**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++»»**

**Выполнили:**

Студенты

Физико-технического

Института группы 21418

Красавцев М.Ю

Ботько Н.Д.

Еремеева Д.Н.

Самсонов Е.А.

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:

constants.h

Файл хранит константы, используемые в проекте.

bomb.h

bomb.cpp

Класс описывает сущность бомбы и ее поведение.

game\_types.h

game\_types.cpp

Класс содержит перечисляемые типа и структуру, используемые в проекте, например, для хранения позиции.

ghost.h

ghost.cpp

Описание класса призрака. Содержит в себе поведение призрака

ghost\_manager.h

ghost\_manager.cpp

Управляет поведением всех призраков.

graphics.h

graphics.cpp

Класс отвечает за отрисовку карты и текста.

pacman.h

pacman.cpp

Класс описывает сущность пакмана и его поведение.

score.h

score.cpp

Класс работает с игровым счетом: сохранение, загрузка и обновление в ходе игры.

tools.h

tools.cpp

Класс содержит инструменты, такие как: установление соответствия между картой и тем, что нарисовано в единице карты(в т.ч. начальные позиции призраков, пакмана, еды), проверка на коллизию и подсчет координат для нее, а также логику сбора еды.

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.

# Диаграмма вариантов использования

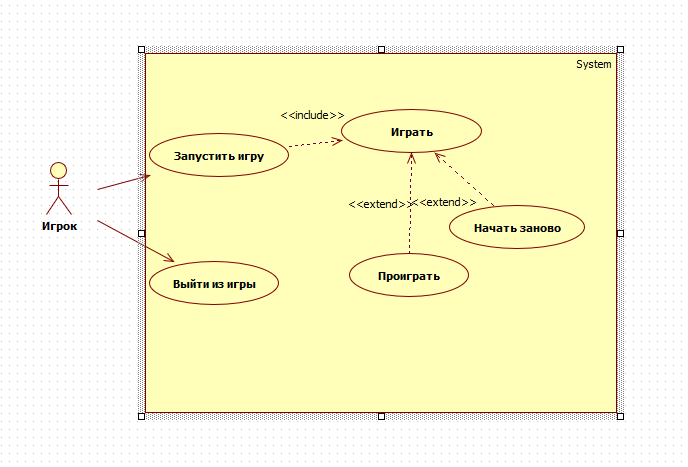


Рис 1. Диаграмма вариантов использования

Запустить игру

Игрок запускает игру

Играть

С помощью стрелок игрок управляет пакманов, собирая точки и уничтожает призраков.

Выйти из игры

Игрок выходит из игры

Проиграть

Если призрак поймал игрока, то игра прекращается и записывается его счет.

Начать заново

При желании, игрок может начать игру заново сбросив свой счет и восстановив убитых врагов

# 

# Список существительных

* Пакман

Класс, представляющий игрока. Он отвечает за передвижение, а также управляет таймерами.

* Призрак

Класс, описывающий поведение отдельного призрака, то есть его передвижение, отрисовка и работу бомбы(когда и где ей появляться).

* Группа призраков

Класс, отвечающий за совместное поведение призраков, движение, сброс данных о каждом призраке

* Бомба

Класс, описывающий поведение бомбы. Класс отвечает за проверку на пересечение с пакманом.

* Счет

Класс работает со игровым счетом: записывает в течение игры, сохраняет при завершении игры и загружает в начале.

* Инструменты

Класс хранит в себе второстепенные функции, такие как: установление соответствия между картой и тем, что нарисовано в единице карты(в т.ч. начальные позиции призраков, пакмана, еды), проверка на коллизию и подсчет координат для нее, а также логику сбора еды.

