НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

УКРАЇНИ

«Київський політехнічний інститут імені Ігоря

Сікорського»

кафедра Технічної кібернетики

Створення комп’ютерної гри “Tower Defence”

Курсова робота

дисципліна

“ Алгоритми та структури данних ”

Керівник:

ст. викладач Дзінько Р.І.

Виконав:

Рижий А.В

“Захист дозволено” гр. ІК-82

“\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2019 р.

/підпис керівника/ /підпис студента/

Захищено з оцінкою

Завдання

Створити гру типу Tower Defense з користувацьким інтерфейсом, покращенням веж та кількома рівнями.

Під час створення гри реалізувати алгоритм пошуку найкоротшого шляху, який не вивчався до цього.

Зробити фреймову анімацію руху ворогів.

Реалізувати вежі, які атакують ворогів, якщо останні заходять в радіус атаки вежі, а пошкодження зараховуються і впливають на результат гри.

Зробити удар блискавкою, який при натисканні буде завдавати шкоди всім ворогам на карті.

Зміст

Вступна частина………………………………………………………………………..4

Обґрунтування і вибір алгоритму……………………………………………………..5

Розробка програми……………………………………………………………………..7

Керівництво розробнику……………………………………………………………….8

Керівництво користувачу……………………………………………………………...9

Висновок………………………………………………………………………………10

Додатки………………………………………………………………………………..11

Вступна частина

У грі використовується зручний та зрозумілий для користувача інтерфейс. Гра збалансована та має п’ять рівнів, які відрізняються один від одного і кожен рівень відрізняється складністю його проходження.

Чи не найбільшою перевагою є те, що моя гра не потребує підключення до мережі інтернет, отже вона чудово підходить для гри у подорожах.

Гра займає дуже мало місця і має низькі системні вимоги. Моя гра займає близько чотирьох з половиною мегабайт.

Ця гра є актуальною, адже всі люблять відпочити від роботи, або навчання, посидівши за грою, тому мої одногрупники вже встигли відвести душу в цій грі.

Моя гра підходить і для гри у короткий проміжок часу і для довготривалого ігрового процесу. Гра розвиває логіку та стратегічне мислення. У грі є свій час, тому можна змагатись з друзями хто швидше пройде рівень.

Обґрунтування і вибір алгоритму

У своїй грі я реалізував алгоритм пошуку найкоротшого шляху для ворогів, так як карта має декілька стежок, а вороги повинні як найшвидше дістатись до бази гравця. У моєму випадку найкраще підходить Алгоритм Лі для пошуку найкоротшого шляху, так як нам потрібно шукати найкоротшу відстань між двома точками на карті, так як ми користуємось цим алгоритмом тільки на початку рівня, нам не настільки важливий час його виконання, тому саме цей алгоритм є найбільш підходящим у цій ситуації і його реалізація не викликає ніяких складнощів. Алгоритм реалізовувався на стандартній карті, яка має вигляд двомірного масиву цілих чисел.

У реалізації цього алгоритму виконується пошук ортогонального шляху, тобто рахуються 4 сусідні клітинки. Наш найкоротший шлях будемо записувати у два цілочисельні масиви для іксів та ігреків. Мета алгоритму – прокласти найкоротший шлях від клітинки старту до клітинки фінішу , якщо це звичайно, можливо. ». Хвиля, у свою чергу, може проходити тільки через клітинки помічені як «прохідні». Хвиля рухається, поки не досягне точки старту. Після досягнення хвилею старту, прокладається шлях в правильному напрямі (від старту до фінішу) і зберігається в два масиви.

Алгоритм складається з трьох основних етапів: етап зчитування карти, етап поширення хвилі та етап побудови шляху.

На етапі зчитування карти проводиться аналіз клітинок, що присутні на карті, визначаються «прохідні» клітинки. Решта клітинок позначаються як «непрохідні».

