**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**T120B516** Objektinis programų projektavimas

**Laboratorinis darbas 4**

Darbą atliko:

Ignas Lapėnas

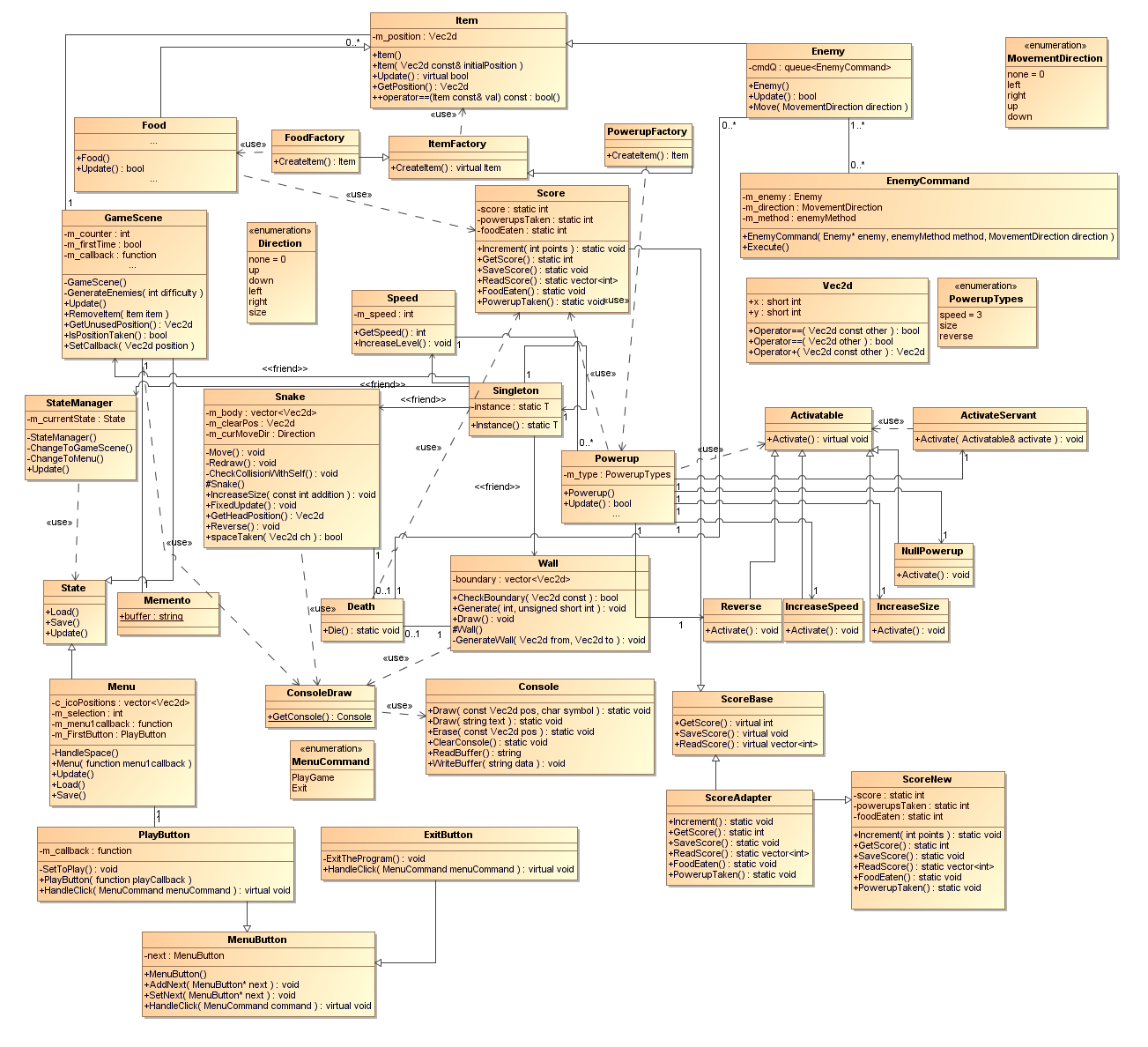
Andrius Jankauskas

KTU 2016

# Užduotis

Sukurtos programos refactoring (+2 balai): Realizuoti Adapter šabloną, panaudoti Dependency Injection principą, panaudoti NullObject šabloną. Realizuoti CahinOfResponsibility arba Visitor (max 8 balai) arba Memento (max 6 balai). Panaudoti savo pasirinktą šabloną. Šablonų veikimas turi būti pristatytas per veikiantį main() metodą

# Suprojektuota klasių diagrama:



pav. 1 Suprojektuota klasių diagrama

# Facade šablono panaudojimas

struct Console

{

private:

int m\_width, m\_height;

public:

Console(int height, int width);

void Draw(Vec2d const& pos, char const& symbol);

void Draw(std::string const& text);

void Draw(Vec2d const& pos, std::string const& text);

void Erase(Vec2d const& pos);

void ClearConsole();

std::string ReadBuffer();

void WriteBuffer(std::string data);

};

using std::cout;

static Console& GetConsole()

{

static Console instance(25,25);

return instance;