# Диаграмма классов

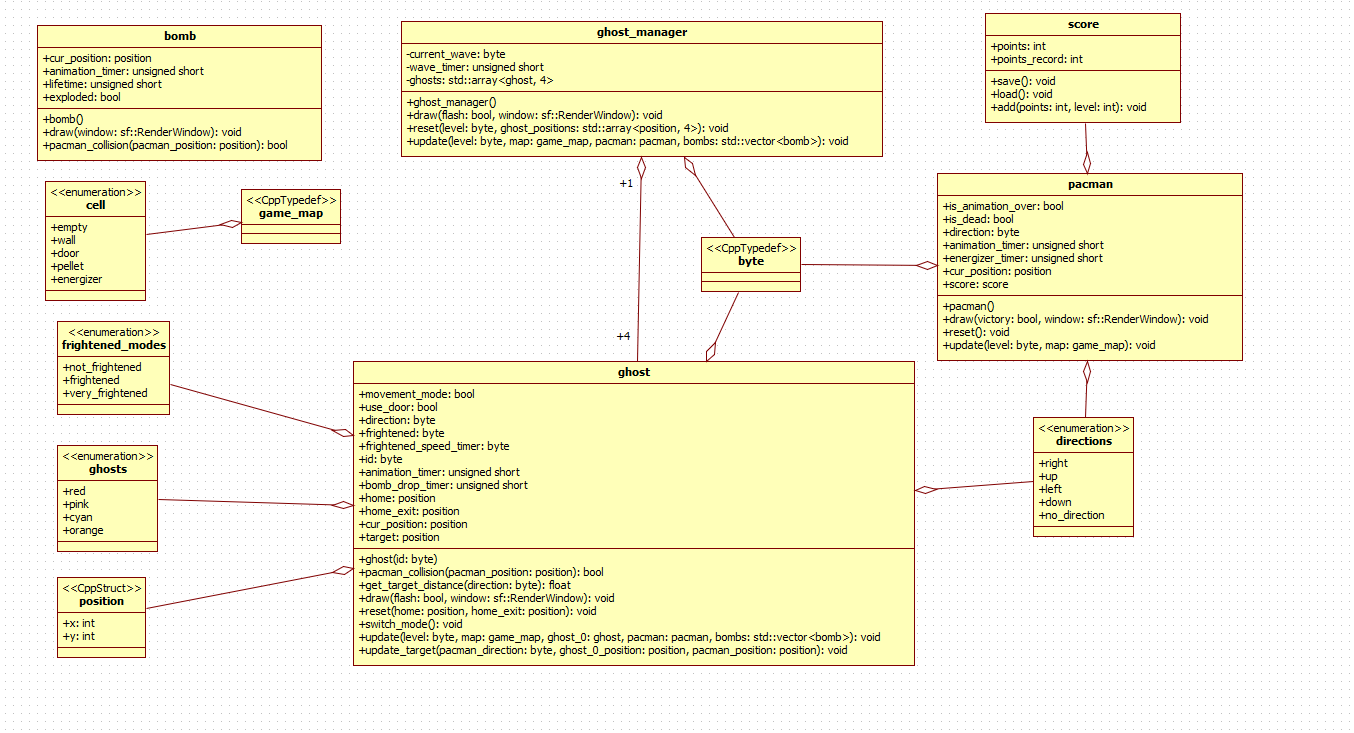


Рис2. Диаграмма классов

# Заголовочные файлы

## bomb.h

#ifndef BOMB\_H

#define BOMB\_H

#include "game\_types.h"

class bomb

{

public:

position cur\_position;

unsigned short animation\_timer;

unsigned short lifetime;

bool exploded;

bomb();

void draw(sf::RenderWindow &window);

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

};

#endif // BOMB\_H

## constants.h

#ifndef CONSTANTS

#define CONSTANTS

#define CELL\_SIZE 16

#define FONT\_HEIGHT 16

#define GHOST\_1\_CHASE 2

#define GHOST\_2\_CHASE 1

#define GHOST\_3\_CHASE 4

#define TOTAL\_GHOSTS 4

#define GHOST\_ANIMATION\_FRAMES 6

#define GHOST\_ANIMATION\_SPEED 4

#define GHOST\_ESCAPE\_SPEED 4

#define GHOST\_FRIGHTENED\_SPEED 3

#define GHOST\_SPEED 1

#define MAP\_HEIGHT 21

#define MAP\_WIDTH 21

#define PACMAN\_ANIMATION\_FRAMES 6

#define PACMAN\_ANIMATION\_SPEED 4

#define PACMAN\_DEATH\_FRAMES 12

#define PACMAN\_SPEED 2

#define SCREEN\_RESIZE 2

#define CHASE\_DURATION 1024

#define ENERGIZER\_DURATION 512

#define FRAME\_DURATION 16667

#define GHOST\_FLASH\_START 64

#define LONG\_SCATTER\_DURATION 512

#define SHORT\_SCATTER\_DURATION 256

#define POINTS\_FOR\_PELLET 25

#define POINTS\_FOR\_GHOST 200

#define BOMB\_ANIMATION\_FRAMES 2

#define BOMB\_ANIMATION\_SPEED 4

#define BOMB\_BOOM\_FRAME 3

#define BOMB\_DROP\_INTERVAL 256

#define BOMB\_MAX\_LIFETIME 128

#endif // CONSTANTS

## game\_types.h

#ifndef GAME\_TYPES\_H

#define GAME\_TYPES\_H

#include "constants.h"

