# 1 Вступ

* 1. **Опис програми**

Arkanoid – відеогра для ігрових автоматів, розроблена компанією Taito в 1986 році. Гра заснована на іграх серії Breakout фірми Atari [1]. Саме її назва стала загальною для класу подібних ігор.

Гравець контролює невелику платформу-ракетку, яку можна пересувати горизонтально від однієї стінки до іншої, підставляючи її під кульку, запобігаючи його падіння вниз. Удар кульки по блокові ​​призводить до його руйнування. Після того як всі блоки на даному рівні знищені, рівень завершується перемогою.

**1.2 Перегляд аналогів**

**Аналог 1: класична гра «Арканоїд».**

*Ресурс:* [http://www.officegamespot.com/arcade/arkanoid.htm [2](http://www.officegamespot.com/arcade/arkanoid.htm%20%5b2)].

*Вимоги:*

* наявність браузеру зі Flash Player;
* доступ до мережі Інтернет.

*Переваги:*

* інтуїтивно зрозумілий інтерфейс;
* різноманітні рівні;
* реалізована система бонусів;
* доречний звуковий супровід та анімація.

*Недоліки:*

* не має можливості зберегти та завантажити гру;
* не має можливості призупинити та продовжити призупинену гру.
* не ведеться облік часу, що робить гру нецікавою, оскільки за конкретний рівень можливо набрати постійну кількість ігрових балів;
* ігрові бали надаються тільки за певні види блоків, а не за всі;
* не має можливості відключити звук;
* некоректно ведеться облік «життів».

**Аналог 2: Космічний простір: Арканоід (Outer Space Arkanoid):**

*Ресурс:* http://www.game-game.com.ua/172489/ [3].

*Вимоги:*

* наявність браузеру зі Flash Player;
* доступ до мережі Інтернет.

*Переваги:*

* інтуїтивно зрозумілий інтерфейс.
* реалізована система проходження гри за рівнями.
* реалізована система бонусів.
* доречний звуковий супровід та анімація.
* система нарахування ігрових балів.
* на проходження рівня відводиться певна кількість часу.
* зручне керування.
* можливість відключення звуку.

*Недоліки:*

* не має можливості зберегти та завантажити гру.
* гра довго завантажується.
* гра містить рекламу.

**1.3 Актуальність**

В ході розгляду зазначених вище аналогів, був зроблений висновок, що на сьогоднішній день існує велика кількість ігор у жанрі «Арканоїд», проте більшість з них потребують доступу до мережі Інтернет. Також не має можливості збереження і завантаження гри.

 Облік недоліків і переваг аналогів дозволить об'єднати їх в більш якісну програму. Простота керування та роботи з програмою роблять її призначеною для широкого кола користувачів. Оскільки даний додаток носить розважальний характер і не примушує людину думати, то він зазичай використовується для коротшання часу при очікуванні. Тож реалізація гри, яка не потребує доступу до мережі Інтернет, є актуальною, оскільки дає можливість користуватися програмою при відсутності або низько швидкісного з’єднання з мережею.

Можливість збереження та продовження гри дасть змогу користувачу за необхідності перервати роботу з програмою, а за можливості продовжити незакінчений сеанс в роботі. Дана функція є корисною для користувачів, які коротають час при очікувані.

Тобто розроблена програма є актуальною та корисною.

# 2 Обґрунтування і вибір алгоритму

2.1 Параметри сцени і фігур

Розміри вікна цього додатка постійні, тому висота і ширина сцени так само незмінні. В якості фону використовується зображення формату JPG. Зберігається в папці з додатком.

Для промальовування блоку використовуються заздалегідь підготовлені зображення формату PNG – кожного кольору для кожного типу. Розмір зображення відповідає розміру блока і зберігається в папці з додатком. Блоки мають прямокутну форму, витягнуту по ширині.

Для промальовування ігрового поля використовується заздалегідь підготовлене зображення формату PNG. На полі розміщуються блоки у порядку встановленими налаштуваннями певного рівня. Рівень являє собою матрицю символів, де 0 – пуста коморка, А – сіра комірка, R – червона, G – зелена, B – синя, Y – жовта, O – золотиста. Ця матриця міститься у папці levels у файлі N.TXT, де N – номер рівня. Поле має прямокутну форму, витягнуту по висоті.

Для промальовування ракетки-платформи використовується підготовлене зображення формату PNG . Ракетка має прямокутну форму, витягнуту по ширині.

Для промальовування кульки використовується підготовлене зображення формату PNG. Кулька має форму сфери. Всі PNG зображення зберігаються в папці з додатком.

У верхній частині ігрового вікна знаходиться панель з елементами, які відображають додаткову інформацію про гру: рівень, час, кількість очок. У лівому нижньому куті – кількість залишилися життів.

2.2 Алгоритм ігрового процесу

Після запуску гри стартують 2 таймери – таймер, який відповідає за рахування часу у секундах, та таймер, по тіку якого рухається м’яч. Основна увага приділена руху м’яча. У стартовому положенні м’яч розташований таким чином, що його нижня сторона грань торкається ракетки (центр ракетки, розташований у центрі ширини поля), а координата верхнього лівого кута обчислюється за формулою 1:

***(1)***

*fieldWidth – ширина поля;*

*size – ширина м’яча.*

Таке розташування дає змогу розпочати рух м’яча з відхиленням убік (в даному випадку у правий), оскільки траєкторія руху м’яча залежить від місця його торкання з ракеткою, і обчислюється настуканим чином:

1. За формулою 2 обчислюється котангенс кута торкання, між висотою м’яча та уявною лінією проведеною з центра ракетки до верхньої середини м’яча:

***(2)***

*XM – координата верхнього лівого кута м’яча;*

*XP – координата верхнього лівого кута ракетки;*

*width – ширина ракетки;*

*size – ширина та висота м’яча;*

*сtg – котангенс кута, показаного на рисунку 1.*

*Рисунок 1 – котангенс кута*

2. Задається горизонтальна швидкість руху м’яча за формулою 3:

***(3)***

*xspeed –* горизонтальна швидкість руху м’яча*;*

*yspeed –* вертикальна швидкість руху м’яча*.*

Такі співвідношення обрані тому, що при рухові м’яча в діагональному напрямку, він стає більш складно керованим, що підвищує складність, та більш швидким, що підвищує швидкість проходження гри, порівняно з рухом м’яча вертикально вгору (адже при рухові вертикально вгору відсутня горизонтальна швидкість).

М’яч починає рух. При кожному тіку таймеру, м’яч горизонтальна координата м’ячу збільшується або зменшується шляхом додавання до цієї координати горизонтальної швидкості, а вертикальної координати – шляхом додавання до неї вертикальної швидкості (формула 4):

***(4)***

*YM – друга координата верхнього лівого кута м’яча.*

На подальшу траєкторію його руху будуть впливати зіткнення з блоками, ракеткою та стінками поля.

Умова зіткнення м’яча з ракеткою перевіряється за формулою 5:

***(5)***

Кожного разу при зіткненні з ракеткою перераховується горизонтальна швидкість руху за вище представленими формулами. Порівняння застосовуються зі знаками менше та більше, оскільки як вже було сказано вище – м’яч переміщується на задану величину, за якої не гарантовано опиниться за умов, коли його координати повністю відповідають координатам стінок об’єкта, з яким він зіткнувся, а може опинитися поза ними.

Після зіткнення з об’єктами з цієї причини робиться коригування координат. Наприклад, після зіткнення з ракеткою: *YM = YP – size.*

Розглянемо умови зіткнення зі стінками поля.

Умова зіткнення з правою стінкою поля обчислюється за формулою 6:

***(6)***

Після чого *XM = XM + size.* Наслідки зіткнення усуваються за формулою 7:

***(7)***

Умова зіткнення з лівою стінкою поля обчислюється за формулою 8:

***(8)***

Після чого *XM = 0.* Наслідки зіткнення усуваються за формулою 7.

