Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного забезпечення «What is in common»

Студента 4 курсу, 6 групи,
спеціальності 121
«Інженерія програмного
забезпечення» _____ Баркара Ігора
підпис студента Олександровича

Науковий керівник
кандидат технічних наук
професор _____ Пашорін Валерій
підпис керівника Іванович

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола
підпис керівника Олександрович

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет *інформаційних технологій*
Кафедра *інженерії програмного забезпечення та кібербезпеки*
Освітній ступінь *бакалавр*
Спеціальність *121 «Інженерія програмного забезпечення»*
Спеціалізація / освітня програма *121 «Інженерія програмного забезпечення»*

Затверджую
Зав. кафедри інженерії
програмного забезпечення
та кібербезпеки
Криворучко О. В.
"7" листопада 2019 р.

Завдання
на випускну кваліфікаційну роботу студентові

Баркара Ігоря Олександровича

1. Тема випускної кваліфікаційної: *«Розробка програмного забезпечення «What is in common»»*.
Затверджена наказом ректора від *«02» грудня 2019 р. № 4112*
2. Строк здачі студентом закінченої роботи _____
3. Цільова установка та вихідні дані до роботи

Мета роботи *аналіз засобів, методів, побудови, розроблення і застосування довідково-інформаційної системи, яка реалізує довідки та виводить результати на екран.*

Об'єкт дослідження *дослідження програмних додатків, що сприяють розвитку мислення та логіки*

Предмет дослідження *методи та алгоритми розробки Windows – додатку.*

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

4. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Постановка задачі

1. Загальна частина

1.1. Постановка задачі

1.2. Дослідження і аналіз об'єкту програмування

1.3. Використані прогамні засоби

1.4. Вимоги до апаратного та програмного забезечення

Висновок до розділу 1

РОЗДІЛ2.Практична частина

2.1. Створення та налагодження програми

2.2. Опис програми та її алгоритмів

2.3. Інструкція програміста

2.4. Інструкція оператора

Висновки до розділу 2

РОЗДІЛ 3. Організаційно-економічна частина

3.1. Оцінка вартості програмного продукту

3.2. Розрахунок собівартості програмного продукту

3.3. Розрахунок ціни програмного продукту

3.4. Зведена таблиця показників

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*		
2.	*Вступ та перелік літературних джерел*		
3.	*Розділ 1. «Постановка задачі»*		
4.	*Розділ 2. «Практична частина»*		
5.	*Розділ 3. «Організаційно-економічна частина»*		
6.	*Висновки*		
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*		
8.	*Підготовка автореферату та презентації доповіді*		
9.	*Попередній захист випускної кваліфікаційної роботи*		
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*		
11.	*Здача прошитої випускної кваліфікаційної роботи на кафедру*		
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «___»_____20__ р.

8. Науковий керівник випускної кваліфікаційної роботи Котенко Н.О.

9. Гарант освітньої програми Цензура М. О.

10. Завдання прийняв до виконання студент Баркар І.О.

11. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускної кваліфікаційної роботи _____
(підпис, дата)

Відмітка про попередній захист Котенко Н.О _____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студента _____
(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____
(прізвище, ініціали, підпис)

Завідувач кафедри _____
(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (60с., 18 рис., 7таб., 1 додаток).

Об’єкт розробки – програмний додаток під назвою «What is in common» – розвиваюча гра-додаток для стаціонарних комп’ютерів та телефонів, яку можна випускати у повноцінне використання для людей будь-якого віку.

Завданням дослідження є: розробка розвиваючої гри-додатку, можливість гри на будь-яких пристроях, можливість монетизації.

В процесі розробки було використано мову програмування C#, а також C# – модулі. Visual Studio – для розробки графічного інтерфейсу. UWP – для створення додатку.

Ключові слова: гра-додаток, об'єктно-орієнтоване програмування, розробка графічного інтерфейсу, phpmyadmin, C#, Visual Studio, UWP.

ANNOTATION

Qualification work contains explanatory note (60 pages, 18 pictures, 7 tables, 1 addion).

An object of development – What is in common is a developmental game application for desktop computers and phones that can be released for full use for people of all ages. The task of research is: development of visual addition of advertising agency of "Argo", creation of individual included in addition, determination of catalogue of commodities of enterprise.

In the process of development, a programming of C#, and also C# language – modules was used. Visual Studio – for development of graphic interface. UWP – to create an application.

Keywords: game, application , object-oriented programming, development of graphic interface, C#, Visual Studio, UWP.

Перелік умовних позначень, символів, одиниць, сокрочень, термінів

XHTML – англ. Extensible Hypertext Markup Language – розширювана мова розмітки гіпертексту.

MySQL – вільна система керування реляційними базами даних.

PHP – PHP: Hypertext Preprocessor — популярна скрипкова мова.

SQL – англ. Structured Query Language — мова структурових запитів.

UML – англ. Unified Modeling Language — уніфікована мова моделювання.

XHTML – англ.. Extensible Hypertext Markup Language — мова розмітки гіпертексту.

XML – англ. EXtensible Markup Language — розширювана мова розмітки.

БД – база даних.

Грн – грошова одиниця, гривні.

ЕОМ – електронна обчислювальна машина – загальна назва для обчислювальних машин.

ПЗ – пояснювальна записка.