#include "SFML/Graphics.hpp"

#include <array>

typedef unsigned char byte;

enum cell

{

empty,

wall,

door,

pellet,

energizer

};

enum directions

{

right,

up,

left,

down,

no\_direction

};

enum frightened\_modes

{

not\_frightened,

frightened,

very\_frightened

};

enum ghosts

{

red,

pink,

cyan,

orange

};

struct position

{

int x;

int y;

};

typedef std::array<std::array<cell, MAP\_HEIGHT>, MAP\_WIDTH> game\_map;

bool operator==(position &a, position &b);

#endif // GAME\_TYPES\_H

## 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

## ghost\_manager.h

#ifndef GHOST\_MANAGER\_H

#define GHOST\_MANAGER\_H

#include "ghost.h"

#include <array>

class ghost\_manager

{

byte current\_wave;

unsigned short wave\_timer;

std::array<ghost, 4> ghosts;

public:

ghost\_manager();

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

void reset(byte level, const std::array<position, 4> &ghost\_positions);

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

};

#endif // GHOST\_MANAGER\_H

## graphics.h

#ifndef GRAPHICS\_H

#define GRAPHICS\_H

#include "game\_types.h"

#include "pacman.h"

void draw\_map(game\_map &map, sf::RenderWindow &window);

void draw\_text(bool center, position pos, std::string text, sf::RenderWindow &window);

#endif // GRAPHICS\_H

## pacman.h

#ifndef PACMAN\_H

#define PACMAN\_H

#include "game\_types.h"

#include "score.h"

class pacman

{

public:

bool is\_animation\_over;

bool is\_dead;

byte direction;

unsigned short animation\_timer;

unsigned short energizer\_timer;

position cur\_position;

score score;

pacman();

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

void reset();

void update(byte level, game\_map &map);

};

#endif // PACMAN\_H

## score.h

#ifndef SCORE\_H

#define SCORE\_H

#include "game\_types.h"

class score

{

public:

int points;

int points\_record;

void save();

void load();

void add(int points, int level);

};

#endif // SCORE\_H

## tools.h

#ifndef TOOLS\_H

#define TOOLS\_H

#include "game\_types.h"

#include "pacman.h"

#include "score.h"

game\_map make\_game\_map(const std::array<std::string, MAP\_HEIGHT> &map\_from\_strings, std::array<position, 4> &ghost\_positions, pacman &pacman);

void free\_map(game\_map p);

bool map\_collision(bool use\_door, position point, game\_map& map);

bool collect\_pellets(position point, game\_map &map, score &score, int level);

#endif // TOOLS\_H

# Исполнительные файлы

## bomb.cpp

#include "bomb.h"

#include <math.h>

bomb::bomb() : cur\_position({0, 0}),

animation\_timer(0),

lifetime(0),

exploded(0)

{

}

void bomb::draw(sf::RenderWindow &window)

{

byte frame = (byte)(floor(animation\_timer / (float)(BOMB\_ANIMATION\_SPEED)));

sf::Sprite sprite;

sf::Texture texture;

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

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

sprite.setTexture(texture);

lifetime++;

if (lifetime > BOMB\_MAX\_LIFETIME)

{

sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* (BOMB\_BOOM\_FRAME - 1), 0, CELL\_SIZE, CELL\_SIZE));

exploded = true;

}

else

{

sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* frame, 0, CELL\_SIZE, CELL\_SIZE));

}

window.draw(sprite);

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

}

bool bomb::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;

}

## game\_types.cpp

#include "game\_types.h"

bool operator==(position &a, position &b)

{

return a.x == b.x && a.y == b.y;

}

## 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)};

}

}

}

}

}

## ghost\_manager.cpp

#include "ghost\_manager.h"

#include <math.h>

#include "SFML/Graphics.hpp"

#include "pacman.h"

#include "ghost\_manager.h"

ghost\_manager::ghost\_manager() : current\_wave(0),

wave\_timer(LONG\_SCATTER\_DURATION),

ghosts({ghost(0), ghost(1), ghost(2), ghost(3)})