Умова зіткнення з верхньою стінкою поля обчислюється за формулою 9:

***(9)***

Після чого *YM = 0.* Наслідки зіткнення усуваються за формулою 10:

***(10)***

Умова зіткнення з нижньою стінкою поля обчислюється за формулою 11:

***(11)***

*fieldHeight – висота поля.*

Ця умова є умовою виходу м’яча за межі поля, що є втратою життя. Після чого м’яч та ракетка повертаються у початкове положення.

Розглянемо умови зіткнення із блоками.

Почергово розглядаються координати стінок м’яча, уявляючи, що він має форму квадрату. Перевіряються умови перетину стінки м’яча зі стінками блоків, що розміщенні на полі. Для цього необхідно обчислити:

1. Рядок, поруч з яким розташована перша задана вертикальна координата стінки за формулою 12:

***(12)***

*Y1 – перша вертикальна координата стінки м’яча.*

*blockHeight – висота блока.*

2. За умови, що рядок менше загальної кількості рядків, розраховується стовпець, поруч з яким розташована перша задана горизонтальна координата стінки за формулою 13:

***(13)***

*X1 – перша горизонтальна координата стінки м’яча.*

*blockWidth – ширина блока.*

3. Якщо комірка матриці блоків з номером (row; column) – не порожня, то вважається, що м’яч зіткнувся з відповідною стінкою блока.

4. Перевіряється сусідня комірка по вертикалі або горизонталі.

# 3 Розробка програми

## 3.1 Загальні відомості

*Найменування продукту.*

Програмна реалізація гри-«Арканоїд».

*Коротка характеристика області застосування.*

Гра «Арканоїд» являє собою гру для одного користувача. Призначена для розваги користувачів.

*Призначення розробки.*

Програма відноситься до розряду розважальних. Вона дає можливість любителям комп'ютерних ігор провести вільний час.

*Вимоги до функціональних характеристик.*

Програма повинна здійснювати управління за допомогою клавіатури. Повинна мати можливість записувати і зберігати рекорди гравців, а також зберігати та завантажувати гру. Вести облік часу на проходження рівня та кількість ігрових балів.

*Вимоги до програмного та апаратного забезпечення.*

До складу технічних засобів повинен входити персональний комп'ютер (ПЕОМ), що включає в себе:

Вимоги до апаратного забезпечення:

* оперативна пам'ять: 1 Гб і більше;
* вільне місце на жорсткому диску: 200 МБ;
* частота процесора: від 1 ГГц;
* комп'ютерна миша і монітор;
* графічний пристрій з підтримкою DirectX 9.

Вимоги до інформаційної та програмної сумісності: програмне забезпечення має працювати під управлінням операційних систем сімейства Windows XP і вище.

Вимоги до захисту інформації не пред'являються.

*Мова програмування.*

Програма написана із застосуванням об'єктно-орієнтованого програмування на мові високого рівня С ++. Розробка програми проводилася в середовищі програмування MS Visual Studio 2010. Тип додатка Windows Forms (платформа .NET).

## 3.2 Функціональне призначення

Програма призначена для користувачів широкого рівня, не порубуючи особливих знань та навичок. Користувач має бути користувачем ПК початкового рівня. Дана програма носить розважальний характер і призначена для поліпшення моторних навичок, швидкості реакції, коротшання часу.

## 3.3 Опис логічної структури

Програма складається з класів, діаграма яких представлена у Додатку А та модуля ініціалізації. Код програми представлений у додатку Б.

***Модуль «Arkanoid.cpp»***

У цьому модулі містяться оператори ініціалізації компонентів додатка, створення головного вікна програми.

***Клас Bat***

Даний клас ініціює та зберігає дані та характеристики про ракетку, що розміщена на ігровому полі. Забезпечує розрахунок її координат для руху. Розраховує коефіцієнт зіткнення з м’ячем.

Реалізовує наступні публічні функції:

*Bat (int \_fieldWidth) –* конструктор класу, що використовує ширину поля для розрахунків початкових координат.

*virtual ~ Bat (void)* – віртуальний деструктор.

*void retToStartPos ()* – задає ракетці початкові координати по центру.

*void moveLeft ()* – пересуває ракетку вліво на один крок.

*void moveRight ()* – пересуває ракетку вправо на один крок.

*double collisionBat (int & x, int & y, int size) –* перевірка на зіткнення із об’єктом з координатами x, y та розміру size, знаходження котангенсу його кута.

Наступні методи надають доступ до розмірів і координат ракетки:

*int & batWidth ();*

*int & batHeight ();*

*int & batX ();*

*int & batY ();*

***Клас Ball***

Даний клас ініціює та зберігає дані та характеристики про м’яч, що рухається по на ігровому полю. Забезпечує розрахунок координат для руху.

Реалізовує наступні публічні функції:

*Ball (int \_fieldWidth, int \_batY)* *–* конструктор класу, що використовує ширину поля та вертикальну координату ракетки для розрахунків початкових координат.

*virtual ~ Ball (void)* – віртуальний деструктор.

*void retToStartPos()* – задає м'ячу початкові координати по центру на ракетці.

*void move()* – пересуває м'яч в залежності від швидкості.

Наступні методи надають доступ до швидкості м'яча:

*int & xSpeed ();*

*int & ySpeed ();*

Наступні методи надають доступ до розмірів і координат м'яча:

*int & ballSize ();*

*int & ballX ();*

*int & ballY ();*

**Клас Field**

Даний клас ініціює та зберігає дані та характеристики про поле та блоки, що на ньому розміщенні. Розраховує зіткнення м’яча зі стінками поля, а також із блоками. Веде облік балів за збиті блоки.

Реалізовує наступні публічні функції:

*Field ()* *–* конструктор класу.

*virtual ~ Field (void)* – віртуальний деструктор.

*void createFromFile (string fName)* – заповнення матриці блоків з файлу, що має ім’я fName.

*void save ()* – збереження матриці блоків в файл.

*int collisionWalls (int & x, int & y, int size)* ­– перевірка на зіткнення на зіткнення об’єкта з координатами x, y та розміру size з краями поля.

vector <int> collisionBlocks (int & x, int & y, int size, int & board) – перевірка на зіткнення об’єкта з координатами x, y та розміру size з краями блоків. Повертає вектор з номерами блоків, з якими зіткнувся об’єкт.

bool isEmpty () – перевірка чи залишились на полі блоки.

string operator [] (int n) – оператор для доступу до матриці через [].

int rowsNumber () – повертає кількість рядків матриці блоків.

Наступні методи надають доступ до ширини поля і до набраних балів.

*int & width ();*

*int & scores ();*

**Клас Game**

Даний клас ініціює та зберігає дані та характеристики про гру, в тому числі про ракетку, поле, м’яч. Доступ з класу, що реалізує інтерфейс до компонентів гри здійснюється тільки через цей клас, що забезпечує гнучкість програмного продукту. Містить у собі дані про рівень, час, гри, ігрові бали, кількість ігрових життів та види бонусів.

Реалізовує наступні публічні функції:

*Game (Field \_field)* *–* конструктор класу, що реалізує початок нової гри на полі, яке передане в якості параметра. Інші параметри ініціюються значеннями по замовченню.

*Game ()* – конструктор класу, що реалізує завантаження гру з файлу. Параметри ініціюються значеннями, зчитаними з файлу.

*virtual ~ Game (void)* – віртуальний деструктор.

*void save ()* – вивантажує гру в файл.

int testCollision () – протестувати м’яч на всі види зіткнення і вжити заходів.

*bool gameLose ()* – істина, якщо гра закінчилася поразкою.

*bool gameOver ()* – істина, якщо гра закінчилася перемогою.