Зміст

					КНТЕУ 121 06-02.БР		
Зм.	Аркуш	№ докум	Підпис	Дата			
Зав. кафедри		Криворучко О.В.			Розробка програмного забезпечення «What is in common»	Стадія	Аркуш	Аркушів
Керівник		Пашорін В.І.				З	1	62
Гарант		Цензура М.О.						
Розроб.		Баркар і.О..			Зміст	Факультет інформаційних технологій, 4 курс, 6 група		

ВСТУП

«Наше життя-гра» . Кожного дня ми відкриваємо щось нове для себе.

У наших відкриттях є багато спільних рис, тому ми згадуємо багато речей через ассоціації.

Ассоціація – поняття, що виникає при згадуванні іншого.

Відповідно до пов'язування між собою, за принципом подібності, аналогії, протиставлення тощо, асоціації поділяють на декілька видів. Традиційно розрізняють прості та складні асоціації. Прості асоціації – це такі асоціації, які пов'язують між собою тільки два уявлення: асоціації за суміжністю, за схожістю, за контрастом, за каузальністю.

Асоціації за суміжністю уявляють собою встановлення зв'язків між предметами та явищами за ознакою просторово-часових відносин; такі виникають під час сприймання предметів, що розташовуються близько один від одного в просторі або йдуть безпосередньо один за одним у часі.

Асоціації за схожістю виникають у тому випадку, коли предмети або явища чимось схожі. За контрастом асоціюються контрастні, протилежні факти або явища. Асоціації за каузальністю створюються за ознакою причинно-наслідкових відносин. Вищі асоціації формують асоціативні комплекси.

Аристотель перший помітив явище асоціацію і ввів класичний поділ їх на 4 види:

1) асоціація за схожістю (кішка – тигр);
2) асоціація за контрастом (холодне – гаряче);
3) асоціація за суміжністю в просторі (поле – квітка);
4) асоціація за суміжністю в часі (ніч – сон).

Коли в групу поєднується три уявлення чи відчуття і більше, тоді виникає асоціятивний ряд. Пригадуючи якийсь предмет, особу, подію, враження, ми найчастіше одночасно пригадуємо цілий ряд інших, які з пригаданим образом стоять у більш або менш тісному зв'яку. Також стикаючись з якимось явищем чи ситуацією, ми часто пригадуємо схожі на них явища чи ситуації, раніше нами пережиті. Цей асоціативний ряд може практично до нескінченності продовжуватися: пригадаються друзі тих літ, веселі чи сумні пригоди, тодішні мрії, страхи, радощі, прикрощі тощо. І то пригадатися може навіть те, що ти вже давним-давно забув, тобто асоціативні ряди постачаються головним чином матеріалом зі сховів позасвідомого.

Особливість асоціативного мислення в тому, що його можна безперервно розвивати і покращувати, що дозволяє розширити свій потенціал. Розвитку асоціативного мислення сприяють різні вправи. Наприклад, найпростіше – це складання ланцюжків асоціацій. Ви просто берете будь-яке слово або ситуацію, а далі встигайте записувати, які асоціації будуть спливати в голові. Ще одне гарне вправа – пошук шляху асоціацій. Треба взяти два слова і написати між ними шлях з асоціацій. Будь-які вправи, у процесі яких потрібно працювати з асоціаціями, допомагають розвинути цей тип мислення. Саме для цього підійде гра «Що спільного?»

РОЗДІЛ 1.
ЗАГАЛЬНА ЧАСТИНА

1.1 Постановка задачі

Завдання дипломної роботи – розробити розширену версію гри «Що спільного?». Дані необхідно оформити у вигляді графічного інтерфейсу. В програмі повинні бути реалізовані наступні операції:

1) Використання класів при написанні програми.
2) Зчитування інформації з екрану.
3) Вибір рівнів
4) Завантаження останнього пройденого рівня
5) Таблиця рекордів.

Користувач націлений на пошук спільної риси, серед картинок за певний час. Програма повинна мати зручний інтерфейс і забезпечувати ряд операцій для допомоги користувачу під час використання програми.

При поразці гравцю надається можливість завантажити останній пройдений ним рівень, а також реалізована функція завантаження будь-якого пройденого гравцем рівня.

Гра завершується, коли гравець не встигає вгадати їх спільну рису, або робить вибір на не вірну рису.

Після запуску, для спрощення дій користувача, програма надає за замовчуванням рандомне ім’я з можливістю його зміни, для подальшого ведення рахунку очок.

Гра розрахована на широку аудиторію користувачів, без вікових обмежень.

1.2 Дослідження і аналіз об’єкту програмування

Гра – діяльність людини з моделювання іншого виду діяльності з розважальною чи навчальною метою. Гра відрізняється від роботи тим, що не ставить перед собою безпосередньо корисної мети (хоча сама гра може мати свою власну корисність), а також тісно межує з мистецтвом, хоч зазвичай не створює художніх цінностей.

Для гри «Що спільного?» є багато ігор аналогів, деякі приклади приведенні нижче.

Асоціації

Цей режим жає нам гру з двума картинками та одним словом. Дається 2 картинки, необхідно здогадатися, що вийде в результаті змішування понять, представлених на 2-х картинках. Доступні підказки: відкрити букву, прибрати зайві літери, текстова підказка.

Рис. 1.1 Гра «Асоціації»

Що заховано?

В цьому режимі дається всього одна картинка, розділена на квадрати. Ви можете безкоштовно відкрити 4 квадрата, за інші доведеться платити по 3 монети. Для проходження рівня потрібно здогадатися, яке слово означає захована картинка.