{

}

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

{

for (ghost &ghost : ghosts)

ghost.draw(flash, window);

}

void ghost\_manager::reset(byte level, const std::array<position, 4> &ghost\_positions)

{

current\_wave = 0;

wave\_timer = (unsigned short)(LONG\_SCATTER\_DURATION / pow(2, level));

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

ghosts[a].cur\_position = {ghost\_positions[a].x, ghost\_positions[a].y};

for (ghost &ghost : ghosts)

ghost.reset(ghosts[2].cur\_position, ghosts[0].cur\_position);

}

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

{

if (!pacman.energizer\_timer)

{

if (!wave\_timer)

{

if (7 > current\_wave)

{

current\_wave++;

for (ghost &ghost : ghosts)

ghost.switch\_mode();

}

if (1 == current\_wave % 2)

wave\_timer = CHASE\_DURATION;

else if (2 == current\_wave)

wave\_timer = (unsigned short)(LONG\_SCATTER\_DURATION / pow(2, level));

else

wave\_timer = (unsigned short)(SHORT\_SCATTER\_DURATION / pow(2, level));

}

else

wave\_timer--;

}

for (ghost &ghost : ghosts)

ghost.update(level, map, ghosts[0], pacman, bombs);

}

## graphics.cpp

#include "graphics.h"

#include "SFML/Graphics.hpp"

#include <math.h>

void draw\_map(game\_map &map, sf::RenderWindow &window)

{

sf::Sprite sprite;

sf::Texture texture;

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

sprite.setTexture(texture);

for (int a = 0; a < MAP\_WIDTH; a++)

{

for (int b = 0; b < MAP\_HEIGHT; b++)

{

sprite.setPosition((float)(CELL\_SIZE \* a), (float)(CELL\_SIZE \* b));

switch (map[a][b])

{

case cell::empty:

break;

case cell::door:

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

window.draw(sprite);

break;

case cell::energizer:

sprite.setTextureRect(sf::IntRect(CELL\_SIZE, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

window.draw(sprite);

break;

case cell::pellet:

sprite.setTextureRect(sf::IntRect(0, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

window.draw(sprite);

break;

case cell::wall:

bool down = 0;

bool left = 0;

bool right = 0;

bool up = 0;

if (b < MAP\_HEIGHT - 1)

if (map[a][1 + b] == cell::wall)

down = 1;

if (a > 0)

{

if (map[a - 1][b] == cell::wall)

left = 1;

}

else

left = 1;

if (a < MAP\_WIDTH - 1)

{

if (map[1 + a][b] == cell::wall)

right = 1;

}

else

right = 1;

if (b > 0)

if (map[a][b - 1] == cell::wall)

up = 1;

sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* (down + 2 \* (left + 2 \* (right + 2 \* up))), 0, CELL\_SIZE, CELL\_SIZE));

window.draw(sprite);

}

}

}

}

void draw\_text(bool center, position pos, std::string text, sf::RenderWindow &window)

{

short character\_x = pos.x;

short character\_y = pos.y;

byte character\_width;

sf::Sprite character\_sprite;

sf::Texture font\_texture;

font\_texture.loadFromFile("img/font.png");

character\_width = font\_texture.getSize().x / 96;

character\_sprite.setTexture(font\_texture);

if (center)

{

character\_x = (short)(round(0.5f \* (CELL\_SIZE \* MAP\_WIDTH - character\_width \* text.substr(0, text.find\_first\_of('\n')).size())));

character\_y = (short)(round(0.5f \* (CELL\_SIZE \* MAP\_HEIGHT - FONT\_HEIGHT \* (1 + std::count(text.begin(), text.end(), '\n')))));

}

for (std::string::const\_iterator a = text.begin(); a != text.end(); a++)

{

if ('\n' == \*a)

{

if (center)

character\_x = (short)(round(0.5f \* (CELL\_SIZE \* MAP\_WIDTH - character\_width \* text.substr(1 + a - text.begin(), text.find\_first\_of('\n', 1 + a - text.begin()) - (1 + a - text.begin())).size())));

else

character\_x = pos.x;

character\_y += FONT\_HEIGHT;

continue;

}

character\_sprite.setPosition(character\_x, character\_y);

character\_sprite.setTextureRect(sf::IntRect(character\_width \* (\*a - 32), 0, character\_width, FONT\_HEIGHT));

character\_x += character\_width;

window.draw(character\_sprite);