Наступні методи надають доступ до параметрів класу:

*int & level ();*

*int & gLifes ();*

*int & gTime ();*

*int getBonus ();*

*int getScore ();*

*vector <int> getBitedBlocks ();*

Наступні методи забезпечують непрямий зв'язок з м'ячем і ракеткою:

*void moveBatLeft ();*

*void moveBatRight ();*

*int & batWidth ();*

*int & batHeight ();*

*int & batX ();*

*int & batY ();*

*int & ballSize ();*

*int & ballX ();*

*int & ballY ();*

**Класс LevelSelect.**

Являє собою графічну реалізацію гри. Дає можливість обрати рівень та грати в нього, використовуючи клас гри та клас поля для того, щоб обрати рівень.

Містить наступні основні функції:

­*void initLifeLabel ()* – ініціалізація мітки, що відображає кількість життів.  
*void initBat ()* – ініціалізація картинки ракетки і відображення її на екрані.

*void initBall ()* – ініціалізація картинки м'ячика і відображення його на екрані.

*void contGame (System :: Object ^ sender, System :: EventArgs ^ e)* –   
почати раніше збережену гру.

*void drawField (Field f)* – промальовувати поле з блоками на екрані.  
*void moveObject (int x, int y, PictureBox ^ pb)* – помістити картинку у місце з координатою x та y.

*System :: Void gameTime\_Tick (System :: Object ^ sender, System :: EventArgs ^ e)* – відображення часу від початку гри.

*System :: Void ballTimer\_Tick (System :: Object ^ sender, System :: EventArgs ^ e)* – забезпечення пересування м’яча за вище описаними законами полем.

*void endedGame ()* – дії, що необхідно виконати у разі закінчення гри.

*void checkResult ()* – перевірка результату гри (чи встановлено рекорд).  
void victoriActions () – дії, що необхідно виконати у разі перемоги.

*void loseActions (System :: Object ^ sender, System :: EventArgs ^ e)* – дії, що необхідно виконати у разі поразки.

*System :: Void LevelSelect\_KeyDown (System :: Object ^ sender, System :: Windows :: Forms :: KeyEventArgs ^ e)* ­– обробник події при натискання на певні клавіші.

*System :: Void LevelSelect\_Load (System :: Object ^ sender, System :: EventArgs ^ e)* ­– ініціалізація компонентів при завантаженні форми.

## 3.4 Виклик і завантаження

Установка гри здійснюється копіюванням її програмних файлів в каталог на жорсткому диску.

Процедура видалення гри виконується шляхом видалення її каталогу з жорсткого диска.

Запуск здійснюється шляхом запуску виконуваного файлу arcanoid.exe з директорії програми на жорсткому диску.

У програмі реалізований графічний інтерфейс. При завантаженні програми відображається головне меню програми.

## 3.5 Вхідні дані

*Вхідними даними для програми є:*

* зображення, що використовуються для фону та промальовування ігрових об’єктів форматів JPG та PNG;
* рівень, що являє собою матрицю символів; матриця міститься у папці levels у файлі N формату TXT, де N – номер рівня;
* список рекордів до певного рівня, який зберігається в папці result у файлі N формату TXT, де N – номер рівня;
* збережені параметри гри у файлі *saveGame* формату TXT;
* збережена матриця рівня гри у файлі save формату TXT;
* музика супроводу гри у форматі WAV.

## 3.6 Вихідні дані.

*Вхідними даними для програми є:*

* список рекордів до певного рівня, який зберігається в папці result у файлі N формату TXT, де N – номер рівня;
* збережені параметри гри у файлі *saveGame* формату TXT;
* збережена матриця рівня гри у файлі save формату TXT.

# 4 Керівництво користувачу

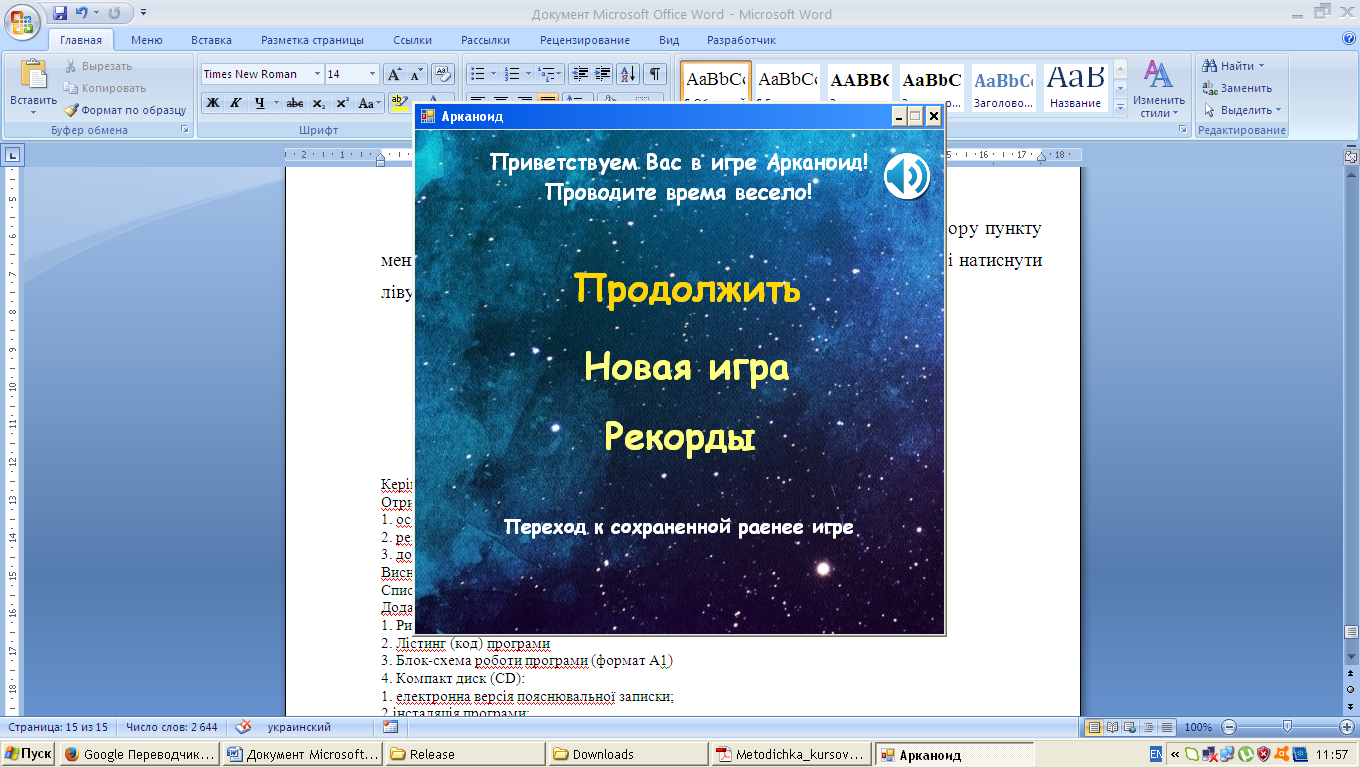
У програмі реалізований графічний інтерфейс. При завантаженні програми відображається головне меню програми з наступними пунктами (рисунок 4.1):

1. Продовжити.

2. Нова гра.

3. Рекорди.

Навігація по меню здійснюється за допомогою миші. Для вибору пункту меню користувачеві необхідно навести курсор на потрібний пункт і натиснути ліву кнопку миші. користувач також може вимкнути музику в грі.



*Рисунок 4.1 – Головне меню*

У відкритому пункті меню «Нова гра», щоб запустити гру користувач повинен натиснути спершу обрати рівень, а потім за готовністю натиснути клавішу пробіл.

*Правила гри:* ігрове поле містить блоки різного кольору.