Етап поширення хвилі полягає у знаходженні точки старту шляхом перебору сусідніх клітинок та присвоєння їм відповідної кількості кроків. В першу чергу перевіряються всі клітинки, суміжні з початковою. Якщо комірка «прохідна», їй присвоюється значення з лічильника кроків. Клітинки з іншою вагою (заповнені на попередніх кроках), а також клітинки, відмічені як «непрохідні», залишаються без змін. Якщо одна з клітинок є точкою старту, алгоритм переходить на наступний етап – побудову шляху. На етапі побудови шляху (згортання хвилі) починаємо рух з початкової точки. Для цієї точки обирається клітинка з її околу (вага цієї комірки буде n). Тоді для цієї клітинки знову шукається суміжна з вагою n-1. Таким чином продовжується доти, поки не знайдено клітинку з вагою 1, в околі якої знаходиться кінцева точка (вага початкової точки 0). Таким чином маємо побудований шлях, який також відносить до множини шляхів мінімальної довжини.

Розробка програми

У ході написання програми використовувались 8 класів, які відповідають за ігровий процес:

1. Ігрової карти
2. Сутності
3. Ворога
4. Вежі
5. Кількість життів у ворога
6. Гравця
7. Клітинки для вежі
8. Кулі для вежі

Кожен клас включає в себе необхідний набір функцій для робити з ними.

Гра створена за допомогою бібліотеки SFML, яка дає нам можливість зручно працювати з вікнами, графікою та подіями.

Використував контейнери std::list, std::vector зі стандартної бібліотеки шаблонів.

Керівництво розробнику

Для роботи даної програми потрібен комп’ютер зі встановленою операційною системою Windows та встановлений компілятор mingw32. Програма написана на мові програмування С++ з використанням бібліотеки SFML.

У програмі є два вікна:

* Меню
* Ігрова сцена

В залежності від рівня, на якому знаходиться користувач, завантажується відповідна карта, ворогам задається свій таймер появлення та затримка нової хвилі, при якій швидкість ворогів збільшується, також задається кількість ворогів, яких потрібно вбити, щоб виграти даний рівень. Також присвоюється початкова швидкість ворогів та кількість грошей, які гравець отримає при смерті ворога.

При заходження ворога на базу гравця, ворог зникає, а у гравця віднімається певна кількість життів.

Рівень вважається пройденим, коли гравець убив задану кількість ворогів на даному рівні, та кількість його життів більша від нуля. Коли ж гравець зазнав поразки(кількість життів дорівнює 0), то пропонується вихід у меню, або вихід з гри. Карта для кожного рівня обирається з 5 варіантів(Додаток 3-6).

Керівництво користувачу

Для встановлення даної гри потрібно запустити файл TD\_Setup.exe, який знаходиться на диску, обрати місце встановлення.(Додаток 1)

При запуску гри можна обрати рівень, або вийти з гри, натиснувши кнопку “QUIT” або ж нажавши хрестик у правому верхньому куті вікна.

Під час гри можна вийти в меню, натиснувши кнопку “MENU”, або вийти з гри нажавши кнопку “EXIT” або ж на хрестик в правому верхньому куті.

Для того щоб побудувати вежу, потрібно щоб кількість ігрової валюти дозволяла це зробити, якщо це так, тоді потрібно перетягнути картинку вежі на ігрову карту та розмістити її на одній з клітинок для веж. Темнішим кольором зафарбовані ті клітинки, на які ставити вежі буде ефективніше, чим темніша клітинка, тим більший внесок може робити вежа.

Для покращення вежі потрібно мати необхідну кількість ігрової валюти для цього, якщо це так, то покращення здійснюється при натисканні ЛКМ на вежу.

Якщо гравець виграв, то він побачить вікно з повідомленням про перемогу, якщо ж програв, то вікно з повідомленням про поразку, далі гравець може обрати може покинути гру, нажавши кнопку “EXIT”, або ж повернутись до меню, нажавши на “MENU”.