Рис. Гра «Що заховано?»

Ребуси

Загадка, в якій слова, що розгадуються, зображено у вигляді комбінації малюнків з літерами та іншими знаками.

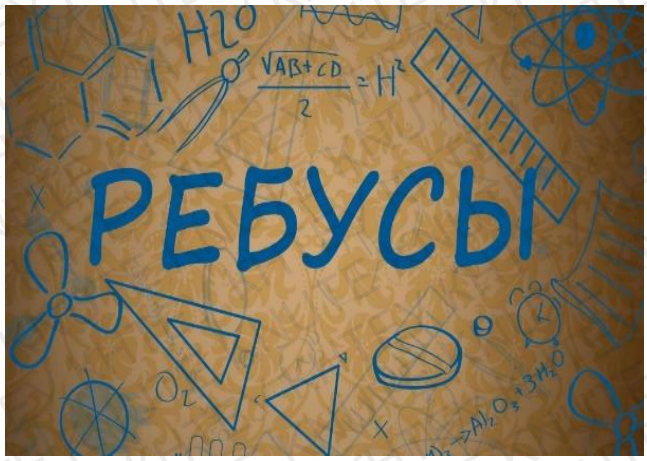

Рис. 1.3 Гра «Ребуси»

Антоніми

Суть режиму в тому, що потрібно відгадувати 2 слова по дом картинкам, які є антонімами по відношенню один до одного.

Рис 1.4 Гра «Антоніми»

Боротьба умів

Це інтелектуальна і захоплююча соціальна гра-вікторина, в якій можна змагатися в знаннях з друзями та іншими гравцями.

Рис. 1.5 Гра «Боротьба умів»

Таблиця 1

Зведена таблиця типу та опису ігор-аналогів

Назва	Тип	Короткий опис
1	2	3
Асоціації	Гра на асоціативність	Необхідно за двома картинками знайти їх схожість
Що заховано?	Флеш гра на уважність	У цій грі гравцю необхідно знайти заховані на картинці предмети
Ребуси	Флеш гра на уважність	Необхідно вгадати загадане слово у ребусі
Антоніми	Гра на асоціативність	Дається 2 картинки, и треба знайти їх відмінну рису
Боротьба умів	Гра на мобільний пристрій	Необхідно здолати іншого гравця, таким чином, щоб у вас було більше правильних відповідей на запитання гри

1.3 Використані програмні засоби

На підставі задачі, вхідної інформації, вимог до результатів рішення, наявних обчислювальних ресурсів і операційної системи було визначено вирістання потрібних технологій та алгоритмів.

Опишемо причини використання кожної з використаних технологій та алгоритмів.

Для написання додатку було вирішено використовувати мову C#. **C#** – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET.

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього

використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++).

UWP (Universal Windows Platform) являє собою уніфіковану платформу для створення і запуску додатків в Windows 10 і Windows 10 Mobile.

UWP стала результатом фолюції більш ранніх технологій. Так, з виходом Windows 8 була впроваджена нова архітектурна платформа для додатків - Windows Runtime (WinRT), яка дозволяла запускати додатки в так званому режимі Modern (Metro) на десктопах, планшетах. Потім з виходом Windows 8.1 і Windows Phone 8.1 ця технологія отримала розвиток - з'явилися "універсальні додатки", які можна було запускати відразу Windows 8.1 і WP8.1. І в липні 2015 офіційно вийшла нова ОС Windows 10. Вона використовує платформу UWP, яка представляє собою розвиток Windows Runtime.

Як підказує назва платформи, вона є універсальною – універсальної для всіх пристроїв екосистеми Windows 10. А це звичайні дестопи, планшети, мобільні пристрої, пристрої IoT (інтернет речей), Xbox, пристрої Surface Hub. І додаток UWP може однаково працювати на всіх цих платформах, якщо на них встановлена Windows 10.

Чому UWP?

- Програмування під UWP несе ряд переваг:
- Широта поширення. На поточний момент (квітень 2017) Windows 10 встановлена вже більш ніж на 400 мільйонах пристроїв. На десктопах Windows 10 вже випередила Windows 8 / 8.1.
- Підтримка широкого кола пристроїв. Музичні кліпи, планшети, смартфони, великі планшети Surface Hub, різні IoT-пристрої, в перспективі пристрою віртуальної реальності HoloLens - коло устрйоства, на яких може працювати Windows 10 дійсно широкий.

- Підтримка різних мов і технологій програмування. UWP-додатки можна створювати за допомогою таких мов, як Visual C ++, C #, Visual Basic, JavaScript. Як технології для створення графічного інтерфейсу Visual C ++, C # і Visual Basic використовують XAML, JavaScript застосовує HTML. Крім того, C ++ може замість XAML використовувати DirectX. Тобто досить поширені і та знайомі багатьом технології.

- Магазин додатків і зручність поширення. Windows Store є прекрасне місце для поширення UWP-додатків, як платних, так і безкоштовних. Самі можливості платформи і магазину Windows Store дозволяють використовувати різні способи монетизації. Наприклад, можна інтегрувати в додатка блоки для показу реклами через різні SDK. Можна поширювати за певну плату, причому оплату можна гнучко налаштовувати. При необхідності можна вбудувати надання ознайомчої версії, після використання якої користувач може вирішити, купувати додаток чи ні. І також можна монетизувати за моделлю freemium, при якій додаток умовно безкоштовне, а окремі послуги всередині програми надаються за певну плату. Причому всі ці можливості монетизації обесечіваются вбудованими інструментами SDK.