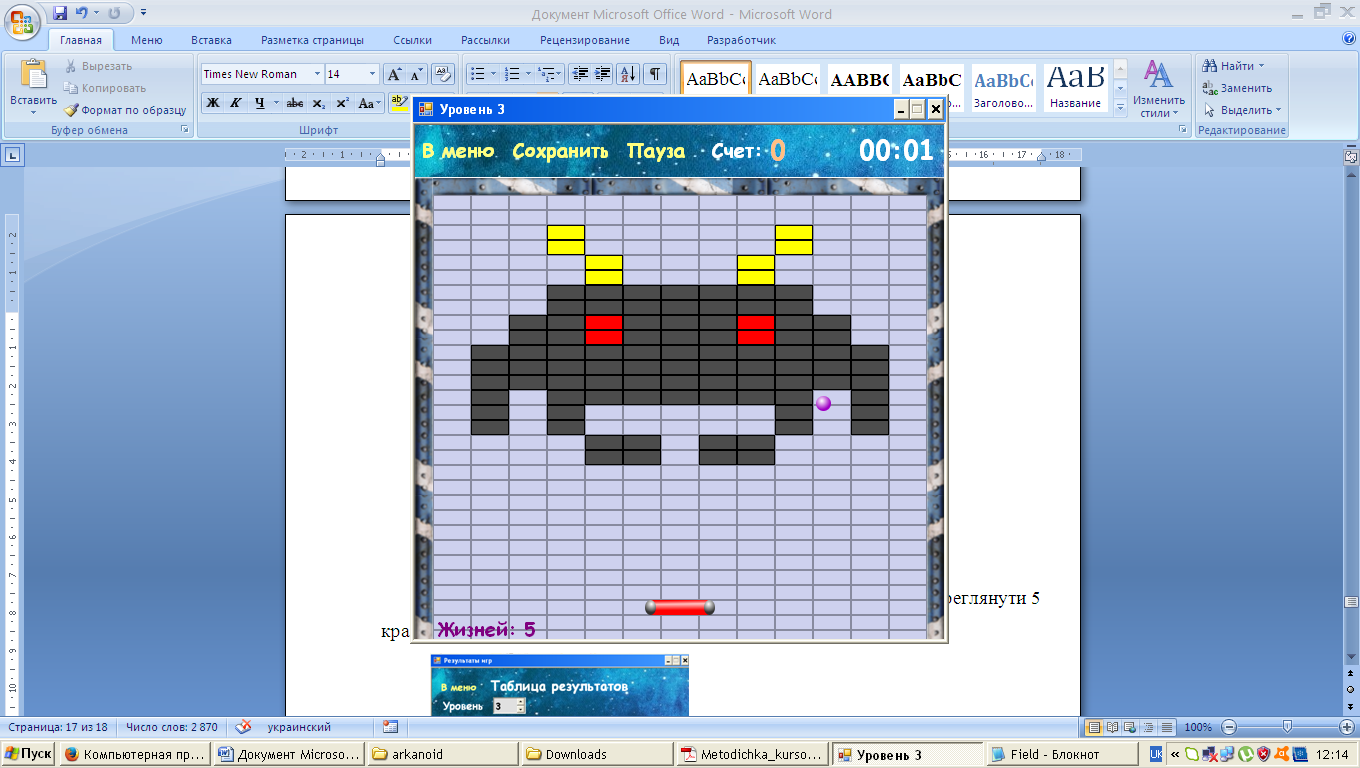
*Ціль гри* ­– за допомогою м'ячика розбити якнайшвидше всі блоки.

Усі блоки, крім блоків золотистого та сірого розбиваються з першого разу. Останні розбиваються лише з другої спроби. За кожен розбитий блок нараховуються бали.

Кількість балів за розбитий блок залежить від його кольору:

* сірий – 50;
* золотистий – 100;
* червоний – 4;
* зелений – 2;
* синій – 1;
* жовтий – 3.

Кількість балів, час, кнопка «Головне меню», кнопка «Зберегти» та кнопка «Пауза» відображаються в верхній частині екрану. Кількість життів – в нижньому лівому куті (рисунок 4.2).



*Рисунок 4.2 – Ігровий процес*

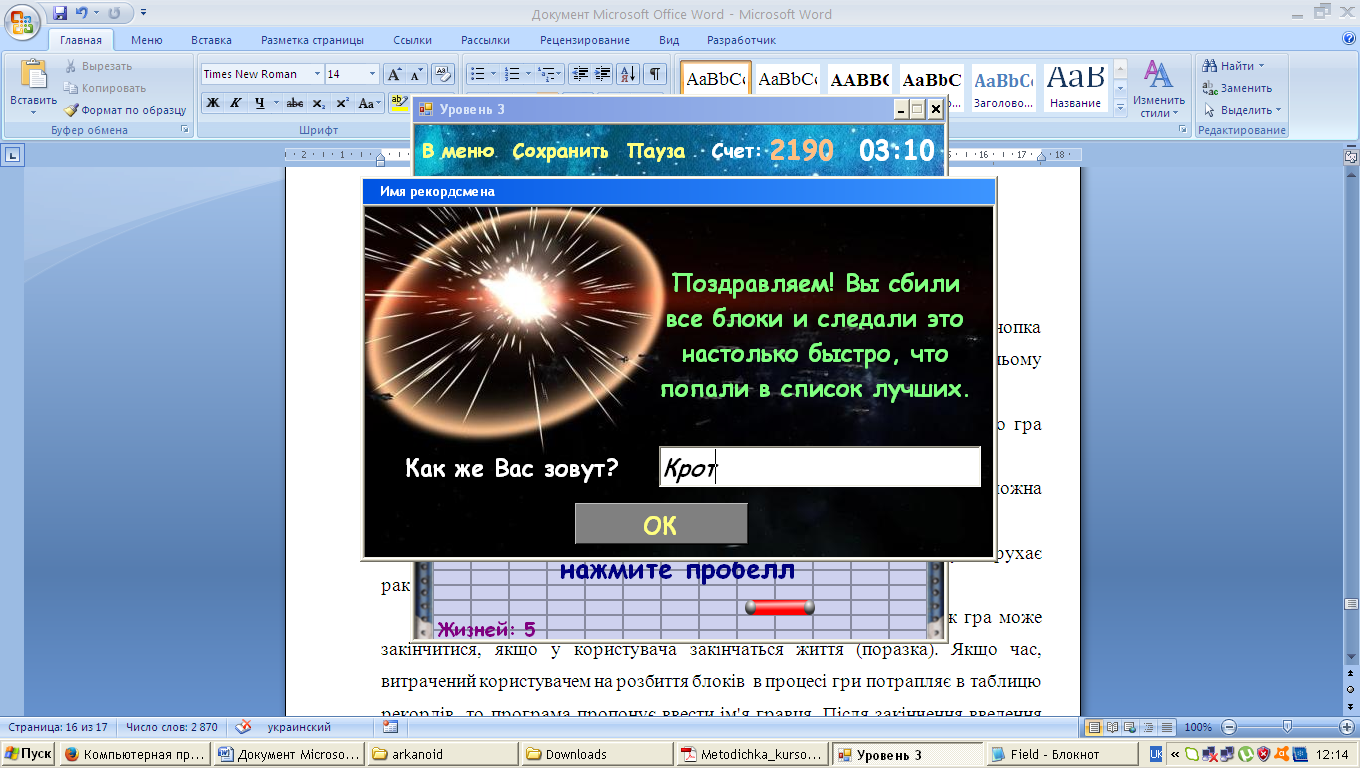
Якщо під час гри користувач натискає на «Пауза» або пробіл, то гра призупиняється. При повторному натисканні – продовжується.

При натисканні на кнопку зберегти гра зберігається. Збережену гру можна продовжити, викликавши з головного меню кнопкою «Продовжити».

За допомогою стрілочки вліво та вправо на клавіатурі користувач рухає ракетку у відповідну сторону.