}

ConsoleDraw::Console::Console(int height, int width) : m\_height(height), m\_width(width)

{

CONSOLE\_FONT\_INFOEX info = { 0 };

info.cbSize = sizeof(info);

info.dwFontSize.X = 16;

info.dwFontSize.Y = 12; // leave X as zero

info.FontWeight = FW\_NORMAL;

wcscpy\_s(info.FaceName, L"CONSOLAS");

SetCurrentConsoleFontEx(GetStdHandle(STD\_OUTPUT\_HANDLE), NULL, &info);

MoveWindow(GetConsoleWindow(), posX, posY, windowX, windowY, TRUE);

}

void ConsoleDraw::Console::Draw(Vec2d const & pos, char const & symbol)

{

//Initialize the coordinates

COORD coord = { pos.x, pos.y }; //Static cast unable to work

//Set the position

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

cout << symbol;

}

void ConsoleDraw::Console::Draw(std::string const & text)

{

cout << text;

return;

}

void ConsoleDraw::Console::Draw(Vec2d const & pos, std::string const & text)

{

COORD coord = { pos.x,pos.y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

Draw(text);

}

void ConsoleDraw::Console::Erase(Vec2d const & pos)

{

Draw(pos, ' ');

}

void ConsoleDraw::Console::ClearConsole()

{

for (short int i = 0; i < 25; ++i)

for (short int j = 0; j < 25; ++j)

Draw({ i,j }, ' ');

}

std::string ConsoleDraw::Console::ReadBuffer()

{

PCHAR\_INFO buffer = new CHAR\_INFO[m\_height\*m\_width];

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD buffer\_size = { m\_width, m\_height };

COORD buffer\_index = { 0, 0 }; // read/write rectangle has upper-right corner at upper-right corner of buffer

SMALL\_RECT read\_rect = { 0, 0, m\_width - 1, m\_height - 1 };

ReadConsoleOutput(hStdOut, buffer, buffer\_size, buffer\_index, &read\_rect);

std::string result = [&]() {

std::string r = "";

for (int i=0; i<m\_height\*m\_width; i++)

{

r.append(1, buffer[i].Char.AsciiChar);

}

return r;

}();

delete[] buffer;

return result;

}

void ConsoleDraw::Console::WriteBuffer(std::string data)

{

PCHAR\_INFO buffer = new CHAR\_INFO[m\_height\*m\_width];

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD buffer\_size = { m\_width, m\_height };

COORD buffer\_index = { 0, 0 }; // read/write rectangle has upper-right corner at upper-right corner of buffer

SMALL\_RECT read\_rect = { 0, 0, m\_width - 1, m\_height - 1 };

for (int i = 0; i<m\_height\*m\_width; i++)

{

buffer[i].Attributes = 7; //ascii encoding

buffer[i].Char.AsciiChar = data[i];

buffer[i].Char.UnicodeChar = data[i];

}

WriteConsoleOutput(hStdOut, buffer, buffer\_size, buffer\_index, &read\_rect);

delete[] buffer;

}

# Facade šablono paskirtis

Šis šablonas leido atskirti sudentingesnes console specific funkcijos iš viso kodo. Pati console dirba su UTF8 ir Ascii koduotėmis. Mūsų programai, nereikėtu laikyti viso info apie tai. Turi užtekti standartinio string darbui su console. Jei norėsime praplėsti User interface, galėsime pakeist veikimą, nekeičiant darbo logikos.

# Chain of Responsibility šablono panaudojimas

enum class MenuCommand

{

PlayGame,

Exit

};

class MenuButton

{

MenuButton\* next;

public:

MenuButton() : next(0) { }

void AddNext(MenuButton\* next);

void SetNext(MenuButton\* next);

virtual void HandleClick(MenuCommand menuCommand);

};

class PlayButton : public MenuButton

{

private:

std::function<void()> m\_callback;

void SetToPlay();

public:

PlayButton(std::function<void()> playCallback);

virtual void HandleClick(MenuCommand menuCommand);

};

class ExitButton : public MenuButton

{

private:

void ExitTheProgram();

public:

virtual void HandleClick(MenuCommand menuCommand);

};

struct Menu : State

{

private:

const std::vector<Vec2d> c\_icoPositions = { { 5, 5 } , { 5, 7 } };

int m\_selection;

MenuNS::PlayButton m\_FirstButton;

std::function<void()> m\_menu1callback;

void HandleSpace();

public:

Menu(std::function<void()> playCallback);

void Update();

void Load();

void Save();

};

void MenuNS::MenuButton::AddNext(MenuButton \* next)

{

if (this->next)

this->next->AddNext(next);

else

SetNext(next);

}

void MenuNS::MenuButton::SetNext(MenuButton \* next)

{

this->next = next;

}

void MenuNS::MenuButton::HandleClick(MenuCommand menuCommand)

{

if (next)

next->HandleClick(menuCommand);

}

void MenuNS::ExitButton::HandleClick(MenuCommand menuCommand)

{

if (menuCommand == MenuCommand::Exit)

ExitTheProgram();

else

MenuButton::HandleClick(menuCommand);

}

void MenuNS::PlayButton::HandleClick(MenuCommand menuCommand)

{

if (MenuCommand::PlayGame == menuCommand)

SetToPlay();

else

MenuButton::HandleClick(menuCommand);

}

# Chain of Responsibility šablono paskirtis

Suteikia galimybę atskirti coupling kodą. Tereikia paveldėti MenuButton klasę ir veliau į menu užregistruojame mygtuką. Mygtukam tarpusavy nebereikia žinoti nei tikslios pozicijos, nei jį supančių mygtukų.