- Багаті можливості платформи. UWP багато успадковує від Windows Runtime з Windows 8.1 і в той же час надає багато нових функцональностей, як, більш багаті можливості по інтеграції з хмарою, використання Cortana, системи повідомлень в Win10 і багато іншого.

Структура проекту UWP

Щоб зрозуміти, як Windows10 дозволяє визначати різні класи пристроїв, необхідно зрозуміти концепцію під назвою сімейства пристроїв. Сімейство пристроїв визначає API-інтерфейси, характеристики системи і поведінки, очікувані на пристроях певного класу. Воно також визначає набір пристроїв, на які може бути встановлено додаток з магазину. Сімейство пристроїв (за винятком сімейства універсальних пристроїв) реалізовано у вигляді розширення SDK, яке

ми незабаром обговоримо. Ось ієрархія сімейства пристроїв.

- Сімейство пристроїв визначає набір API і має різні версії. Сімейство пристроїв є основою ОС. Комп'ютери та планшети працюють під управлінням класичної ОС, яка ґрунтується на сімействі настільних пристроїв. Телефони працюють під управлінням мобільної ОС, яка ґрунтується на сімействі мобільних пристроїв.

- Кожне дочірнє сімейство пристроїв додає в успадковані інтерфейси власні API-інтерфейси. Гарантується, що отримується в результаті об'єднання API-інтерфейсів в дочірньому сімействі пристроїв буде представлено в ОС, заснованої на цьому сімействі пристроїв, і на кожному пристрої під керуванням цієї ОС.

- Одним з переваг універсального сімейства пристроїв є те, що додаток можна запускати на всій безлічі пристроїв-телефонах, планшетах, настільних комп'ютерах, пристроях Surface Hub, консолях Xbox і HoloLens. Крім того, ваш додаток може використовувати адаптивний код для динамічного виявлення і використання функцій пристрою, що виходить за рамки універсального сімейства пристроїв.

- Цільове сімейство пристроїв (або сімейства) для вашого застосування вибираєте ви. Цей вибір впливає на важливі аспекти вашого застосування. Воно визначає:

- Набір API-інтерфейсів, які можуть бути присутніми у вашому додатку під час його роботи (і тому можуть бути вільно викликані).

- Набір викликів API, що захищаються тільки в межах умовних операторів.

- Набір пристроїв, на яких може бути встановлено ваш додаток з магазину (і, відповідно, форм-факторів, які необхідно враховувати при створенні користувальницького інтерфейсу).

- Існує два основні наслідки вибору сімейства пристроїв: поверхню API, яку додаток може викликати безумовно, і кількість пристроїв, з якими це додаток може зв'язатися. Ці два фактори припускають вибір оптимального співвідношення і є обернено пропорційними. Наприклад додаток UWP є додатком, яке головним чином орієнтоване на універсальні сімейства пристроїв і тому є для всіх пристроїв. Додаток, яке орієнтоване на універсальне сімейство пристроїв, може припускати наявність API-інтерфейсів тільки з універсального сімейства пристроїв. Інші API-інтерфейси необхідно викликати за умовою. Крім того, такий додаток повинен мати високоадаптівний призначений для користувача інтерфейс і широкі можливості введення, так як воно може запускатися на самих різних пристроях. Мобільний додаток для Windows є додатком, яке головним чином орієнтоване на сімейство мобільних пристроїв і доступно для пристроїв під управлінням ОС, заснованої на сімействі мобільних пристроїв (яке включає телефони, планшети і подібні пристрої). Додаток сімейства мобільних пристроїв може припускати наявність всіх API-інтерфейсів в сімействі мобільних пристроїв, а його користувальницький інтерфейс повинен бути помірно адаптивним. Додаток, яке орієнтоване на сімейство пристроїв IoT, може бути встановлено тільки на пристроях IoT і може припускати наявність всіх API-інтерфейсів в сімействі пристроїв IoT. Ця програма може бути вузькоспеціалізованим по частині призначеного для користувача інтерфейсу і можливостей введення, так як ви знаєте, що воно буде працювати тільки на певному типі пристроїв.

- Ось кілька питань, які допоможуть вам вирішити, яке сімейство пристроїв вибрати:

Максимальне розширення охоплення вашої програми

Щоб ваше додаток охопило максимальну кількість пристроїв і могло використовуватися на максимально можливій кількості типів пристроїв, воно буде орієнтоване на універсальне сімейство. Це дозволить додатком автоматично

орієнтуватися на все сімейства пристроїв, засновані на універсальному сімействі (на схемі це все дочірні елементи універсального сімейства). Це означає, що під час запуску програми на будь-який ОС, заснованої на цих родинах пристроїв, і на всіх пристроях, що працюють під управлінням цих операційних систем. Єдині API-інтерфейси, які гарантовано будуть доступні на всіх цих пристроях, – це набір, певний конкретною версією універсального сімейства пристроїв, на яке ви орієнтуєтеся. Щоб дізнатися, як додаток може викликати API-інтерфейси, що виходять за межі встановленої версії сімейства пристроїв, ознайомтеся з підрозділом

Обмеження додатки одним типом пристрою

Можливо, ви не хочете, щоб ваше додаток працювало на широкому ряді пристроїв; скажімо, воно призначається для настільних комп'ютерів або консолі Xbox. В цьому випадку ви можете орієнтувати додаток на одне з дочірніх родин пристроїв. Наприклад, якщо ви орієнтуєтеся на сімейство настільних пристроїв, для вашого застосування гарантовано будуть доступні API-інтерфейси, включаючи успадковані від універсального сімейства пристроїв і специфічні для сімейства настільних пристроїв.