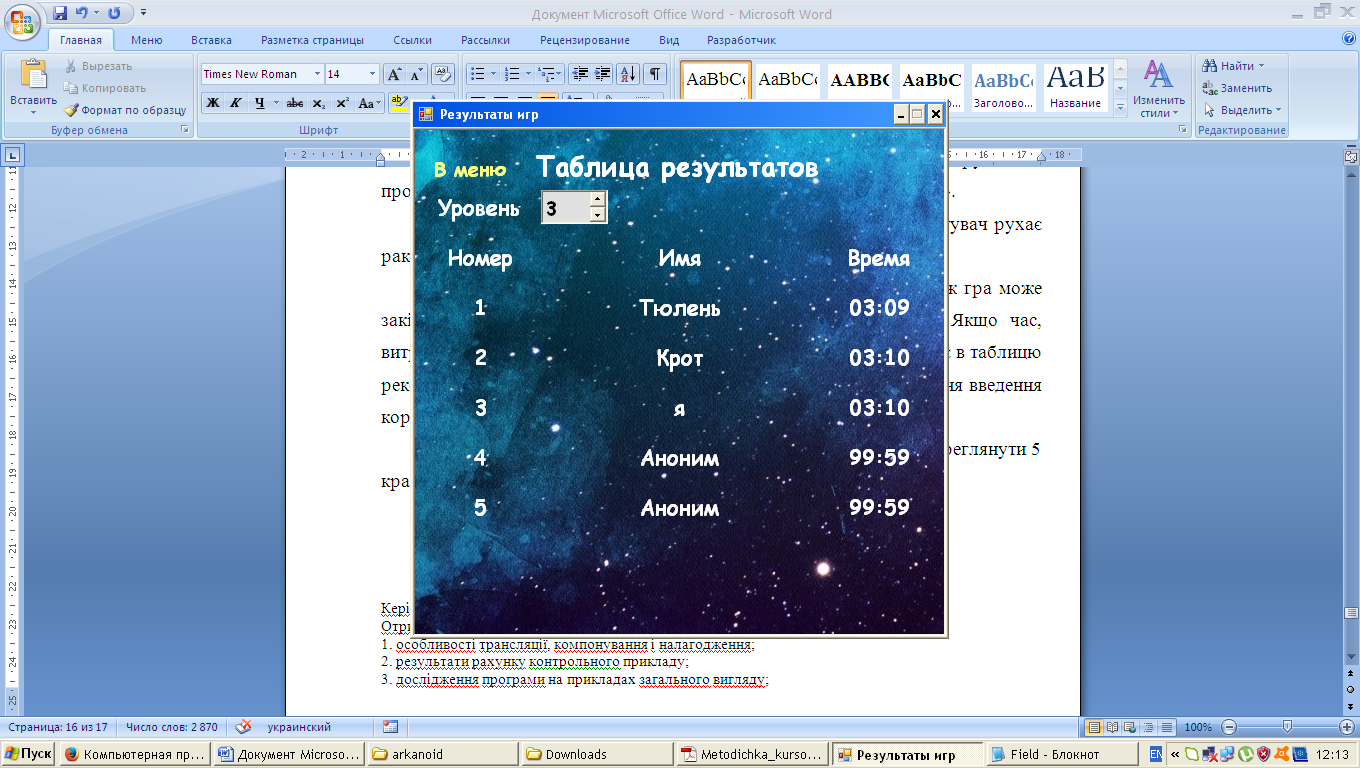
Якщо під час гри м’яч випав за межі поля, то гравець втрачає життя. Додаткове життя можна отримати у разі збиття блоку, що є бонусним.

Гра закінчується, коли користувач розбиває усі блоки. Також гра може закінчитися, якщо у користувача закінчаться життя (поразка). Якщо час, витрачений користувачем на розбиття блоків в процесі гри потрапляє в таблицю рекордів, то програма пропонує ввести ім'я гравця. Після закінчення введення користувачеві необхідно натиснути кнопку «ОК» (рисунок 4.3).



*Рисунок 4.3 – Введення ім’я рекордсмена*

Після переходу у пункт меню «Рекорди» користувач зможе переглянути 5 кращих часів проходження гри за рівнями.

**

*Рисунок 4.4 – Таблиця результатів*

# 5 Керівництво розробнику

Складні програмні системи розробляються колективом програмістів і, як правило, згодом модифікуються. Тому кожна програма добре описана, так щоб її зміг зрозуміти і використовувати інший розробник. До того ж, використаний об’єктна орієнтований підхід робить програму більш гнучкою, завдяки чому програму легко модифікувати, реалізовувати у 3Д інтерфейсі, додавати нові функції та можливості.

Об'єктно-орієнтоване програмування (ООП) – це новий підхід до створення програм. ООП акумулює кращі ідеї, втілені в структурному і модульному програмуванні, і поєднує їх з новими потужними концепціями, які дозволяють оптимально організовувати програми. ООП дозволяє розкласти проблему на складові частини найбільш природним способом з урахуванням ієрархії взаємозв'язків. Кожна частина стає самостійним об'єктом, який містить свої власні дані, операції і функції, які відносяться до цього об'єкту. В результаті процес складання програм спрощується і програміст отримує можливість оперувати з великими за обсягом програмами. Одна з головних особливостей ООП – це можливість визначати нові типи даних так, що їх використання нічим не буде відрізнятися від використання вбудованих в мову типів даних. Нові типи даних повинні в більшій мірі відповідати поняттям тієї прикладної області, для якої розробляється програма, ніж вбудовані типи, орієнтовані на архітектуру комп'ютера.

Задля забезпечення тривалого програмного життя необхідно знати та виконувати SOLID принципи [4]:

1. S – принцип єдиної відповідальності (The Single Responsibility Principle).  
Існує лише одна причина, яка веде до зміни класу.

2. O ­– принцип відкритості / закритості (The Open Closed Principle).  
«Програмні сутності ... повинні бути відкриті для розширення, але закриті для модифікації».

3. L – принцип підстановки Ліскової (The Liskov Substitution Principle).  «Об'єкти в програмі повинні бути замінними на екземпляри їх підтипів без зміни правильності виконання програми».

4. I – принцип поділу інтерфейсу (The Interface Segregation Principle)  
«Багато інтерфейсів, спеціально призначених для клієнтів, краще, ніж один інтерфейс загального призначення».

5. D – принцип інверсії залежностей (The Dependency Inversion Principle)  
    «Залежність на абстракцію. Немає залежності на щось конкретне».

Тобто дозволяється в подальшому розширювати існуючу задачу, проте забороняється вносити зміни у все існуючі методі.

Дозволяється використання окремих модулів. Програма складається з трьох модулів – модуль інтерфейсу, власне сама гра та модуль ініціалізації. Схема модулів представлена у додатку А.

# 6 Отримані результати

## 6.1 Особливості розробки програми та налагодження

Розробка програми велась поступово, реалізуючи певні її компоненти:

1. Реалізовано інтерфейс гри, налагодження переходу між його компонентами.

2. Реалізовано завантаження та вивантаження рівня з файлу та в файл.

3. Реалізовано відображення рівня на екрані.

4. Перевірка та налагодження описаних вище пунктів.

5. Реалізовано функцію вибору рівня.

6. Реалізований запуск гри, рух ракетки.

7. Розроблений алгоритм руху м’яча. Реалізований рух м’яча полем, відбивання від стінок, ракеток, блоків.

8. Налагодження, тестування програми у існуючій реалізації.

9. Виявлення помилок у траєкторії руху м’яча.

10. Доопрацювання алгоритму.

11. Повторення дій у пунктах 7-10, доки м’яч не почав рухатись необхідним чаном.

12. Реалізована можливість ставити гру на паузу.

13. Реалізована можливість збереження гри у файл.

14. Реалізована можливість продовження збереженої гри.

15. Налагодження, тестування нового функціоналу.

16. Виправлення помилок.

17. Доопрацювання деяких можливостей інтерфейсу, задля забезпечення естетичної привабливості.

18. Реалізація можливості закінчити гру поразкою та перемогою.

19. Реалізація можливості обліку кращих часів та введення імен таких рекордсменів.

20. Реалізація можливості переглянути результати гри за рівнями.

21. Налагодження нового функціоналу, тестування.

22. Виправлення недоліків.

23. Додавання музичного супроводу.