Висновок

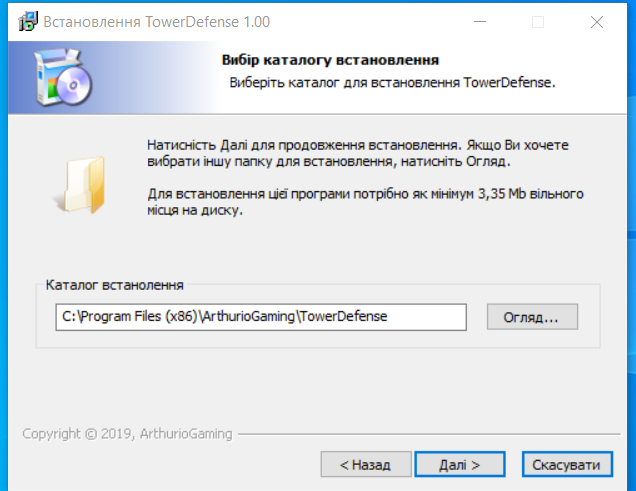
У ході виконання курсової роботи я опанував принципи ООП, а саме інкапсуляцію, наслідування та поліморфізм та навчився працювати з бібліотекою SFML. Створив перший великий проект на мові С++(більше 1500 рядків коду).

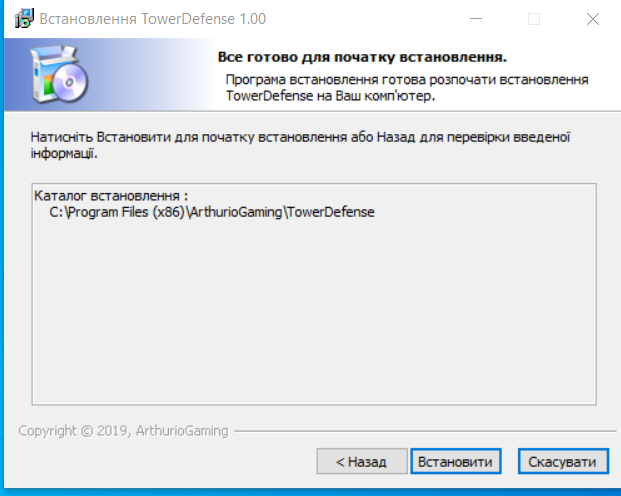
Гра є оптимізованою та використовує не більше 40 мб оперативної пам’яті та потребує невеликі ресурси процесора. Програма встановлюється менше ніж за три секунди та запускається менш ніж за 1 секунду.

Мету було досягнути, гра працює коректно, має низку рівнів різної складності то готова до встановлення на комп’ютери користувачів.

