**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Физико-технический институт**

**Командное задание в рамках курсового проекта «Разработка игры «Pacman++»»**

**Индивидуальный отчет**

**Выполнил:**

Студент

Физико-технического Института

группы 21418

Черняк А. С.

**Проверил:**

Преподаватель:

Бульба Артем Владимирович

**Петрозаводск 2023**

# Цель работы

Разработать аналог игры «Pacman», добавив в не функционал создания призраками бомбы. Вести разработку с помощью системы контроля версий Git через веб-сервис GitHub. Закрепить полученные знания использования языков С++ с использованием библиотеки SFML и UML.

# Задачи

1. Познакомиться с принципами работы в команде.
2. Научиться использовать при работе систему контроля версии Git.
3. Научиться строить UML-диаграммы.
4. Научиться использовать SFML библиотеку в программной среде С++.
5. Написание отчёта о проделанной работе.

# Кратко о программной реализации

Программная реализация выполнена на языке C++ с использованием библиотеки SFML, среда разработки - IDE Qt Creator (Qt 5.4.2)

# Единицы компиляции:

ghost.h

ghost.cpp

Класс работает с призраками: коллизия с пакманом, захват цели и их движением.

# Описание сюжета игры

Игрок управляет пакманом, передвигаясь по карте и избегая призраков. Целью игры является избегание контакта с призраком и сбор всей еды(точек). При съедении большой точки, у призраков включается состояние “испуг”, в котором его можно скушать, за что даются очки, а сам призрак начнет свое движение со своего домика. При сборе всех точек, начинается новый уровень.

# Заголовочные файлы

## ghost.h

#ifndef GHOST\_H

#define GHOST\_H

#include "game\_types.h"

#include "pacman.h"

#include "bomb.h"

class ghost

{

public:

bool movement\_mode;

bool use\_door;

byte direction;

byte frightened;

byte frightened\_speed\_timer;

byte id;

unsigned short animation\_timer;

unsigned short bomb\_drop\_timer;

position home;

position home\_exit;

position cur\_position;

position target;

ghost(byte id);

bool pacman\_collision(const position &pacman\_position);

float get\_target\_distance(byte direction);

void draw(bool flash, sf::RenderWindow &window);

void reset(const position &home, const position &home\_exit);

void switch\_mode();

void update(byte level, game\_map &map, ghost &ghost\_0, pacman &pacman, std::vector<bomb> &bombs);

void update\_target(byte pacman\_direction, const position &ghost\_0\_position, const position &pacman\_position);

};

#endif // GHOST\_H

# Исполнительные файлы

## ghost.cpp

#include "ghost.h"

#include <array>

#include <cmath>

#include <vector>

#include "SFML/Graphics.hpp"

#include "game\_types.h"

#include "pacman.h"

#include "tools.h"

ghost::ghost(byte id) : id(id)

{

}

bool ghost::pacman\_collision(const position &pacman\_position)

{

if (cur\_position.x > pacman\_position.x - CELL\_SIZE && cur\_position.x < CELL\_SIZE + pacman\_position.x)

if (cur\_position.y > pacman\_position.y - CELL\_SIZE && cur\_position.y < CELL\_SIZE + pacman\_position.y)

return 1;

return 0;

}

float ghost::get\_target\_distance(byte direction)

{

short x = cur\_position.x;

short y = cur\_position.y;

switch (direction)

{

case 0:

x += GHOST\_SPEED;

break;

case 1:

y -= GHOST\_SPEED;

break;

case 2:

x -= GHOST\_SPEED;

break;

case 3:

y += GHOST\_SPEED;

}

return (float)(sqrt(pow(x - target.x, 2) + pow(y - target.y, 2)));

}

void ghost::draw(bool flash, sf::RenderWindow &window)

{

byte body\_frame = (byte)(floor(animation\_timer / (float)(GHOST\_ANIMATION\_SPEED)));

sf::Sprite body;

sf::Sprite face;

sf::Texture texture;

texture.loadFromFile("img/ghost" + std::to\_string(CELL\_SIZE) + ".png");

body.setTexture(texture);

body.setPosition(cur\_position.x, cur\_position.y);

body.setTextureRect(sf::IntRect(CELL\_SIZE \* body\_frame, 0, CELL\_SIZE, CELL\_SIZE));

face.setTexture(texture);

face.setPosition(cur\_position.x, cur\_position.y);

if (!frightened)

{

switch (id)

{

case ghosts::red:

body.setColor(sf::Color(255, 0, 0));

break;

case ghosts::pink:

body.setColor(sf::Color(255, 182, 255));

break;

case ghosts::cyan:

body.setColor(sf::Color(0, 255, 255));

break;

case ghosts::orange:

body.setColor(sf::Color(255, 182, 85));

}

face.setTextureRect(sf::IntRect(CELL\_SIZE \* direction, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

window.draw(body);

}

else if (frightened == frightened\_modes::frightened)