Обмеження додатка підмножиною всіх можливих пристроїв

Замість орієнтації на універсальне сімейство пристроїв або на одне з дочірніх родин, ви можете вибрати два (або більше) дочірніх сімейства пристроїв. Для вашої програми може бути доцільна орієнтація на настільні комп'ютери і мобільні пристрої. Або настільні комп'ютери і HoloLens. Або настільні комп'ютери, Xbox, Surface Hub і т.д.

Виняток підтримки для певної версії сімейства пристроїв

Можливо, ви хочете, щоб в деяких випадках ваше програма не запускалася на конкретної версії конкретного сімейства пристроїв. Наприклад, ваш додаток орієнтоване на універсальне сімейство пристроїв версії 10.0.x.0. У момент, коли в майбутньому версія операційної системи зміниться, наприклад на 10.0.x.2, ви

можете вказати, що ваш додаток виконується на всіх версіях, крім версії 10.0.x.1 для Xbox, зорієнтувавши свій додаток на версію 10.0.x. 0 для універсальних пристроїв і 10.0.x.2- для Xbox. Після цього ваш додаток виявиться недоступним для набору версій сімейства пристроїв в рамках Xbox 10.0.x.1 (включно) і більш ранніх версій.

- За замовчуванням Microsoft Visual Studio визначає Windows.Universal як цільове сімейство пристроїв у файлі маніфесту пакета програми. Щоб вказати в Магазині сімейство або сімейства пристроїв, для яких призначене ваше додаток, вручну налаштуйте елемент TargetDeviceFamily в файлі Package.appxmanifest.

Пакети SDK розширення

Після вибору сімейства пристроїв, на який буде орієнтований ваш додаток, додайте посилання на розширення SDK, яке реалізує інтерфейси API для цього сімейства пристроїв. Якщо ви орієнтовані на сімейство універсальних пристроїв, не потрібно посилатися на розширення SDK. Але якщо ви орієнтовані на сімейство інших пристроїв, крім універсальних, в Visual Studio необхідно додати посилання на розширення SDK, яке відповідає обраному сімейству пристроїв. Наприклад, якщо ви орієнтовані на сімейства мобільних пристроїв, необхідно додати посилання на Windows Mobile Extensions для UWP в диспетчері посилань в Visual Studio.

Вибір сімейства пристроїв не забороняє додавати розширення SDK для інших типів пристроїв.

Інтерфейс і універсальний введення

Додаток UWP може виконуватися на багатьох типах пристроїв, які мають різні форми введення, дозволу екрану, щільність точок на дюйм і інші унікальні характеристики. Windows10 надає нові універсальні елементи управління, панелі макетів і інструменти, які допомагають адаптувати користувальницький інтерфейс для пристроїв, на яких може запускатися додаток. Наприклад ви можете адаптувати користувальницький інтерфейс, щоб скористатися перевагами

різниці в дозволі екрану під час запуску програми на настільному комп'ютері і на мобільному пристрої.

Деякі аспекти призначеного для користувача інтерфейсу вашої програми будуть автоматично адаптовані під всі пристрої. Елементи управління, наприклад кнопки і повзунки, автоматично адаптуються під всі сімейства пристроїв і режими введення. Проте, в залежності від пристрою, на якому буде виконуватися додаток, може знадобитися адаптація дизайну інтерфейсу користувача. Наприклад в додатку для фотографій необхідно адаптувати користувальницький інтерфейс під управління на маленькому наладонних пристрої, щоб забезпечити оптимальне використання однією рукою. Якщо додаток для фотографій виконується на настільному комп'ютері, призначений для користувача інтерфейс повинен бути адаптований для використання переваг додаткового простору екрану.

Windows допомагає орієнтувати ваш призначений для користувача інтерфейс на кілька пристроїв за допомогою наступних функцій:

- Універсальні елементи управління і панелі макета допомагають оптимізувати призначений для користувача інтерфейс для дозволу екрану пристрою.

- Обробка загального введення дозволяє отримати вхідні дані за допомогою дотику, пера, миші або клавіатури, або контролера, такого як контролер Microsoft Xbox.

- Інструменти допоможуть розробити призначений для користувача інтерфейс, який адаптується під різні дозволи екрану.

- Адаптивне масштабування підлаштовується під відмінності в дозволі і DPI на всіх пристроях.

Універсальні елементи управління і панелі макета

Windows 10 включає нові елементи управління, такі як календар і комбінований режим. Елемент управління Pivot, який раніше був доступний тільки для Windows Phone, тепер також доступний для універсального сімейства пристроїв.

Елементи управління були оновлені, щоб забезпечити роботу на екранах більшого розміру, власну адаптацію, засновану на кількості доступних на пристрої пікселів екрану, і роботу з декількома типами введення, наприклад клавіатурою, мишею, сенсорним введенням, пером і контролерами, наприклад контролером Xbox.