Додатки

Додаток 1

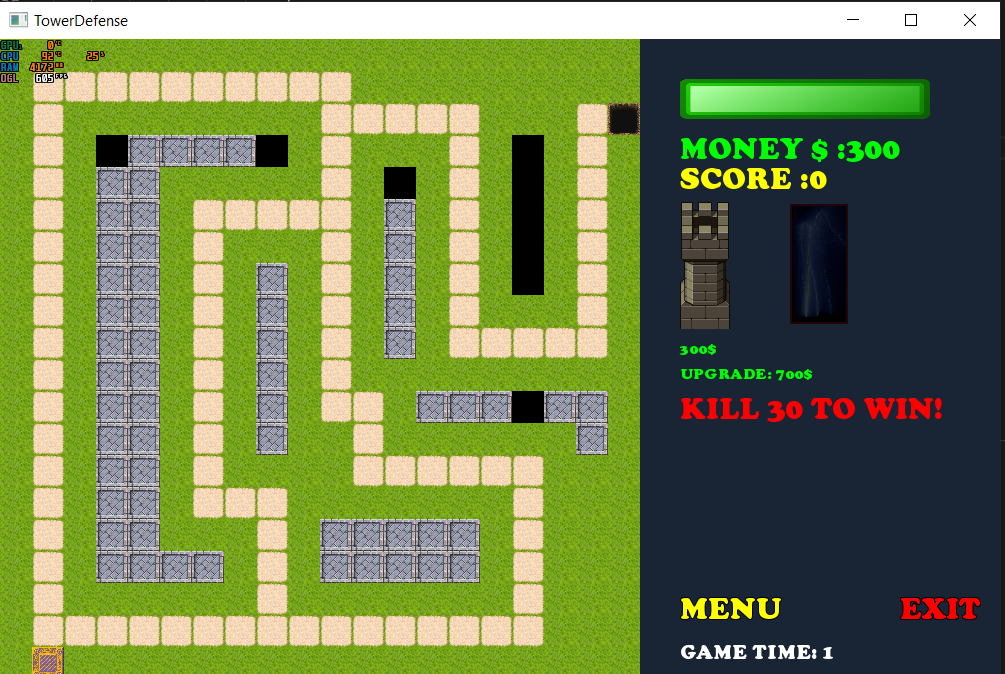




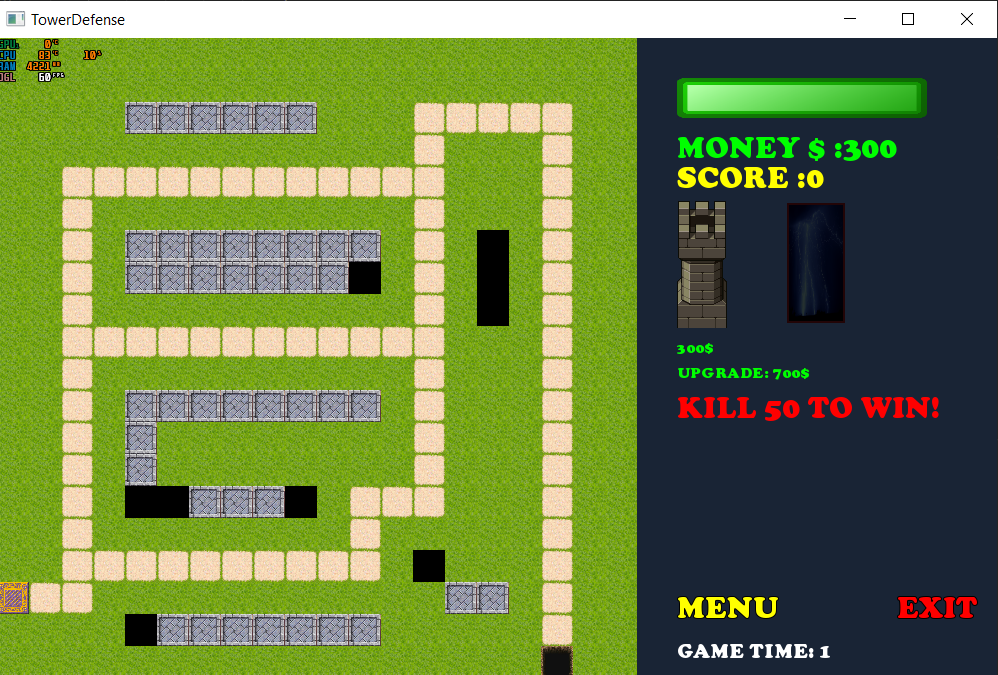
Додаток 2(меню)



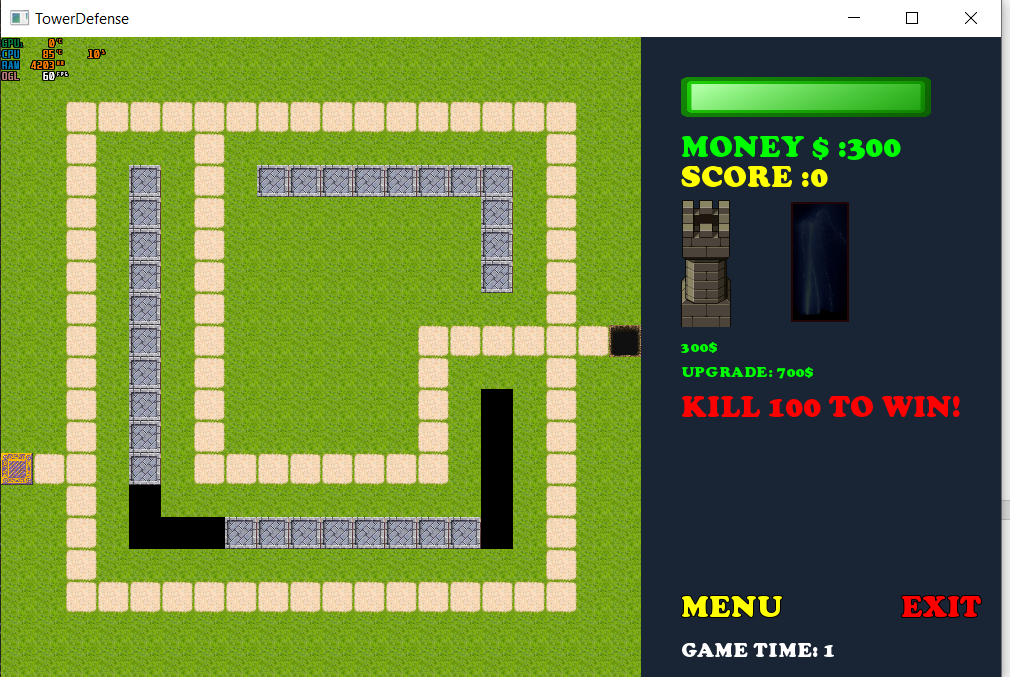
Додаток 3(карта 1)