24. Доробка програмного продукту.

25. Тестування всього програмного продукту, підготовка фінальної версії.

Тобто як видно з опису етапів роботи над програмою, весь процес складається з етапів, під час яких реалізується певний функціонал. Розробляється алгоритм, реалізується алгоритм, налагоджується та тестується. Якщо на заданому етапі програма на давала необхідної точності та правил роботи, етап допрацьовувався, починаючи з початку, доти, доки програма не починала працювати наміченим чином. Тобто на кожному етапі програма стає працювати більш якісно та більш точно, нарощує свій функціонал.

Такий підхід, коли на кожному етапі є працююча програма забезпечується саме принципом ООП. Отриману програму можна доробляти таким ж самим чином.

## 6.2 Тестування

Для тестування програма була поширена серед кількох осіб, які були повинні протестувати весь функціонал програми. У кожного з тестерів була своя конфігурація комп'ютера і програмне забезпечення. Всі виявлені помилки і недоробки були усунуті.

# ВИСНОВОК

При виконанні даної роботи, потрібні були високі знання мови програмування. В ході написання програми потрібно звертатися до додаткових джерел, щоб розібратися з такими важливими аспектами програмування: робота з файлами, символьними даними, графічним режимом, вміння користуватися конструкторами, деструкторами, спадкуванням, а також ознайомиться з правилами гри «Арканоїд».

В цілому дана робота допомогла краще освоїти мову С ++ і поліпшити свої навички програмування.

В результаті курсової роботи створений новий програмний продукт – гра «Арканоїд». У ньому були реалізовані всі вимоги, встановлені в постановці задачі.

Варто відзначити, що в подальшому програма може бути поліпшена. Наприклад, реалізація бонусів, ввести «анти-бонуси», зробити 3D інтерфейс.

# Список використаних джерел

1. Арканоїд [Електронний ресурс]. — Режим доступу: https://ru.wikipedia.org/wiki/Arkanoid

2. Класична реалізація комп’ютерної гри «Арканоїд» [Електронний ресурс]. — Режим доступу: http://www.officegamespot.com/arcade/arkanoid.htm

3. Космічний простір: Арканоїд (Outer Space Arkanoid) [Електронний ресурс]. — Режим доступу: http://www.game-game.com.ua/172489/

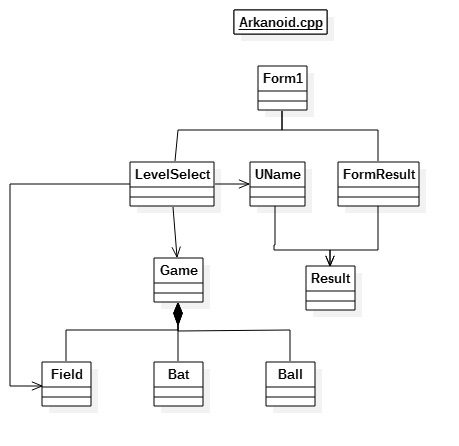
4. *Мартин Роберт.* Принципы, паттерны и методики гибкой разработки на языке C# / *Роберт Мартин, Мика Мартин.* — СПб.: Символ-плюс, 2011. — 768 с.

5. Хилл Ф. Программирование компьютерной графики. Для профессионалов. — СПб.: Питер, 2002. — 1088 с.

6. Мережа розробників Microsoft [Електронний ресурс] — Режим доступу: https://msdn.microsoft.com/ru-ru/

# Додаток А

Додаток містить діаграму класів.



Модуль гри

Модуль інтерфейсу

Модуль ініціалізації

# Додаток Б

**Модуль завантаження.**

// arkanoid.cpp: главный файл проекта.

#include "stdafx.h"

#include "Form1.h"

using namespace arkanoid;

[STAThreadAttribute]

int main(array<System::String ^> ^args)

{

// Включение визуальных эффектов Windows XP до создания каких-либо элементов управления

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

// Создание главного окна и его запуск

Application::Run(gcnew Form1());

return 0;

}

**Модуль гри.**

**Клас Ball**

#pragma once

#define SIZE 15 //размер мяча

#define MIN\_POS 0

#define SPEED 0

class Ball

{

public:

Ball(int \_fieldWidth, int \_batY);

virtual ~Ball(void);

void retToStartPos(); //задать мячу начальные координаты по центру на ракетке

void move(); //предвинуть мяч в зависимости от скорости

//методы предоставляющие доступ к скорости мяча

int& xSpeed();

int& ySpeed();

//методы предоставляющие доступ к размерам и координатам мяча

int& ballSize();

int& ballX();

int& ballY();

private:

int x, y; //координаты мяча

int xspeed, yspeed; //скороть мяча по х и у

int size; //размеры мяча

int fieldWidth, batY;

};

#include "StdAfx.h"

#include "Ball.h"

//конструктор использует ширину поля, у ракетки и инициализирует поля класса

Ball::Ball(int \_fieldWidth, int \_batY)

{

fieldWidth = \_fieldWidth;

batY = \_batY;

size = SIZE;

xspeed = yspeed = SPEED;

retToStartPos();

}

Ball::~Ball(void)

{

}

//задать мячу начальные координаты по центру на ракетке

void Ball::retToStartPos()

{

y = batY-size;

x = (fieldWidth + size)/2;

}

//предвинуть мяч в зависимости от скорости

void Ball::move()

{

y += yspeed;

x += xspeed;

}

//методы предоставляющие доступ к скорости мяча

int& Ball::xSpeed()

{

return xspeed;

}

int& Ball::ySpeed()

{

return yspeed;

}

//методы предоставляющие доступ к размерам и координатам мяча

int& Ball::ballSize()

{

return size;

}

int& Ball::ballX()

{

return x;

}

int& Ball::ballY()

{

return y;

}

**Клас Bat**

#pragma once

#define NARROW 70 //ширина узкой ракетки

#define BAT\_HEIGHT 15 //высота ракетки

#define MIN\_POS 0 //минимальная позиция слева

#define STEP 25 //шаг передвижения

#define BAT\_Y 405 //положение на поле по y

#define NO\_BAT\_COLLISION -100

class Bat

{

public:

Bat(int \_fieldWidth);

virtual ~Bat(void);

void retToStartPos(); //задать ракетке начальные координаты по центру

//предвинуть ракетку влево или в право

void moveLeft();

void moveRight();

//проверка на столкновение и нахождение точки

double collisionBat(int& x, int& y, int size);

//методы предоставляющие доступ к размерам и координатам ракетки

int& batWidth();

int& batHeight();

int& batX();

int& batY();

private:

int fieldWidth; //ширина поля

int width, height; //размеры ракетки

int x, y; //координаты ракетки

};

#include "StdAfx.h"

#include "Bat.h"

//конструктор использует ширину поля и инициализирует поля класса

Bat::Bat(int \_fieldWidth)

{

fieldWidth = \_fieldWidth;

width = NARROW;

height = BAT\_HEIGHT;

retToStartPos();

}

Bat::~Bat(void)

{

}

//задать ракетку начальные координаты по центру

void Bat::retToStartPos()

{

x = (fieldWidth - NARROW)/2;

y = BAT\_Y;

}

//предвинуть ракетку влево

void Bat::moveLeft()

{

//если ракетка не вызодит за пределы поля

if (x-STEP > MIN\_POS)

{

x -= STEP; //двигаем влево

}

else

{

x = MIN\_POS;

}

}

//предвинуть ракетку вправо

void Bat::moveRight()

{

//если ракетка не вызодит за пределы поля

if (x+STEP < fieldWidth-width)

{

x += STEP; //двигаем вправо

}

else

{

x = fieldWidth-width; //задаем крайнее правое положение

}

}

//проверка на столкновение и нахождение точки

double Bat::collisionBat(int& a, int& b, int size)

{

double ctg = NO\_BAT\_COLLISION;

//проверка на столкновение

if (y <= b + size && (x-a >= -width && x-a <= size))