}

}

## main.cpp

#include <array>

#include <vector>

#include <chrono>

#include <ctime>

#include "SFML/Graphics.hpp"

#include "game\_types.h"

#include "ghost\_manager.h"

#include "tools.h"

#include "graphics.h"

int main()

{

bool game\_won = 0;

unsigned lag = 0;

byte level = 0;

std::chrono::time\_point<std::chrono::steady\_clock> previous\_time;

std::array<std::string, MAP\_HEIGHT> map\_sketch = {

" ################### ",

" #........#........# ",

" #o##.###.#.###.##o# ",

" #.................# ",

" #.##.#.#####.#.##.# ",

" #....#...#...#....# ",

" ####.### # ###.#### ",

" #.# 0 #.# ",

"#####.# ##=## #.#####",

" . #123# . ",

"#####.# ##### #.#####",

" #.# #.# ",

" ####.# ##### #.#### ",

" #........#........# ",

" #.##.###.#.###.##.# ",

" #o.#.....P.....#.o# ",

" ##.#.#.#####.#.#.## ",

" #....#...#...#....# ",

" #.######.#.######.# ",

" #.................# ",

" ################### "};

game\_map map;

std::array<position, 4> ghost\_positions;

std::vector<bomb> bombs;

sf::Event event;

sf::RenderWindow window(sf::VideoMode(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE), "Pac-Man", sf::Style::Close);

window.setView(sf::View(sf::FloatRect(0, 0, CELL\_SIZE \* MAP\_WIDTH, FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT)));

ghost\_manager ghost\_manager;

pacman pacman;

srand(time(0));

map = make\_game\_map(map\_sketch, ghost\_positions, pacman);

ghost\_manager.reset(level, ghost\_positions);

previous\_time = std::chrono::steady\_clock::now();

pacman.score.load();

while (window.isOpen())

{

unsigned delta\_time = std::chrono::duration\_cast<std::chrono::microseconds>(std::chrono::steady\_clock::now() - previous\_time).count();

lag += delta\_time;

previous\_time += std::chrono::microseconds(delta\_time);

while (FRAME\_DURATION <= lag)

{

lag -= FRAME\_DURATION;

while (window.pollEvent(event))

{

switch (event.type)

{

case sf::Event::Closed:

pacman.score.save();

window.*close*();

default:

break;

}

}

for (long unsigned int i = 0; i < bombs.size(); i++)

if (bombs[i].exploded)

bombs.erase(bombs.begin() + i);

for (bomb &b : bombs)

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

pacman.is\_dead = true;

if (!game\_won && !pacman.is\_dead)

{

game\_won = 1;

pacman.update(level, map);

ghost\_manager.update(level, map, pacman, bombs);

for (int i = 0; i < MAP\_WIDTH; i++)

{

for (int j = 0; j < MAP\_HEIGHT; j++)

if (map[i][j] == cell::pellet)

{

game\_won = 0;

break;

}

if (!game\_won)

break;

}

if (game\_won)

pacman.animation\_timer = 0;

}

else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Return))

{

game\_won = 0;

if (pacman.is\_dead)

{

level = 0;

pacman.score.points = 0;

pacman.score.save();

}

else

level++;

map = make\_game\_map(map\_sketch, ghost\_positions, pacman);

ghost\_manager.reset(level, ghost\_positions);

pacman.reset();

bombs.clear();

}

if (FRAME\_DURATION > lag)

{

window.clear();

if (!game\_won && !pacman.is\_dead)

{

draw\_map(map, window);

for (bomb &b : bombs)

b.draw(window);

ghost\_manager.draw(GHOST\_FLASH\_START >= pacman.energizer\_timer, window);

draw\_text(0, {0, CELL\_SIZE \* MAP\_HEIGHT}, "Level: " + std::to\_string(1 + level), window);

draw\_text(0, {CELL\_SIZE \* MAP\_WIDTH - CELL\_SIZE \* 16, CELL\_SIZE \* MAP\_HEIGHT}, "Score: " + std::to\_string(pacman.score.points), window);

draw\_text(0, {CELL\_SIZE \* MAP\_WIDTH - CELL\_SIZE \* 8, CELL\_SIZE \* MAP\_HEIGHT}, "Record: " + std::to\_string(pacman.score.points\_record), window);

}

pacman.draw(game\_won, window);

if (pacman.is\_animation\_over)