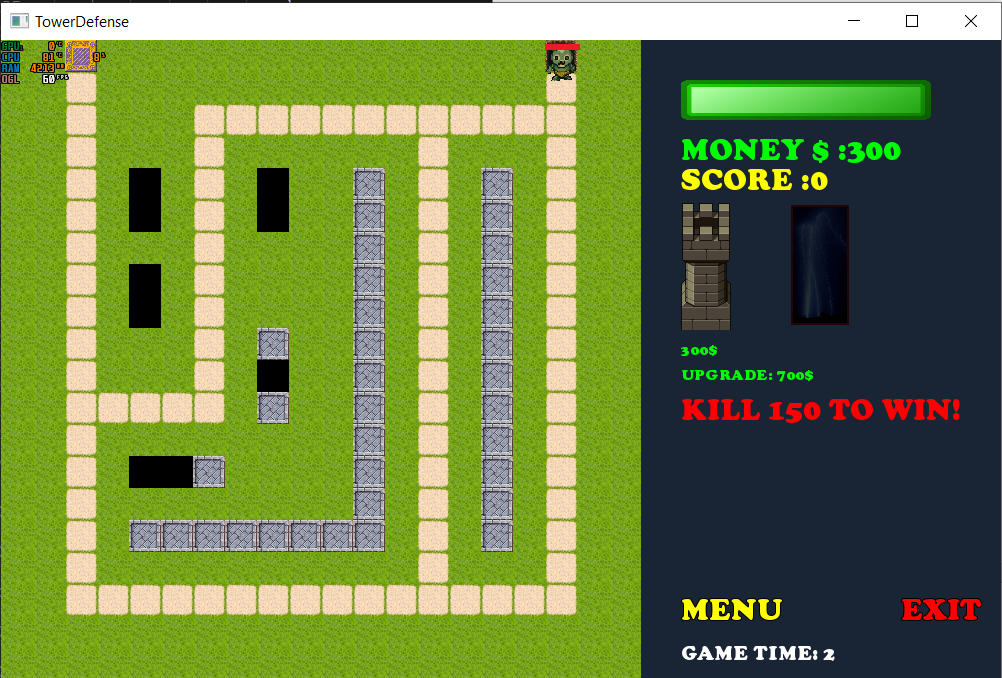
Додаток 4(карта 2)