{

b = y-size;

//нахождение параметра столкновения в зависимости от точки

ctg = (a + size/2 - (x + width/2))/(double)size;

}

return ctg;

}

//методы предоставляющие доступ к размерам и координатам ракетки

int& Bat::batWidth()

{

return width;

}

int& Bat::batHeight()

{

return height;

}

int& Bat::batX()

{

return x;

}

int& Bat::batY()

{

return y;

}

**Клас Field**

#pragma once

#include <map>

#include <vector>

#include <fstream>

#include <string>

#include <iostream>

using namespace std;

#define SAVE\_FIELD\_NAME "saveGame.txt"

#define BLOCK\_HEIGHT 15

#define BLOCK\_WIDTH 38

#define WIDTH 494

#define HEIGHT 415

#define MIN\_POS 0

class Field

{

private:

vector<string> matrix; //матрица блоков на поле

map<char, int> scoreTable; //таблица очков за убитые блоки

int score; //очки за сбитые блоки

int fWidth; //ширина поля

//проаерить коснулся ли объект блока и соседнему к нему

int checkBoard(int x, int y, int x1, int y1, int &row, int &col);

//применить результаты столкновения к полю

void applyCollision(vector<int> &index);

public:

Field();

virtual ~Field(void);

void createFromFile(string fName); //заполнение матирицы из файла

void save(); //сохранение матрицы в файл

int collisionWalls(int& x, int& y, int size); //проверка на столкновение объекта с краями поля

vector<int> collisionBlocks(int &x, int &y, int size, int &board); //проверка на столкновение объекта с краями блоков

bool isEmpty(); //проверка просты ли все ячейки поля

string operator[](int n); //для доступа к матрици через []

int rowsNumber(); //возвращает колицество строк матрицы

//методы предоставляющие доступ к ширине поля и очкам набранным

int& width();

int& scores();

};

#include "StdAfx.h"

#include "Field.h"

//конструктор класса - инициализирует переменніе и таблицу очков

Field::Field(void)

{

score = 0;

fWidth = WIDTH;

scoreTable.insert(pair<char, int>('R', 4));

scoreTable.insert(pair<char, int>('Y', 3));

scoreTable.insert(pair<char, int>('G', 2));

scoreTable.insert(pair<char, int>('B', 1));

scoreTable.insert(pair<char, int>('C', 50));

scoreTable.insert(pair<char, int>('D', 25));

}

Field::~Field(void)

{

}

//заполнение матирицы из файла

void Field::createFromFile(string fName)

{

ifstream fin;

string temp = "";

//загрузка уровней из файла

fin.open(fName);

while (!fin.eof())

{

fin >> temp;

matrix.push\_back(temp);

}

fin.close();

}

//сохранение матрицы в файл

void Field::save()

{

ofstream fout;

//загрузка уровня в файл

fout.open(SAVE\_FIELD\_NAME);

int rows = matrix.size();

for(int i = 0; i < rows-1; i++)

{

fout << matrix[i] << endl;

}

fout << matrix[rows-1];

fout.close();

}

//проверка на столкновение объекта с краями поля

int Field::collisionWalls(int& x, int& y, int size)

{

int wall = 0;

if (x + size >= WIDTH) //правый край

{

wall = 1;

x = WIDTH-size;

}

if (x <= MIN\_POS) //левый край

{

wall = 3;

x = MIN\_POS;

}

if (y <= MIN\_POS) //верхний край

{

wall = 2;

y = MIN\_POS;

}

if (y >= HEIGHT) //нижний край

{

wall = 4;

}

return wall;

}

//проверка на столкновение объекта с краями блоков

vector<int> Field::collisionBlocks(int &x, int &y, int size, int &board)

{

vector<int> result;

result.clear();

board = 0;

int row, col;

int num;

//вычислить с какой стороны было столкновение с блоком и было ли вообще

if (num = checkBoard(x, y, x+size, y, row, col) && (row+1 == matrix.size() ? true : (matrix[row+1][col] == '0' ? true : false)))

{ //снизу

board = 2;

y = (row+1)\*BLOCK\_HEIGHT;

}

else if (num = checkBoard(x, y+size, x+size, y+size, row, col) && matrix[row-1][col] == '0')

{ //сверху

board = 4;

y = (row)\*BLOCK\_HEIGHT-size;

}

else if (num = checkBoard(x, y, x, y+size, row, col) && matrix[row][col+1] == '0')

{ //справа

board = 1;

x = (col+1)\*BLOCK\_WIDTH;

}

else if (num = checkBoard(x+size, y, x+size, y+size, row, col) && matrix[row][col-1] == '0')

{ //слева

board = 3;

x = (col)\*BLOCK\_WIDTH-size;

}

if (board) //если столкеновение было

{

//создать результирущий вектор

result.push\_back(row);

result.push\_back(col);

if (num == 2)

{

if (board%2 == 0)

{

result.push\_back(row);

result.push\_back(col+1);

}

else

{

result.push\_back(row+1);

result.push\_back(col);

}

}

applyCollision(result); //применить результаты столкновения к полю

}

return result;

}

//проверка просты ли все ячейки поля

bool Field::isEmpty()

{

bool result = true;

for (int i = 0; i < matrix.size(); i++)

{

if (matrix[i] != "0000000000000")

{

result = false;

break;

}

}

return result;

}

//для доступа к матрици через []

string Field::operator[](int n)

{

string result = "";

if (n <= matrix.size())

{

result = matrix[n];

}

return result;

}

//возвращает колицество строк матрицы

int Field::rowsNumber()

{

return matrix.size();

}

//методы предоставляющие доступ к ширине поля и очкам набранным

int& Field::scores()

{

return score;

}

int& Field::width()

{

return fWidth;

}

//проаерить коснулся ли объект блока и соседнему к нему

int Field::checkBoard(int x, int y, int x1, int y1, int &row, int &col)

{

int result = 0;

row = y/BLOCK\_HEIGHT;

if (row < matrix.size())

{

col = x/BLOCK\_WIDTH;

if (col < matrix[row].length() && matrix[row][col] != '0')

{

result++;

}

if (col+1 < matrix[row].length() && x1 >= (col+1)\*BLOCK\_WIDTH)

{

if (matrix[row][col+1] != '0')

{

if (!result) col++;

result++;

}

}

else if (row+1 < matrix.size() && y1 >= (row+1)\*BLOCK\_HEIGHT)

{

if (col < matrix[row].length() && matrix[row+1][col] != '0')

{

if (!result) row++;

result++;

}

}

}

return result;

}

//применить результаты столкновения к полю

void Field::applyCollision(vector<int>& index)

{

for(unsigned int i = 0; i < index.size(); i+=2)

{

int a = index.size();

char bl = matrix[index[i]][index[i+1]];

if (bl != 'O' && bl != 'A')

{

score += scoreTable.find(matrix[index[i]][index[i+1]])->second;

matrix[index[i]][index[i+1]] = '0';

}

else

{

if (bl == 'O') matrix[index[i]][index[i+1]] = 'C';

if (bl == 'A') matrix[index[i]][index[i+1]] = 'D';

index.erase(index.begin()+i, index.begin()+i+2);

int b = index.size();

i-=2;

}

}

}

**Клас Game**

#pragma once

#include "Field.h"

#include "Bat.h"

#include "Ball.h"

#define BALL\_FALL 1

#define BLOCK\_BIT 2

#define NO\_CRITICAL\_S 0

#define START\_TIME 0

#define START\_LIFES 5

#define MAX\_BONUS 10

#define LIFE\_BONUS 1

#define SAVE\_NAME "save.txt"

class Game

{

public:

Game(Field \_field); //начало новой игры

Game(); //загрузить игру из файла

virtual ~Game(void);

void save(); //выгрузить игру в файл

//протестировать на столкновение и принять меры

int testCollision();

//покзывает закончилась ли игра поражением

bool gameLose();

//покзывает закончилась ли игра победой

bool gameOver();