{

if (game\_won)

draw\_text(1, {0, 0}, "Next level!", window);

else

draw\_text(1, {0, 0}, "Game over!", window);

}

window.display();

}

}

}

}

## pacman.cpp

#include "pacman.h"

#include <array>

#include <cmath>

#include "SFML/Graphics.hpp"

#include "game\_types.h"

#include "tools.h"

pacman::pacman() : is\_animation\_over(0),

is\_dead(0),

direction(0),

energizer\_timer(0),

cur\_position({0, 0}),

score({0,0})

{

}

void pacman::draw(bool is\_victory, sf::RenderWindow &window)

{

byte frame = (byte)(floor(animation\_timer / (float)(PACMAN\_ANIMATION\_SPEED)));

sf::Sprite sprite;

sf::Texture texture;

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

if (is\_dead || is\_victory)

{

if (animation\_timer < PACMAN\_DEATH\_FRAMES \* PACMAN\_ANIMATION\_SPEED)

{

animation\_timer++;

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

sprite.setTexture(texture);

sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* frame, 0, CELL\_SIZE, CELL\_SIZE));

window.draw(sprite);

}

else

{

is\_animation\_over = true;

}

}

else

{

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

sprite.setTexture(texture);

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

window.draw(sprite);

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

}

}

void pacman::reset()

{

is\_animation\_over = 0;

is\_dead = 0;

direction = 0;

animation\_timer = 0;

energizer\_timer = 0;

}

void pacman::update(byte level, game\_map &map)

{

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

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

walls[1] = map\_collision(0, {cur\_position.x, cur\_position.y - PACMAN\_SPEED}, map);

walls[2] = map\_collision(0, {cur\_position.x - PACMAN\_SPEED, cur\_position.y}, map);

walls[3] = map\_collision(0, {cur\_position.x, PACMAN\_SPEED + cur\_position.y}, map);

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))

if (!walls[directions::right])

direction = directions::right;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))

if (!walls[directions::up])

direction = directions::up;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))

if (!walls[directions::left])

direction = directions::left;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))

if (!walls[directions::down])

direction = directions::down;

if (!walls[direction])

switch (direction)

{

case directions::right:

cur\_position.x += PACMAN\_SPEED;

break;

case directions::up:

cur\_position.y -= PACMAN\_SPEED;

break;

case directions::left:

cur\_position.x -= PACMAN\_SPEED;

break;

case directions::down:

cur\_position.y += PACMAN\_SPEED;

}

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

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

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

cur\_position.x = PACMAN\_SPEED - CELL\_SIZE;

if (collect\_pellets(cur\_position, map, score, level))

energizer\_timer = (unsigned short)(ENERGIZER\_DURATION / pow(2, level));

else

energizer\_timer = std::max(0, energizer\_timer - 1);

}

## score.cpp

#include "score.h"

#include <fstream>

void score::save()

{

std::ofstream file("max\_score.txt", std::ofstream::out);

file << points\_record;

file.close();

}

void score::load()

{

std::ifstream file("max\_score.txt", std::ifstream::in);

if (!file.is\_open())

return;

file >> points\_record;

file.close();

}

void score::add(int points, int level)

{

this->points += points \* (level + 1);

if (this->points > points\_record)

points\_record = this->points;

}

## 

## tools.cpp

#include "tools.h"

#include <math.h>

#include <array>

#include <string>

#include "SFML/Graphics.hpp"

game\_map make\_game\_map(const std::array<std::string, MAP\_HEIGHT> &map\_sketch, std::array<position, 4> &ghost\_positions, pacman &pacman)

{

game\_map output\_map;

for (byte a = 0; a < MAP\_HEIGHT; a++)

{

for (byte b = 0; b < MAP\_WIDTH; b++)

{

output\_map[b][a] = cell::empty;

switch (map\_sketch[a][b])

{

case '#':

output\_map[b][a] = cell::wall;

break;

case '=':

output\_map[b][a] = cell::door;

break;

case '.':

output\_map[b][a] = cell::pellet;

break;

case '0':

ghost\_positions[0].x = CELL\_SIZE \* b;

ghost\_positions[0].y = CELL\_SIZE \* a;

break;

case '1':

ghost\_positions[1].x = CELL\_SIZE \* b;

ghost\_positions[1].y = CELL\_SIZE \* a;

break;

case '2':

ghost\_positions[2].x = CELL\_SIZE \* b;

ghost\_positions[2].y = CELL\_SIZE \* a;

break;

case '3':

ghost\_positions[3].x = CELL\_SIZE \* b;

ghost\_positions[3].y = CELL\_SIZE \* a;

break;

case 'P':

pacman.cur\_position = {CELL\_SIZE \* b, CELL\_SIZE \* a};

break;

case 'o':

output\_map[b][a] = cell::energizer;