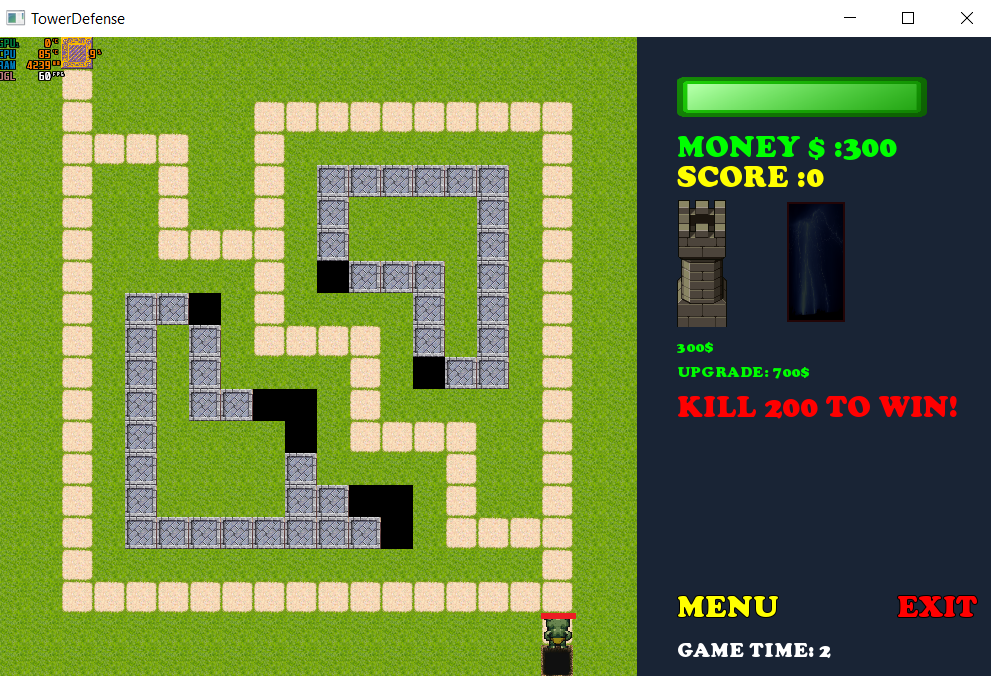
Додаток 5(карта 3)



Додаток 6(карта 4)



Додаток 7(карта 5)



Додаток 8(код програми)

bool tileMap::lee\_search()

{

const int WALL = -1; //impassable tile

const int BLANK = -2; //tile was not touched

int len; // length of the path

int grid[W][H];

for(int i = 0; i < H; i++)

for(int j = 0; j < W; j++)

{

if(tileArray[i][j] != 2 && tileArray[i][j] != 4 && tileArray[i][j] != 0)

{

grid[i][j] = WALL; // impassable

}

else

{

grid[i][j] = BLANK; // 0 steps were made

}

}

// lets find the shortest path from startPos to FinishPos

int dx[4] = {1, 0,-1,0}; //offset

int dy[4] = {0, 1, 0, -1};//r d l u

int d, x, y, k;

bool stop;

if(grid[startPos.y][startPos.x] == WALL || grid[finishPos.y][finishPos.x] == WALL)

{

return false; // one of the tiles is wall

}

//sharing a wave

d = 0;

grid[startPos.y][startPos.x] = 0; // mark the start point as 0

do

{

stop = true; //assuming all free tiles are marked

for( y = 0; y < H; ++y)

for(x = 0; x < W; ++x)

if(grid[y][x] == d) //if tile marked as d

{

for( k = 0; k < 4; ++k) // checking all nonmarked neigbours

{

int iy = y + dy[k], ix = x + dx[k];

if( iy >= 0 && iy < H && ix >= 0 && ix < W && grid[iy][ix] == BLANK)

{

stop = false; //found non-marked tiles

grid[iy][ix] = d + 1;

}

}

}

d++;

}while(!stop && grid[finishPos.y][finishPos.x] == BLANK); // path is not found

//setting the path

len = grid[finishPos.y][finishPos.x]; // length of the shortest path

x = finishPos.x;

y = finishPos.y;

d = len;

while(d > 0)

{

px[d] = x;

py[d] = y; //adding tile (x,y) to the path

d--;

for( k = 0; k < 4; ++k)

{

int iy = y + dy[k], ix = x + dx[k];

if( iy >= 0 && iy < H && ix >= 0 && ix < W && grid[iy][ix] == d)

{

x = x + dx[k];

y = y + dy[k]; // moving towards start on 1 tile

break;

}

}

}

px[0] = startPos.x;

py[0] = startPos.y;

px.resize(len + 1);

py.resize(len + 1);

return true;

};

tileMap::mapDraw(sf::RenderWindow& window)