{

body.setColor(sf::Color(36, 36, 255));

face.setTextureRect(sf::IntRect(4 \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

if (flash && body\_frame % 2 == 0)

{

body.setColor(sf::Color(255, 255, 255));

face.setColor(sf::Color(255, 0, 0));

}

else

{

body.setColor(sf::Color(36, 36, 255));

face.setColor(sf::Color(255, 255, 255));

}

window.draw(body);

}

else

{

face.setTextureRect(sf::IntRect(CELL\_SIZE \* direction, 2 \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

}

window.draw(face);

animation\_timer = (1 + animation\_timer) % (GHOST\_ANIMATION\_FRAMES \* GHOST\_ANIMATION\_SPEED);

}

void ghost::reset(const position &home, const position &home\_exit)

{

movement\_mode = 0;

use\_door = 0 < id;

direction = 0;

frightened = 0;

frightened\_speed\_timer = 0;

animation\_timer = 0;

this->home = home;

this->home\_exit = home\_exit;

target = home\_exit;

}

void ghost::switch\_mode()

{

movement\_mode = 1 - movement\_mode;

}

byte flip\_direction(byte direction)

{

return (2 + direction) % 4;

}

void ghost::update(byte level, game\_map &map, ghost &ghost\_0, pacman &pacman, std::vector<bomb> &bombs)

{

bool move = 0;

byte available\_ways = 0;

byte speed = GHOST\_SPEED;

if (frightened == frightened\_modes::not\_frightened)

{

bomb\_drop\_timer++;

if (bomb\_drop\_timer > BOMB\_DROP\_INTERVAL)

{

bomb b;

b.cur\_position = cur\_position;

bombs.push\_back(b);

bomb\_drop\_timer = 0;

}

}

std::array<bool, 4> walls{};

if (frightened == frightened\_modes::not\_frightened && pacman.energizer\_timer == ENERGIZER\_DURATION / pow(2, level))

{

frightened\_speed\_timer = GHOST\_FRIGHTENED\_SPEED;

frightened = frightened\_modes::frightened;

}

else if (!pacman.energizer\_timer && frightened == frightened\_modes::frightened)

{

frightened = frightened\_modes::not\_frightened;

}

if (frightened == frightened\_modes::very\_frightened && (cur\_position.x % GHOST\_ESCAPE\_SPEED) == 0 && (cur\_position.y % GHOST\_ESCAPE\_SPEED) == 0)

speed = GHOST\_ESCAPE\_SPEED;

update\_target(pacman.direction, ghost\_0.cur\_position, pacman.cur\_position);

walls[0] = map\_collision(use\_door, {speed + cur\_position.x, cur\_position.y}, map);

walls[1] = map\_collision(use\_door, {cur\_position.x, cur\_position.y - speed}, map);

walls[2] = map\_collision(use\_door, {cur\_position.x - speed, cur\_position.y}, map);

walls[3] = map\_collision(use\_door, {cur\_position.x, speed + cur\_position.y}, map);

if (frightened != frightened\_modes::frightened)

{

byte optimal\_direction = directions::no\_direction;

move = 1;

for (byte a = 0; a < 4; a++)

{

if (a == flip\_direction(direction))

continue;

if (!walls[a])

{

if (optimal\_direction == directions::no\_direction)

optimal\_direction = a;

available\_ways++;

if (get\_target\_distance(a) < get\_target\_distance(optimal\_direction))

optimal\_direction = a;

}

}

if (available\_ways > 1)

direction = optimal\_direction;

else

{

if (optimal\_direction == directions::no\_direction)

direction = flip\_direction(direction);

else

direction = optimal\_direction;

}

}

else

{

byte random\_direction = rand() % 4;

if (!frightened\_speed\_timer)

{

move = 1;

frightened\_speed\_timer = GHOST\_FRIGHTENED\_SPEED;

for (byte a = 0; a < 4; a++)

{

if (a == flip\_direction(direction))

continue;

if (!walls[a])

available\_ways++;

}

if (0 < available\_ways)

{

while (walls[random\_direction] || random\_direction == flip\_direction(direction))

random\_direction = rand() % 4;

direction = random\_direction;

}

else

direction = flip\_direction(direction);

}

else

frightened\_speed\_timer--;

}

if (move)

{

switch (direction)

{

case directions::right:

cur\_position.x += speed;

break;

case directions::up:

cur\_position.y -= speed;

break;

case directions::left:

cur\_position.x -= speed;

break;

case directions::down:

cur\_position.y += speed;

}

if (-CELL\_SIZE >= cur\_position.x)

cur\_position.x = CELL\_SIZE \* MAP\_WIDTH - speed;

else if (cur\_position.x >= CELL\_SIZE \* MAP\_WIDTH)

cur\_position.x = speed - CELL\_SIZE;

}

if (pacman\_collision(pacman.cur\_position))

{

if (!frightened)

{

pacman.is\_dead = true;

}

else

{

use\_door = true;

frightened = frightened\_modes::very\_frightened;

target = home;

pacman.score.add(POINTS\_FOR\_GHOST, level);