}

}

}

return output\_map;

}

std::array<position, 4> calculate\_map\_coordinates(position point, game\_map &map)

{

std::array<position, 4> output;

float cell\_x = point.x / (float)(CELL\_SIZE);

float cell\_y = point.y / (float)(CELL\_SIZE);

output[0] = {(int)floor(cell\_x), (int)floor(cell\_y)};

output[1] = {(int)ceil(cell\_x), (int)floor(cell\_y)};

output[2] = {(int)floor(cell\_x), (int)ceil(cell\_y)};

output[3] = {(int)ceil(cell\_x), (int)ceil(cell\_y)};

return output;

}

bool map\_collision(bool use\_door, position point, game\_map &map)

{

bool output = 0;

std::array<position, 4> corners = calculate\_map\_coordinates(point, map);

for (position &corner : corners)

{

int x = corner.x;

int y = corner.y;

if (0 <= x && 0 <= y && MAP\_HEIGHT > y && MAP\_WIDTH > x)

{

if (map[x][y] == cell::wall)

output = 1;

else if (!use\_door && map[x][y] == cell::door)

output = 1;

}

}

return output;

}

bool collect\_pellets(position point, game\_map &map, score &score, int level)

{

bool output = 0;

std::array<position, 4> corners = calculate\_map\_coordinates(point, map);

for (position &corner : corners)

{

int x = corner.x;

int y = corner.y;

if (map[x][y] == cell::energizer)

{

output = 1;

map[x][y] = cell::empty;

}

else if (map[x][y] == cell::pellet)

{

map[x][y] = cell::empty;

score.add(POINTS\_FOR\_PELLET, level);

}

}

return output;

}

# Руководство пользователя

Запустить программу (игру).

Появится игровое поле, где вы управляете пакменом ( рис. 3, желтый круг) при помощи стрелок. Для прохождения уровня необходимо собрать все точки.

Призраки в это время будут пытаться догнать вас, если хотя бы один из них достигнет игрока - игра будет окончена.

Вы можете вернуть призраков в домик, если игрок подберет большую точку, то при соприкосновении с призраком, тот убежит домой, в центр карты, куда пакману проход запрещен.

Если вы проиграли, чтобы начать игру, снова нажмите ЭНТЕР.

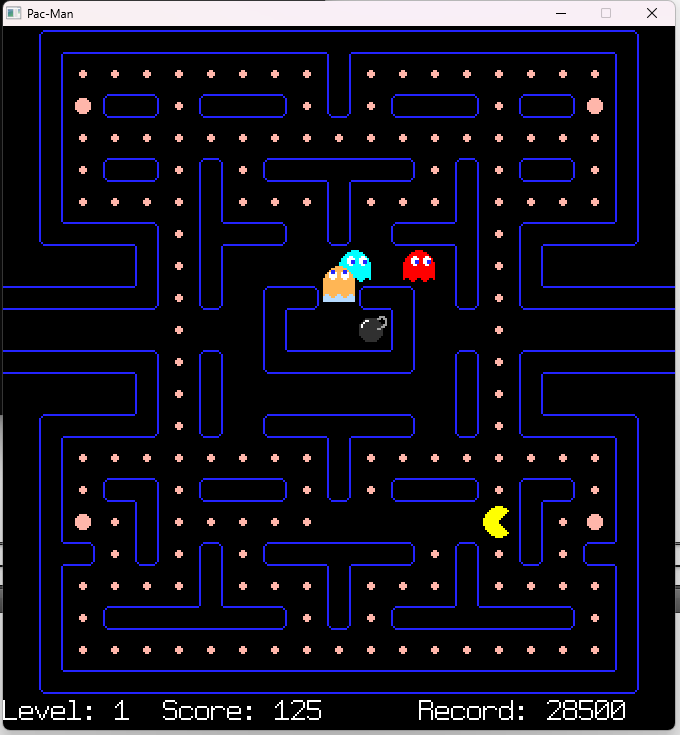
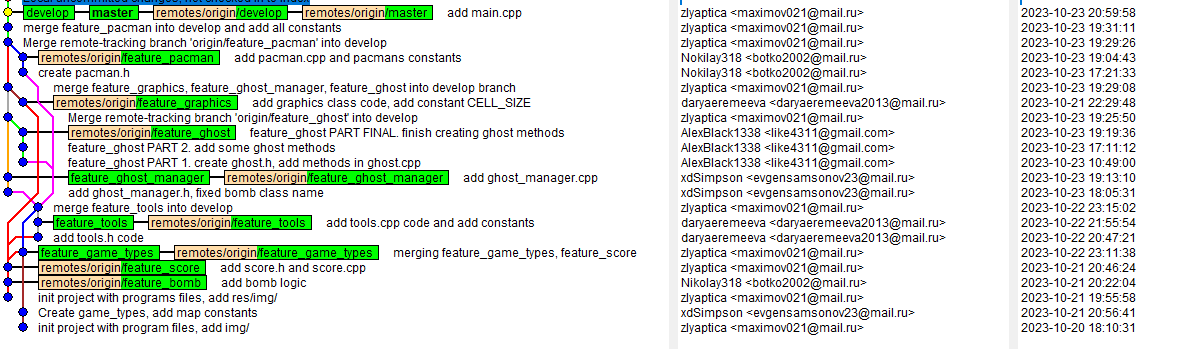