Можливо, вам буде потрібно адаптація всього макета призначеного для користувача інтерфейсу в залежності від дозволу екрану пристрою, на якому буде виконуватися вашу програму.

Проектування адаптивного призначеного для користувача інтерфейсу за допомогою адаптивних елементів управління

Панелі макета передають розміри і положення своїм дочірнім елементам в залежності від доступного простору. Наприклад, StackPanel зумовлює свої дочірні елементи послідовно (горизонтально або вертикально). Клас Grid схожий на сітку CSS, дочірні елементи якої розташовуються в осередках.

Новий клас RelativePanel реалізує стиль макета, який визначається зв'язками між його дочірніми елементами. Він використовується при створенні макетів додатка, які можуть адаптуватися до змін дозволу екрану. RelativePanel спрощує процес переупорядковування елементів за рахунок визначення зв'язків між ними, що дозволяє будувати більш динамічний призначений для користувача інтерфейс без використання вкладених макетів.

У наступному прикладі, незалежно від змін орієнтації або макета, кнопка blueButton з'явиться праворуч від поля textBox1, а кнопка orangeButton з'явиться відразу ж нижче і буде вирівняна щодо кнопки blueButton, навіть якщо ширина поля textBox1 буде змінюватися зі зворотним відліком в нього. Раніше для

досягнення такого ефекту потрібні були б рядки і стовпці в Grid, а тепер це можна зробити за допомогою значно меншої кількості розмітки.

1.4 Вимоги до апаратного та програмного забезпечення:

- Процесор – Intel Pentium x64, не нижче 300 MHz
- Обсяг ОЗУ – не нижче 64 Мб
- Відеокарта: NVIDIA Geforce GT 630M
- Вільний простір на диску – 100 Мб
- Монітор з роздільною здатністю не менше 800x600, TrueColor
- Клавіатура – Windows-сумісна
- Наявність комп'ютерної миші або touch screen
- Операційна система – Windows / XP / 7 / 8 / 8.1 / 10

Висновок до розділу 1

У цьому розділі я провів роботу щодо створення гри-додатку, провів аналіз схожих ігор:

- інтерфейс;
- функціонал.

Також, зробив висновки по використанню програмних засобів, що буду використовувати, через аргументи, які були вказані у розділі 1. Були проведені перші тестові налаштування та виведені системні параметри, які необхідні для запуску програми.

РОЗДІЛ 2.
ПРАКТИЧНА ЧАСТИНА

2.1 Створення та налагодження програми

Для написання програми була використана мова програмування C# та середовище розробки Microsoft Visual Studio 2017.

Для кожної програми створюється один файл проекту, один make-файл і один файл ресурсів. Файл проекту генерується при виборі пункту меню File / New Application. Спочатку файлу проекту присвоюється за замовчуванням ім'я Project1.cpp. Якщо в процесі розробки програми додаються форми і модулі, C# оновлює файл проекту.

Для перегляду файлу проекту слід вибрати пункт меню View / Project Source. Ця операція виконає завантаження вихідного тексту файлу проекту в редактор коду.

У файлі проекту є певний набір ключових елементів:

- Директива препроцесора #include <vcl \ vcl.h> призначена для включення в текст проекту заголовки, що посилається на описи класів бібліотеки компонентів.

- Директива препроцесора #pragma hrdstop призначена для обмеження списку заголовків файлів, доступних для попередньої компіляції.

- Директива USEFORM повідомляє, які модулі і форми використовуються в проекті.

- Директива USERES компілятора приєднує файли ресурсів до виконуваного файлу. При створенні проекту автоматично створюється файл ресурсів з розширенням * .res для зберігання курсорів, піктограми програми та ін.

- Application-> Initialize () – це твердження критично тільки в разі, якщо додаток є OLE automation-сервером. В інших випадках воно фактично нічого не робить.

- Application-> CreateForm () – це твердження створює форму додатка. За замовчуванням, кожна форма в додатку має своє твердження CreateForm.

- Application-> Run () – це твердження запускає додаток (точніше, переводить його в стан очікування настання однієї з подій, на яке воно має реагувати).

- Конструкція try ... catch використовується для коректного завершення програми в разі виникнення помилки при ініціалізації, створення форм, запуску програми.

Коли до проекту додається нова форма, створюються три окремих файлу:

h-файл (.h) – містять задану опис класу форми, тобто опису містяться на формі компонентів і обробників подій.

Файл форми (.dfm) – двійковий файл, який містить відомості про опубліковані (тобто доступних в інспекторові об'єктів) властивості компонентів, що містяться в формі. Двійковий файл форми містить інформацію, використовувану для конструювання форми з компонентів, розташованих на ній. При додаванні компонента до форми та заголовки, і двійковий файл форми модифікуються. При редагуванні властивостей компонента в інспектора об'єктів ці зміни зберігаються в двійковому файлі форми.

Front end

В програмній інженерії терміни «front end» та «back end» розрізняють за принципом розділення відповідальності між рівнем представлення та рівнем доступу до даних відповідно.

Front end – це інтерфейс для взаємодії між користувачем і back end. Front end та back end можуть бути розподілені між однією або кількома системами.

В програмній архітектурі може бути багато рівнів між апаратним забезпеченням та кінцевим користувачем. Кожен з цих рівнів може мати як front end, так і back end. Front – це абстракція, спрощення базового компоненту через надання користувачу зручного (user-friendly) інтерфейсу.