}

}

}

void ghost::update\_target(byte pacman\_direction, const position &ghost\_0\_position, const position &pacman\_position)

{

if (use\_door)

{

if (cur\_position == target)

{

if (home\_exit == target)

{

use\_door = 0;

}

else if (home == target)

{

frightened = frightened\_modes::not\_frightened;

target = home\_exit;

}

}

}

else

{

if (!movement\_mode)

{

switch (id)

{

case ghosts::red:

target = {CELL\_SIZE \* (MAP\_WIDTH - 1), 0};

break;

case ghosts::pink:

target = {0, 0};

break;

case ghosts::cyan:

target = {CELL\_SIZE \* (MAP\_WIDTH - 1), CELL\_SIZE \* (MAP\_HEIGHT - 1)};

break;

case ghosts::orange:

target = {0, CELL\_SIZE \* (MAP\_HEIGHT - 1)};

}

}

else

{

switch (id)

{

case ghosts::red:

{

target = pacman\_position;

break;

}

case ghosts::pink:

{

target = pacman\_position;

switch (pacman\_direction)

{

case directions::right:

target.x += CELL\_SIZE \* GHOST\_1\_CHASE;

break;

case directions::up:

target.y -= CELL\_SIZE \* GHOST\_1\_CHASE;

break;

case directions::left:

target.x -= CELL\_SIZE \* GHOST\_1\_CHASE;

break;

case directions::down:

target.y += CELL\_SIZE \* GHOST\_1\_CHASE;

}

break;

}

case ghosts::cyan:

{

target = pacman\_position;

switch (pacman\_direction)

{

case 0:

target.x += CELL\_SIZE \* GHOST\_2\_CHASE;

break;

case 1:

target.y -= CELL\_SIZE \* GHOST\_2\_CHASE;

break;

case 2:

target.x -= CELL\_SIZE \* GHOST\_2\_CHASE;

break;

case 3:

target.y += CELL\_SIZE \* GHOST\_2\_CHASE;

}

target.x += target.x - ghost\_0\_position.x;

target.y += target.y - ghost\_0\_position.y;

break;

}

case ghosts::orange:

{

if (CELL\_SIZE \* GHOST\_3\_CHASE <= sqrt(pow(cur\_position.x - pacman\_position.x, 2) + pow(cur\_position.y - pacman\_position.y, 2)))

target = pacman\_position;

else

target = {0, CELL\_SIZE \* (MAP\_HEIGHT - 1)};

}

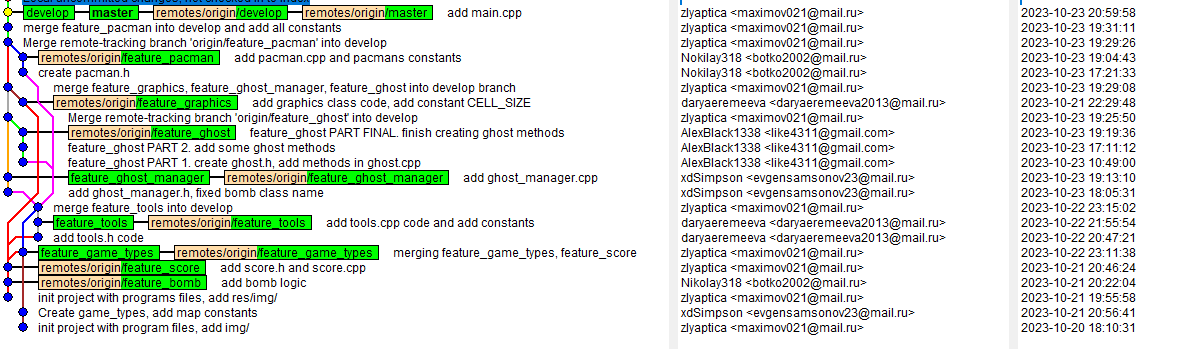
}

}

}

}

# История проекта Github



**Мои коммиты:**

* + - 1. feature\_ghost PART 1. create ghost.h, add methods in ghost.cpp

Создал заголовочный файл "ghost.h", описывающий класс ghost, добавил скелеты методов в "ghost.cpp".

* + - 1. feature\_ghost PART 2. add some ghost methods

Добавил некоторые методы: pacman\_collision(), get\_target\_distance(), switch\_mode(), flip\_direction().

* + - 1. feature\_ghost PART FINAL. finish creating ghost methods.

Добавил методы: draw(), reset(), update(), update\_target(); завершил работу над методами.

# Вывод

В ходе выполнения командного задания в рамках курсового проекта использовалась среда разработки Qt язык C++ с использованием библиотеки SFML. Применялась система контроля версий git. Реализованы все прецеденты. В ходе тестирования игры зависания или сбои не были обнаружены. В алгоритмы методов старались не добавлять не нужных циклов, массивов и др. Был разработан класс ghost, который содержит в себе реализацию движения призраков, их поведения и взаимодействие с пакманом.