Рис 3. Запущенная игра.

# История проекта Github



**Красавцев Максим:**

"init project with program files, add img/"

Инициализация проекта, создание всех файлов, которые необходимо для работы игры. Добавлена папка с изображениями.

**Самсонов Евгений:**

“Create game\_types, add map constants”.

Создана реализация класса game\_types, добавлены константы размеров карты.

**Ботько Николай:**

“add bomb logic”.

Добавлена логика класса бомба.

**Красавцев Максим:**

“add score.h and score.cpp”.

Добавлен код для файлов score.h иscore.cpp.

**Красавцев Максим:**

“merging feature\_game\_types, feature\_score”.

Объединение веток feature\_game\_types, feature\_score с develop.

**Еремеева Дарья:**

“add tools.h code”.

Добавлен код для файла tools.h.

**Еремеева Дарья:**

“add tools.cpp code and add constants”.

Добавлена реализация файла tools.cpp и добавлены константы.

**Красавцев Максим:**

“merge feature\_tools into develop”.

Объединение ветки feature\_tools с develop.

**Самсонов Евгений:**

“add ghost\_manager.h, fixed bomb class name”.

Добавлен код для файла ghost\_manager.h, исправлено название класса bomb в pacman.pro.

**Самсонов Евгений:**

“add ghost\_manager.cpp”.

Добавлен код для файла ghost\_manager.cpp.

**Черняк Александр**

“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".

**Черняк Александр**

“feature\_ghost PART 2. add some ghost methods”.

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

**Черняк Александр**

“feature\_ghost PART FINAL. finish creating ghost methods”.

Добавлены реализация к оставшимся методам: draw(), reset(), update(), update\_target().

**Красавцев Максим**

“Merge remote-tracking branch 'origin/feature\_ghost' into develop”.

Объединение ветки feature\_ghost с develop.

**Еремеева Дарья**

“add graphics class code, add constant CELL\_SIZE”.

Добавлен код для класса graphics, добавлена константа CELL\_SIZE.

**Красавцев Максим**

“merge feature\_graphics, feature\_ghost\_manager, feature\_ghost into develop branch”.

Объединены ветки feature\_graphics, feature\_ghost\_manager, feature\_ghost с develop.

**Ботько Николай**

“create pacman.h”.

Создан код для файла pacman.h.

**Ботько Николай**

“add pacman.cpp and pacmans constants”.

Добавлен код для файла pacman.cpp и добавлены константы для пакмана.

**Красавцев Максим**

“Merge remote-tracking branch 'origin/feature\_pacman' into develop”.

Объединение ветки feature\_pacman' с develop.

**Красавцев Максим**

“merge feature\_pacman into develop and add all constants”.

Объединение ветки feature\_pacman с develop, добавлены все константы.

**Красавцев Максим**

“add main.cpp”.

Добавлен код main.cpp, слияние с веткой master.

# 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.Неиспользуемых переменных в коде нет. В алгоритмы методов старались не добавлять не нужных циклов, массивов и др. К отчету приложены диаграмма вариантов использования и диаграмма классов.