//методы для обеспечения доступа к полям класса

int& level();

int& gLifes();

int& gTime();

int getBonus();

int getScore();

vector<int> getBitedBlocks();

//методы для обеспечения косвенной связи с мячом и ракеткой

void moveBatLeft();

void moveBatRight();

int& batWidth();

int& batHeight();

int& batX();

int& batY();

int& ballSize();

int& ballX();

int& ballY();

//получить поле

Field getField(){return field;}

private:

//компоненты игры

Field field;

Bat\* bat;

Ball\* ball;

//данные игры

int lifes;

int timeG;

int gLevel;

int bonus;

vector<int> bitedBlocks;

//генерирует случайные числа в диапозоне

int generateN(int from, int to);

};

#include "StdAfx.h"

#include "Game.h"

#include <time.h>

#include <stdlib.h>

//начало новой игры

Game::Game(Field \_field)

{

field = \_field;

bat = new Bat(field.width());

ball = new Ball(field.width(), bat->batY());

timeG = START\_TIME;

lifes = START\_LIFES;

bonus = 0;

gLevel = 0;

}

//загрузить игру из файла

Game::Game()

{

field.createFromFile(SAVE\_FIELD\_NAME);

bat = new Bat(field.width());

ball = new Ball(field.width(), bat->batY());

ifstream fin;

fin.open(SAVE\_NAME);

fin >> gLevel;

fin >> timeG;

fin >> field.scores();

fin >> lifes;

fin >> bat->batX();

fin >> bat->batY();

fin >> ball->ballX();

fin >> ball->ballY();

fin >> ball->xSpeed();

fin >> ball->ySpeed();

fin.close();

bonus = 0;

}

Game::~Game(void)

{

delete bat;

delete ball;

}

//генерирует случайные числа в диапозоне

int Game::generateN(int from, int to)

{

time\_t t;

t = time(&t);

srand(t);

return rand() % abs(from - to) + from;

}

//выгрузить игру в файл

void Game::save()

{

ofstream fout;

//загрузка уровня в файл

fout.open(SAVE\_NAME);

fout << gLevel << endl;

fout << timeG << endl;

fout << field.scores() << endl;

fout << lifes << endl;

fout << bat->batX() << endl;

fout << bat->batY() << endl;

fout << ball->ballX() << endl;

fout << ball->ballY() << endl;

fout << ball->xSpeed() << endl;

fout << ball->ySpeed() << endl;

fout.close();

field.save();

}

//протестировать на столкновение и принять меры

int Game::testCollision()

{

int result = NO\_CRITICAL\_S;

//задать скорость мяча, если не задана

if (ball->ySpeed() == 0) ball->ySpeed() = 6;

else ball->move(); //или передвинуть его

//проверить - если ли столкновение со стеной поля

if (int situation = field.collisionWalls(ball->ballX(), ball->ballY(), ball->ballSize()))

{

switch(situation)

{

case 1:

ball->xSpeed() = -ball->xSpeed();

break;

case 2:

ball->ySpeed() = -ball->ySpeed();

break;

case 3:

ball->xSpeed() = -ball->xSpeed();

break;

case 4:

ball->retToStartPos();

bat->retToStartPos();

lifes--;

result = BALL\_FALL;

break;

default:

break;

}

}

//если столкновение с ракеткой

double ctg = bat->collisionBat(ball->ballX(), ball->ballY(), ball->ballSize());

if (ctg != -100.0)

{

//задать скорость по х в зависимоти от точки столкновения

ball->xSpeed() = ctg\*abs(ball->ySpeed())\*0.9;

ball->ySpeed() = -ball->ySpeed();

}

//проверить на столкновение с блоком

int boardN;

bitedBlocks = field.collisionBlocks(ball->ballX(), ball->ballY(), ball->ballSize(), boardN);

if (boardN)

{

bonus = 0;

if (generateN(0, MAX\_BONUS) == LIFE\_BONUS)

{

bonus = LIFE\_BONUS;

lifes++;

}

result = BLOCK\_BIT;

if (boardN%2 == 0) ball->ySpeed() = -ball->ySpeed();

else ball->xSpeed() = -ball->xSpeed();

}

return result;

}

//покзывает закончилась ли игра поражением

bool Game::gameLose()

{

bool result = false;

if (lifes < 0)

{

result = true;

}

return result;

}

//покзывает закончилась ли игра победой

bool Game::gameOver()

{

return field.isEmpty();

}

//методы для обеспечения доступа к полям класса

int& Game::level()

{

return gLevel;

}

int& Game::gLifes()

{

return lifes;

}

int& Game::gTime()

{

return timeG;

}

int Game::getBonus()

{

return bonus;

}

int Game::getScore()

{

return field.scores();

}

vector<int> Game::getBitedBlocks()

{

return bitedBlocks;

}

//методы для обеспечения косвенной связи с мячом и ракеткой

void Game::moveBatLeft()

{

bat->moveLeft();

}

void Game::moveBatRight()

{

bat->moveRight();

}

int& Game::batWidth()

{

return bat->batWidth();

}

int& Game::batHeight()

{

return bat->batHeight();

}

int& Game::batX()

{

return bat->batX();

}

int& Game::batY()

{

return bat->batY();

}

int& Game::ballX()

{

return ball->ballX();

}

int& Game::ballY()

{

return ball->ballY();

}

int& Game::ballSize()

{

return ball->ballSize();

}

**Клас Result**

#pragma once

#include <fstream>

#include <string>

#include <iostream>

using namespace std;

#define FOLDER "results/"

#define RES\_N 5

class Result

{

public:

Result();

Result(int level);

virtual ~Result(void);

void add(string name, int gTime); //добавить время в таблицу результатов

bool isRecord(int gTime); //входит ли время в лучшие

string getName(int i); //имя по заданному индексу

int getTime(int i); //время по заданному индексу

int getLines(){return RES\_N;} //количество резкльтатов

private:

string names[RES\_N]; //массив имен

int times[RES\_N]; //массив времени

string fName; //имя файла

void writeToFile(); //загрузка в файл результатов уровня

void readFromFile(); //загрузка результатов уровня из файла

};

#include "StdAfx.h"

#include "Result.h"

Result::Result()

{

}

Result::Result(int level)

{

char num[3];

itoa(level+1,num,10);

string fN = num;

fName = FOLDER+fN+".txt";

readFromFile();

}

Result::~Result(void)

{

}

//добавить время в таблицу результатов

void Result::add(string name, int gTime)

{

int i = 0;

//найти место под запись нового результата

while (i < RES\_N && gTime >= times[i])

{

i++;

}

if (i < RES\_N)

{

//скопировать последующик на уровень ниже

for (int j = RES\_N-1; j <= i; j++)

{

times[j] = times[j-1];

names[j] = names[j-1];

}

//записать результат

times[i] = gTime;

names[i] = name;

}

writeToFile(); //записать в файл

}

//загрузка результатов уровня из файла

void Result::readFromFile()

{

ifstream fin;

fin.open(fName); //отрываем

//читаем

for (int i = 0; i < RES\_N; i++)

{

fin >> names[i];

fin >> times[i];

}

fin.close(); //закрываем

}

//загрузка в файл результатов уровня

void Result::writeToFile()

{

ofstream fout;

fout.open(fName); //отрываем

//записываем

for(int i = 0; i < RES\_N-1; i++)

{

fout << names[i] << " " << times[i] << endl;

}

fout << names[RES\_N-1] << " " << times[RES\_N-1];

fout.close(); //закрываем

}

//проверить входит ли время в лучшие

bool Result::isRecord(int gTime)

{

bool result = false;

if (gTime < times[RES\_N-1])

{

result = true;

}

return result;

}

string Result::getName(int i)

{

string result = "";

if (i < RES\_N)

{

result = names[i];

}

return result;

}

int Result::getTime(int i)

{

int result = 0;

if (i < RES\_N)

{

result = times[i];

}

return result;

}