if (m_adsr->isNoteOff() && m_adsr->getCurrentAmplitude() == 0.0)
        m_isActive = false;

    return amplitude * sample;
}
```

Рис. 3: Розрахунок значення генератора після

```cpp
// OSCs
for (int i = 0; i < 3; i++) {
    std::string name = "Oscillator " + std::to_string(i + 1);
    ImGui::Begin(name.c_str());
    ImGuiKnobs::Knob("Gain", &m_oscGain[i], 0.0f, 1.0f, 0.05f);
    ImGui::SameLine();
    ImGuiKnobs::KnobInt("Offset", &m_oscNoteOffset[i], 0, 48, 12);
    ImGui::SameLine();
    ImGui::Checkbox("Active", &m_isOscActive[i]);
    if (ImGui::Selectable("Sine", m_oscType[i] == SINE)) m_oscType[i] = SINE;
    if (ImGui::Selectable("Square", m_oscType[i] == SQUARE)) m_oscType[i] = SQUARE;
    if (ImGui::Selectable("Saw", m_oscType[i] == SAW)) m_oscType[i] = SAW;
    if (ImGui::Selectable("Triangle", m_oscType[i] == TRIANGLE)) m_oscType[i] = TRIANGLE;
    if (ImGui::Selectable("Noise", m_oscType[i] == NOISE)) m_oscType[i] = NOISE;
    ImGui::End();
}
```

Рис. 4: UI генераторів після

```cpp
#ifndef Oscillator_h
#define Oscillator_h

enum WaveType
{
    SINE,
    SQUARE,
    TRIANGLE,
    SAW,
    NOISE
};

class Oscillator
{
public:
    Oscillator(WaveType *type, bool *isActive, int *noteOffset, float *gain);
    double getSample(int key, double time, double lfoFrequency);
    void draw();

private:
    WaveType *m_type;
    float *m_gain;
    bool *m_isActive;
    int *m_noteOffset;

    double calculateFrequency(int key);
};

#endif /* Oscillator_h */
```

Рис. 5: Класс генератора

```
26 double Oscillator::getSample(int key, double time, double lfoFrequency)
27 {
28     double sample = 0.0;
29     if (!(*m_isActive))
30         return sample;
31 
32     double frequency = calculateFrequency(key);
33     double radians = 2 * M_PI * frequency * time + 0.001 * frequency * std::sin(2 * M_PI
           * lfoFrequency * time);
34 
35     switch (*m_type) {
36         case SINE:
37             sample = std::sin(radians);
38             break;
39         case SQUARE:
40             sample = 0.8 * (std::sin(radians) > 0 ? 1.0f : -1.0f);
41         break;
42         case SAW: {
43             double out = 0.0;
44             for (double n = 1.0; n < 50.0; n++) {
45                 out += (std::sin(n * radians)) / n;
46             }
47             sample = out;
48             break;
49         }
50         case TRIANGLE:
51             sample = std::asin(std::sin(radians) * (2.0 / M_PI));
52             break;
53         case NOISE:
54             sample = 2.0 * ((double)rand() / (double)RAND_MAX) - 1.0;
55             break;
56         default:
57             sample = 0.0f;
58             break;
59     }
60 
61     return *m_gain * sample;
62 }
```

Рис. 6: Розрахунок значення семпла у генераторі

Підстави для проведення рефакторінгу:

1. Метод (конструктор) мав дуже багато параметрів. Це ускладнює розуміння коду, а також можливість додаваня нового коду
2. Усі параметри можна згрупувати
3. Тому було використано паттерн Introduce parameter object

Було

```
Voice::Voice(float *attackTime, float *decayTime, float *sustainAmplitude, float
    *releaseTime, float *filterLowCuttoff, float *filterHighCuttoff, FilterType
    *filterType, bool *isOsc1Active, WaveType *osc1Type, float *osc1Gain, bool
    *isOsc2Active, WaveType *osc2Type, float *osc2Gain, bool *isOsc3Active, WaveType
    *osc3Type, float *osc3Gain, float *lfoFrequency, bool *lfoActive, int *osc1Offset,
    int *osc2Offset, int *osc3Offset, bool *isFilter)
: m_frequency(0.f)
, m_timeOn(0.f)
, m_timeOff(0.f)
, m_active(false)
, m_key(0)
, m_isFilter(isFilter)
{
```

Рис. 7: Конструктор Voice до

```
{
    m_adsrEnvelope.setAttackTime(attackTime);
    m_adsrEnvelope.setDecayTime(decayTime);
    m_adsrEnvelope.setSustainAmplitude(sustainAmplitude);
    m_adsrEnvelope.setReleaseTime(releaseTime);

    m_filter.setLowCuttoff(filterLowCuttoff);
    m_filter.setHighCuttoff(filterHighCuttoff);
    m_filter.setFilterType(filterType);

    m_oscillators[0].setActive(isOsc1Active);
    m_oscillators[0].setType(osc1Type);
    m_oscillators[0].setGain(osc1Gain);
    m_oscillators[0].setNoteOffset(osc1Offset);
    m_oscillators[1].setActive(isOsc2Active);
    m_oscillators[1].setType(osc2Type);
    m_oscillators[1].setGain(osc2Gain);
    m_oscillators[1].setNoteOffset(osc2Offset);
    m_oscillators[2].setActive(isOsc3Active);
    m_oscillators[2].setType(osc3Type);
    m_oscillators[2].setGain(osc3Gain);
    m_oscillators[2].setNoteOffset(osc3Offset);
    m_oscillators[0].setLfoActivity(lfoActive);
    m_oscillators[0].setLfoFrequency(lfoFrequency);
    for (int i = 1; i < m_oscillators.size(); i++) {
        m_oscillators[i].setLfoActivity(nullptr);
        m_oscillators[i].setLfoFrequency(nullptr);
    }
}
```

Рис. 8: Конструктор Voice до

Стало

```
15 Voice::Voice(Envelope *pEnv, Oscillator *osc1, Oscillator *osc2,
16              Oscillator *osc3, Lfo *lfo)
17 {
18     m_adsr = pEnv;
19
20     m_oscillators[0] = osc1;
21     m_oscillators[1] = osc2;
22     m_oscillators[2] = osc3;
23     m_lfo = lfo;
24 }
```

Рис. 9: Конструктор Voice після

```
25      std::array<Oscillator*, 3> pOsc;
26      for (int i = 0; i < 3; i++) {
27          pOsc[i] = new Oscillator(&m_oscType[i], &m_isOscActive[i],
28                                   &m_oscNoteOffset[i], &m_oscGain[i]);
29      }
30
31      auto pLfo = new Lfo(&m_lfoFrequency, &m_isLfoActive);
32
33
34      for (int i = 0; i < NumberOfVoices; i++) {
35          auto pEnv = new Envelope(&m_adsrParams.attackTime, &m_adsrParams.decayTime,
36                                   &m_adsrParams.sustainAmplitude, &m_adsrParams.releaseTime);
37          auto voice = new Voice(pEnv, pOsc[0], pOsc[1], pOsc[2], pLfo);
38          m_voices[i] = voice;
39      }
40
41  }
```

Рис. 10: Використання конструктора Voice після

Висновок: під час виконання роботи було проведено ознайомлення з методами рефактирингу то було проведено рефакторінг