{

for(int i = 0; i < 20; i++)

for(int j = 0; j < 20; j++)

{

if(tileArray[i][j] == 0) sprite.setTextureRect(sf::IntRect(0, 0, 32, 32));

if(tileArray[i][j] == 1) sprite.setTextureRect(sf::IntRect(32, 0, 32, 32));

if(tileArray[i][j] == 2) sprite.setTextureRect(sf::IntRect(64, 0, 32, 32));

if(tileArray[i][j] == 3)

{

int meanWhileOperator = 0;

sprite.setTextureRect(sf::IntRect(96, 0, 32, 32));

for(auto & it : towerTileList)

{

meanWhileOperator++;

if(meanWhileOperator == 10) break;

if(it.coordinates.x == j && it.coordinates.y == i)

{

spriteTowerTile.setTextureRect(sf::IntRect(96, 0, 32, 32));

spriteTowerTile.setColor(sf::Color(0 / it.effRate, 0 / it.effRate, 0 / it.effRate));

//std::cout<<"Hello"<<std::endl;

spriteTowerTile.setPosition(j \* 32, i \* 32);

}

}

}

if(tileArray[i][j] == 4) sprite.setTextureRect(sf::IntRect(128, 0, 32, 32));

sprite.setPosition(j \* 32, i \* 32);

window.draw(sprite);

window.draw(spriteTowerTile);

}

}

void tileMap::sortTowerTiles()

{

const int TOWERTILE = 3;

const int PATHTILE = 2;

int dx[12] = {-1, 0, 1, -2, -2, -2, -1, 0, 1, 2, 2, 2};

int dy[12] = {-2, -2, -2, 1, 0, -1, 2, 2, 2, 1, 0, -1};

int x,y,k,t;

for(y = 0; y < H; ++y)

for(x = 0; x < W; ++x)

{

if(tileArray[y][x] == TOWERTILE)

{

towerTile ttile;

ttile.effRate = 0;

ttile.coordinates.y = y;

ttile.coordinates.x = x;

ttile.tile\_isSelected = false;

for(k = 0; k < 12; ++k)

{

int iy = y + dy[k], ix = x + dx[k];

if(tileArray[iy][ix] == PATHTILE)

{

for(int t = 0; t < px.size(); t++)

{

if(iy == py[t] && ix == px[t])

{

ttile.effRate++;// increasing efficiency rate of the towerTile

}

}

}

}

towerTileList.push\_back(ttile);

}

}

std::sort(towerTileList.begin(), towerTileList.end());

}

void Player::p\_update(sf::RenderWindow& window, int SCORENEEDED)

{

std::ostringstream p\_scoreString;

std::ostringstream p\_moneyString;

std::ostringstream p\_scoreNeededStr;

p\_scoreString<< p\_score;

p\_moneyString<< p\_money;

p\_scoreNeededStr<< SCORENEEDED;

p\_moneyText.setString("MONEY $ :"+p\_moneyString.str());

p\_scoreText.setString("SCORE :"+p\_scoreString.str());

p\_towerPrice.setString("300$");

p\_upgradePrice.setString("UPGRADE: 700$");

p\_scoreNeeded.setString("KILL "+p\_scoreNeededStr.str()+" TO WIN!");

p\_upgradePrice.setFillColor(sf::Color::Green);

p\_towerPrice.setFillColor(sf::Color::Green);

p\_moneyText.setFillColor(sf::Color::Green);

p\_scoreText.setFillColor(sf::Color::Yellow);

p\_scoreNeeded.setFillColor(sf::Color::Red);

if(!p\_canStrike)

{

p\_sLightningIcon.setColor(sf::Color(37,48,48));

}

else

{

p\_sLightningIcon.setColor(sf::Color::White);

}

p\_sStrike.setPosition(0,0);

p\_sStrike2.setPosition(0,0);

window.draw(p\_upgradePrice);

window.draw(p\_moneyText);

window.draw(p\_scoreText);

window.draw(p\_towerPrice);

window.draw(p\_scoreNeeded);

window.draw(p\_sLightningIcon);

if(p\_health > 0 && p\_health < 1000)