Користувацький інтерфейс додатку складеться з таких основних елементів:

- Сторінка входу (LoginPage)
- Сторінка рівнів (LevelsPage)
- Сторінка гри (MainPage)
- Сторінка рекордів(HighscoresPage)

Розмітка сторінок користувацького інтерфейсу знаходиться у файлах розмітки на мові XML.

Розширювана мо́ва розмітки (англ. Extensible Markup Language, скорочено XML) – запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML. XML-документ складається із текстових знаків, і придатний до читання людиною.

Стандарт XML (англ. Recommendation, перше видання від 10 лютого 1998, останнє, четверте видання 29 вересня 2006) визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів. Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначаєметамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтованімови розмітки даних. Ці обмеження описуються мовами схем (англ. Schema), такими як XML Schema (XSD), DTD або RELAX NG. Прикладами мов, заснованих на XML,

є: XSLT, XAML, XUL, RSS, MathML, GraphML, XHTML, SVG, а також XML Schema.

Фізична структура

Сутності (англ. Entity). Головною сутністю є зміст документа. Інші можливі сутності вказуються за допомогою

- Посилання на сутності (&назва; в самому документі, та, наприклад %назва; у визначені його типу) можуть слугувати в ролі позначень спеціальних символів, посилань на спеціальні символи (вказуючи коди символів &#десяткове;, або &#хшістнадцяткове;) або окремих документів чи фрагментів тексту.

- XML-декларація, в ній вказується версія XML, кодування та інша допоміжна інформація.

- Декларація типу документа може застосовуватись для того, аби додавати нові типи сутностей та визначати логічну структуру документа.

Логічна структура

XML-документ має ієрархічну логічну структуру, і може представлятись у вигляді дерева. Вузлами цього дерева можуть бути:елементи, фізична структура яких складається із коректної пари відкриваючого та закриваючого тегів (<Назва-тега>) та (</Назва-тега>), аботега порожнього елемента (<Назва-тега />), атрибути, що мають вигляд пар ключ-значення (назва атрибута="значення атрибута") і знаходяться або у відкриваючому, або у порожньому тезі (подібно до метаданих), вказівки щодо обробки документа (англ. Processing Instruction) (<?Обробник параметр ?>), коментарі (<!-- Текст коментаря -->), текст, або у вигляді простого тексту, або фрагментів CDATA (<![CDATA[довільний текст]]>).

XML-документ повинен мати лише один кореневий елемент. Решта елементів є піделементами цього кореневого елемента.

Деякі веб-браузери здатні безпосередньо відображати XML-документи. Це може досягатись шляхом застосування таблиці стилів (англ. Stylesheet). Вказані у таблиці стилів операції можуть призводити до перетворення XML-документа в інший, відмінний від XML формат.

Обробка XML-документів

До XML-обробки відносяться форматування, синтаксичний аналіз, редагування, перевірка коректності і перетворення в інші формати.

До традиційних технологій обробки XML-документів належать такі три технології:

- написання програм на мові програмування із використанням API SAX;
- написання програм на мові програмування із використанням API DOM;
- застосування механізму перетворення та фільтра;

До новіших технологій, що почали здобувати поширення останнім часом належать:

- активний аналіз;
- зв'язування даних;

Файли розмітки екранів мають формат .axml.

Посилання на ресурси з XML

Ресурси у XML-файлі доступні спеціальним синтаксисом:

- @ [<PackageName>:] <ResourceType> / <ResourceName>.

PackageName – пакет, який надає ресурс і потрібний лише тоді, коли використовуються ресурси з інших пакетів.

ResourceType – це вкладений тип ресурсу, який знаходиться в класі ресурсів.

Ім'я ресурсу – це назва файлу ресурсу (без розширення типу файлу) або значення атрибуту android: для ресурсів, що знаходяться в елементі XML.

- Модель, вид, ViewModel (MVVM) – шаблон Model-View-ViewModel користується популярністю в рамах, що підтримують зв'язування даних, таких як Xamarin.Forms. Його популяризували SDK із підтримкою XAML, такі як Windows Presentation Foundation (WPF) та Silverlight; де ViewModel діє як перехід між даними (Модель) та користувальницьким інтерфейсом (Перегляд) за допомогою зв'язування даних та команд.

- Model, View, Controller (MVC) – загальний і часто неправильно витлумачений шаблон, MVC найчастіше використовується при побудові користувальницьких інтерфейсів і забезпечує відокремлення між фактичним визначенням інтерфейсу екрану (View), двигуна за ним, який обробляє взаємодію (Контролер) та дані, які його населяють (Модель). Модель насправді є цілком необов'язковою частиною, і тому основою розуміння цієї моделі лежить в області перегляду та контролера. MVC - популярний підхід для програм iOS.

- Business Facade – шаблон AKA Manager, забезпечує спрощену точку входу для складної роботи. Наприклад, у програмі відстеження завдань може бути клас TaskManager з такими методами, як GetAllTasks (), GetTask (taskID), SaveTask (завдання) тощо. Клас TaskManager забезпечує Фасад для внутрішньої роботи з фактичного збереження / вилучення завдань об'єктів.

- Синглтон – шаблон Singleton забезпечує спосіб, у якому може існувати лише один екземпляр певного об'єкта. Наприклад, при використанні SQLite у мобільних додатках ви хочете лише один екземпляр бази даних. Використання моделі Singleton – це простий спосіб це зробити.

