**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**T120B516** Objektinis programų projektavimas

**Laboratorinis darbas 3**

Darbą atliko:

Ignas Lapėnas

Andrius Jankauskas

KTU 2016

# Užduotis

Realizuoti Command arba Bridge/Strategy (max įvertinimas 10) arba Facade (max įvertinimas 8). Panaudoti savo pasirinktą šabloną. Šablonų veikimas turi būti pristatytas per veikiantį main() metodąRealizacija

# E:\KTU programavimas\SnakePatterns\lab03.pngSuprojektuota klasių diagrama:

pav. 1 Suprojektuota klasių diagrama

# State šablono panaudojimas

struct State

{

virtual void Load() = 0;

virtual void Save() = 0;

virtual void Update() = 0;

};

struct Menu : State

{

private:

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

int m\_selection;

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

void HandleSpace();

public:

Menu(std::function<void()> menu1callback);

void Update();

void Load();

void Save();

};

struct GameScene : State

{

friend class Singleton<GameScene>;

GameScene();

void Load();

void Save();

public:

void SetCallback(std::function<void()> returnToMenu);

void Update();

void RemoveItem(Item\* item);

Vec2d const& GetUnusedPosition();

bool IsPositionTaken(Vec2d const& position);

};

struct StateManager

{

private:

friend struct Singleton<StateManager>;

State\* m\_currentState;

StateManager();

void ChangeToGameScene();

void ChangeToMenu();

public:

void Update();

};

# State šablono paskirtis

Šis šablonas leidžia mums valdyti programos būsenas iš vienos vietos, nekeičiant pačio kodo. Tiesiog pateikiame būsenos tipo nuorodą į iš būsenos paveldėtą tipą, ir iš jo galima kviesti pagrindines būsenos komandas.

Būsenos tampa nepriklausomos viena nuo kitos, tačiau turi būti apibrėžtos apribojančiu „interfeisu“

# Memento šablono panaudojimas

struct Memento

{

static PCHAR\_INFO buffer;

# };

Cpp faile memento dinaminio masyvo inicializavimas (vc++ 2015 neleidžia naudoti static { //code })

static int i = []() {Memento::buffer = new CHAR\_INFO[25 \* 25]; return 0; }();

void GameScene::Load()

{

ConsoleDraw::ClearConsole();

Singleton<Wall>::Instance().Draw();

if (m\_firstTime)

{

GenerateEnemies(2);

m\_firstTime = false;

}

else

{

int x = 0, y = 0;

int width = 25, height = 25;

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

COORD buffer\_size = { width, height };

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

SMALL\_RECT read\_rect = { x, y, x + width - 1, y + height - 1 };

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

}

}

void GameScene::Save()

{

int x = 0, y = 0;

int width = 25, height = 25;

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

COORD buffer\_size = { width, height };

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

SMALL\_RECT read\_rect = { x, y, x + width - 1, y + height - 1 };

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

ConsoleDraw::ClearConsole();

}

# Memento šablono paskirtis

Suteikė galimybe saugoti laikinus duomenis nesulaužant inkapsuliacijos principų. Realiai patogus statinis atminties vienetas, kuriame galime saugoti seną būseną.

# NullObject šablono panaudojimas

#pragma once

#include "Activatable.h"

class NullPowerup :

public Activatable

{

public:

void Activate() override;

NullPowerup();

~NullPowerup();

};

bool Powerup::Update()

{

if (m\_position == Singleton<Snake>::Instance().GetHeadPosition())

{

ActivateServant servant;

NullPowerup null;

Activatable\* activate = &null;

IncreaseSpeed speed;

IncreaseSize size;

Reverse reverse;

switch (m\_type)

{

case PowerupTypes::Speed:

activate = &speed;

/\*for (int i = 0; i < 5; i++)

Singleton<Speed>::Instance().IncreaseLevel();\*/

break;

case PowerupTypes::Size:

activate = &size;

//Singleton<Snake>::Instance().IncreaseSize(5);

break;

case PowerupTypes::Reverse:

activate = &reverse;

//Singleton<Snake>::Instance().Reverse();

break;

/\*default:

throw new std::exception("Something fucky was with the Powerups");\*/

}

Score::PowerupTaken();

servant.Activate(\*activate);

return true;

}

return false;

}

# NullObject šablono paskirtis

Šablonas, skirtas suteikti objektams numatytąją („default“) vertę, taip siekiant išvengti programos lūžimo atsiradus klaidai. NullObject šabloną panaudojome Powerupų („pastirpinimų“) gaudyme, kad jei switch‘as gautų ne tai ko tikimės, nesugriūtų žaidimas.

# Facade šablono panaudojimas

#pragma once

#include "Settings.h"

#include <vector>

static class Score {

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();

};

bool Powerup::Update()

{

if (m\_position == Singleton<Snake>::Instance().GetHeadPosition())

{

ActivateServant servant;

NullPowerup null;

Activatable\* activate = &null;

IncreaseSpeed speed;

IncreaseSize size;

Reverse reverse;

switch (m\_type)

{

case PowerupTypes::Speed:

activate = &speed;

/\*for (int i = 0; i < 5; i++)

Singleton<Speed>::Instance().IncreaseLevel();\*/

break;

case PowerupTypes::Size:

activate = &size;

//Singleton<Snake>::Instance().IncreaseSize(5);

break;

case PowerupTypes::Reverse:

activate = &reverse;

//Singleton<Snake>::Instance().Reverse();

break;

/\*default:

throw new std::exception("Something fucky was with the Powerups");\*/

}

Score::PowerupTaken();

servant.Activate(\*activate);

return true;

}

return false;

}

bool Food::Update()

{

if (Singleton<Snake>::Instance().GetHeadPosition() == m\_position)

{

Singleton<GameScene>::Instance().RemoveItem(this);

Singleton<Snake>::Instance().IncreaseSize(1);

Singleton<Speed>::Instance().IncreaseLevel();

Score::FoodEaten();

return true;

}

return false;

}

# Facade šablono paskirtis

Šablono paskirtis apjungti sistemos funkcionalumą į vieną patogų valdymo pultą („interface“). Panaudojom Facade šabloną realizuoti taškų skaičiavimui.

# Išvados

Skirtingi šablonai leidžia itin supaprastinti kvietimus į funkcijas gilesniuose kodo lygiuose.   
State šablonas supaprastina programos šokinėjima tarp skirting būsenų. Pačias būsenas apibrėžia aiškiai, ir verčia programuotoją teisingai pridėti naujas programos būsenas. C++ kalboje lengva memory leak’us palikti su šiuo pattern’u, nes neegzistuoja garbage collectiorius.

Memento šablonas leido saugoti būsena, nededant atminties programos aukštesniuose lygiuose. Atmintis pati nepasiekiama buvo iš kitų klasių. Reiškia vis dar išlaikėme inkapsuliacijos principą.

NullObject šablonas leidžia apsaugoti program nuo nulūžimo gavus null priskirties klaidą.

Facade šablonas yra patogus įrankis apjungti bendro tipo funckionalumui.