{

p\_hbBar.setSize(sf::Vector2f(-((p\_hbMax - p\_health) \* 238) / p\_hbMax , 26));

}

if(p\_health < 100)

{

p\_hbBar.setSize(sf::Vector2f(-230, 26));

}

p\_hbSprite.setPosition(680, 40);

p\_hbBar.setPosition(680 + 250 - 11 ,40 + 7);

p\_spriteTower.setPosition(p\_tX, p\_tY);

p\_sGameOver.setPosition(0,0);

p\_sYouWin.setPosition(0,0);

window.draw(p\_hbSprite);

window.draw(p\_hbBar);

window.draw(p\_spriteTower);

}

int menu(sf::RenderWindow& window)

{

Texture backgroundTexture, lvl1Texture, lvl2Texture, lvl3Texture, lvl4Texture, lvl5Texture, quitTexture;

    lvl1Texture.loadFromFile("images/menu/level1.png");

    lvl2Texture.loadFromFile("images/menu/level2.png");

lvl3Texture.loadFromFile("images/menu/level3.png");

lvl4Texture.loadFromFile("images/menu/level4.png");

lvl5Texture.loadFromFile("images/menu/level5.png");

quitTexture.loadFromFile("images/menu/quit.png");

    backgroundTexture.loadFromFile("images/menu/background.jpg");

    Sprite lvl1(lvl1Texture), lvl2(lvl2Texture), lvl3(lvl3Texture), lvl4(lvl4Texture), lvl5(lvl5Texture), quit(quitTexture), background(backgroundTexture);

    bool isMenu = 1;

    int menuNum = 0;

    lvl1.setPosition(100, 30);

    lvl2.setPosition(100, 90);

    lvl3.setPosition(100, 150);

lvl4.setPosition(100, 210);

lvl5.setPosition(100, 270);

quit.setPosition(100, 330);

    background.setPosition(345, 0);

    /////////MENU//////////////

    while (isMenu)

    {

        sf::Event event;

while (window.pollEvent(event))

{

if(event.type == sf::Event::Closed)

            {

window.close();

            isMenu = false;

            }

}

        lvl1.setColor(Color::White);

        lvl2.setColor(Color::White);

        lvl3.setColor(Color::White);

lvl4.setColor(Color::White);

lvl5.setColor(Color::White);

quit.setColor(Color::White);

        menuNum = 0;

        window.clear(Color(11, 21, 38));

        if (IntRect(100, 30, 249, 51).contains(Mouse::getPosition(window))) { lvl1.setColor(Color::Blue); menuNum = 1; }

if (IntRect(100, 90, 249, 51).contains(Mouse::getPosition(window))) { lvl2.setColor(Color::Blue); menuNum = 2; }

if (IntRect(100, 150, 249, 51).contains(Mouse::getPosition(window))) { lvl3.setColor(Color::Blue); menuNum = 3; }

        if (IntRect(100, 210, 249, 51).contains(Mouse::getPosition(window))) { lvl4.setColor(Color::Blue); menuNum = 4; }

if (IntRect(100, 270, 249, 51).contains(Mouse::getPosition(window))) { lvl5.setColor(Color::Blue); menuNum = 5; }

        if (IntRect(100, 330, 165, 57).contains(Mouse::getPosition(window))) { quit.setColor(Color::Blue); menuNum = 6;}

        if (Mouse::isButtonPressed(Mouse::Left))

        {

            if (menuNum == 1){ isMenu = false; return 1;}//

            if (menuNum == 2) { isMenu = false; return 2; }

if (menuNum == 3){ isMenu = false; return 3; }

if (menuNum == 4){ isMenu = false; return 4; }

if (menuNum == 5){ isMenu = false; return 5; }

            if (menuNum == 6) { window.close(); isMenu = false; return 6;}

        }

        window.draw(background);

        window.draw(lvl1);

        window.draw(lvl2);

        window.draw(lvl3);

window.draw(lvl4);

window.draw(lvl5);

window.draw(quit);

        window.display();

    }

    ////////////////////////////////////////

}

Додаток 9(блок-схема алгоритмів)