# Dependency Injection panaudojimas

MenuNS::PlayButton::PlayButton(std::function<void()> playCallback)

{

m\_callback = playCallback;

# }

Menu::Menu(std::function<void()> playCallback) : m\_FirstButton(MenuNS::PlayButton(playCallback))

{

//m\_menu1callback = playCallback;

//menu1callback();

static MenuNS::ExitButton exitButton;

m\_FirstButton.AddNext(&exitButton);

}

# Dependency Injection paskirtis

Šio būdo paskirtis atskirti menu nuo žaidimo. Dabar menu nereikia nei žinoti kas yra GameScene, kaip jį teisingai iškviesti ir pnš. Tiesiog paprašoma funkciją kuria suteikia mums gamescene. Šiuo atveju tai paprastas callback užnaudojimas.

# Template Method šablono panaudojimas

#pragma once

#include <vector>

class ScoreBase {

virtual int GetScore()=0;

virtual void SaveScore()=0;

virtual std::vector<int> ReadScore()=0;

};

#pragma once

#include "Settings.h"

#include "ScoreBase.h"

#include <vector>

static class Score : public ScoreBase {

private:

static int score;

static int powerupsTaken;

static int foodEaten;

public:

static void PowerupTaken();

static void FoodEaten();

static void Increment(int points);

static int GetScore();

static void SaveScore();

static std::vector<int> ReadScore();

static int GetPowerupsTaken();

static int GetFoodEaten();

};

# Template Method šablono paskirtis

Template method suteikia galimybę lengvai ir stabiliai praplėsti programos funkcionalumą pagal duotus šablonus. Šabloną panaudojau, kad pateikti bazinį šabloną taškų skaičiavimo klasei.

# Adapter šablono panaudojimas

#pragma once

#include "Score.h"

#include "ScoreNew.h"

class ScoreAdapter: public ScoreBase, public ScoreNew{

public:

static int GetScore();

static void SaveScore();

static std::vector<int> ReadScore();

static void Increment(int points);

static void FoodEaten();

static void PowerupTaken();

static int GetPowerupsTaken();

static int GetFoodEaten();

};

#include "ScoreAdapter.h"

int ScoreAdapter::GetScore()

{

return ScoreNew::GetScore();

}

void ScoreAdapter::PowerupTaken()

{

Score::PowerupTaken();

}

void ScoreAdapter::FoodEaten()

{

ScoreNew::FoodEaten();

}

void ScoreAdapter::SaveScore()

{

return ScoreNew::SaveScore();

}

std::vector<int> ScoreAdapter::ReadScore()

{

static vector<int> scores;

ifstream input;

input.open(FILE\_SCORE); //TODO check if file exists

while (input.good()) {

int tempscore;

input >> tempscore;

scores.push\_back(tempscore);

}

return scores;

}

void ScoreAdapter::Increment(int points)

{

ScoreNew::Increment(points);

}

int ScoreAdapter::GetPowerupsTaken()

{

return ScoreNew::GetPowerupsTaken();

}

int ScoreAdapter::GetFoodEaten()

{

return ScoreNew::GetFoodEaten();

}

std::string text("\n\n\nYour score is: " + std::to\_string(ScoreAdapter::GetScore()) + "\n" + "Food eaten: " + std::to\_string(ScoreAdapter::GetFoodEaten()) + "\n" + "Powerups taken: " + std::to\_string(ScoreAdapter::GetPowerupsTaken()) + "\n");

# Adapter šablono paskirtis

Adapter šablonas suteikia galimybę bendrauti 2 kitaip nesuderinamom klasėm, supaprastina valdymą tarp norimų iškviesti funckijų. Panaudojau šabloną taškų skaičiavime. Pademonstravau kaip jis veikia pakeitęs kelis Score algoritmus.

# Išvados

Fascade šablonas leido supaprastinti consoles naudojimą visoje programoje. Vėliau Fascade api gali būti užnaudojimas, lengvai keisti funkcionalumą, pridedant naują grafinę sąsają. Chain of Responsibility šablonas leidžia lengvai apjungti panašius objektus, ir atsikratomas ieškojimas kurio objekto reikia. Naujų objektų pridėjimas palengvintas. Tačiau šiek tiek sulėtina programos veikimą.

Template method šablonas leidžia pasikurti “default” klases, kurias vėliau galima paveldėti. Tai supraprastina ateities palaikymą, norint pridėti naujo funkcionalumo, panašių klasių. Adapter šablonas suteikia galimybę bendrauti 2 klasėm standartizuotu “valdymo pultu”. Interface’as leidžia uždengti metodų kvietimą.