- Постачальник – шаблон, створений Microsoft (можливо, схожий на стратегію або базову ін'єкцію залежностей) для заохочення повторного використання коду через додатки Silverlight, WPF та WinForms. Спільний код може бути написаний проти інтерфейсу або абстрактного класу, а специфічні для платформи конкретні реалізації записуються і передаються, коли код використовується.

- Async не слід плутати з ключовим словом Async, шаблон Async використовується, коли довготривалу роботу потрібно виконати, не затримуючи інтерфейс користувача або поточну обробку. У найпростішому вигляді шаблон Async просто описує, що довготривалі завдання повинні починатися в іншому потоці (або аналогічній абстракції потоку, наприклад, у завданні), тоді як поточна нитка продовжує оброблятись і слухає відгук із фонового процесу; потім оновлює інтерфейс, коли повертаються дані та / або стан.

Взаємодія з користувачем

Універсальний додаток для Windows дозволяє вам скористатися унікальними можливостями пристрою, на якому воно працює. Ваша програма може використовувати всі можливості настільного пристрою, природна взаємодія при безпосередній роботі на планшеті (включаючи сенсорне введення і введення за допомогою пера), портативність і зручність мобільних пристроїв, функції спільної роботи Surface Hub і інших пристроїв, які підтримують програми UWP.

Гарне проектування має на меті визначення взаємодії користувачів і додатки, а також зовнішнього вигляду і функцій. Взаємодія з користувачем грає величезну роль в задоволеності людей при використанні вашого застосування, тому ретельно опрацювати цей крок. У розділі Основи проектування ви познайомитеся з оформленням універсального застосування для Windows. У розділі Знайомство з додатками універсальної платформи Windows (UWP) для дизайнерів ви дізнаєтеся про створення додатків для Windows, які сподобаються вашим користувачам. Перш ніж приступити до написання коду, ознайомтеся з абеткою пристроїв, яка допоможе визначити інтерфейс взаємодії вашої програми на всіх форм-факторах, на які ви орієнтуєтеся.

Пристрої під керуванням Windows

На додаток до взаємодії на різних пристроях, створіть план додатки, щоб скористатися перевагами роботи на кількох пристроях. Приклад.

Використовуйте хмарні служби, щоб виконувати синхронізацію між пристроями. Дізнайтеся, як підключитися до веб-службам для підтримки взаємодії з вашим додатком. Подумайте, як можна забезпечити підтримку користувачів, які переходять с одного пристрою на інший, як підхоплювати елементи, на яких вони зупинилися. Включіть до свого планування повідомлення і покупки з додатків. Ці функції повинні працювати на всіх пристроях. Спроектуйте робочий процес за допомогою розділу Основи проектування навігації для додатків UWP, щоб адаптувати мобільні пристрої і пристрої з маленьким і великим екранами. Розробіть свій користувальницький інтерфейс для різних розмірів і дозволів екрану. Подумайте, чи є в вашому додатку функції, які не мають сенсу на невеликому екрані мобільного пристрою. Можливо також є області, які не мають сенсу на стаціонарному настільному комп'ютері, і потрібні для роботи мобільного пристрою. Наприклад, більшість сценаріїв, які використовують розташування, припускають мобільний пристрій. Вирішіть, яким чином ви розмістите кілька типів введення. Ознайомтеся з Керівництвом по взаємодії, щоб дізнатися, як користувачі можуть взаємодіяти з вашим додатком за допомогою Кортан, голосових функцій, розпізнавання сенсорного введення, сенсорної клавіатури і інших можливостей. Ознайомтеся з Керівництвом по тексту і текстовим введення, щоб дізнатися більше про традиційні способи взаємодії.

Макет

Оскільки додатки UWP автоматично масштабуються на всіх пристроях, розробка додатка UWP для будь-якого пристрою виконується за такою ж структурою. Почнемо з самого початку вашого інтерфейсу програми UWP.

Windows, рамки та сторінки

Коли пристрій UWP запускається на будь-якому пристрої Windows 10, він запускається у вікні з фреймом, який може переходити між екземплярами сторінки.

Ви можете думати про інтерфейс вашого додатка як набір сторінок. Ви вирішите, що повинно йти на кожній сторінці, а також співвідношень між сторінками.

Макет сторінки

Як виглядатимуть ці сторінки? Щоправда, більшість сторінок використовують спільну структуру для забезпечення послідовності, тож користувачі можуть легко переходити між сторінками вашого додатка та в межах них. Сторінки, як правило, містять три типи елементів інтерфейсу:

- навігаційні елементи допомагають користувачам вибирати вміст, який вони хочуть переглянути;
- елементи команд ініціюють дії, такі як маніпулювання, збереження або спільне використання вмісту;
- елементи вмісту відображають вміст додатка.

Щоб дізнатись більше про те, як реалізувати загальні шаблони додатків UWP, перегляньте статтю макета сторінки.

Ви також можете скористатися Studio Template Studio у Visual Studio, щоб почати макет для свого додатка.

Елементи керування

Конструкторська платформа UWP забезпечує набір спільних елементів керування, які гарантовано працюють на всіх пристроях з підтримкою Windows, і дотримуються принципів нашої Системи вільного дизайну . Ці елементи керування включають все, починаючи від простих елементів керування, таких як кнопки та текстові елементи, до складних елементів керування, які можуть генерувати списки з набору даних і шаблону.

Повний список елементів керування UWP та шаблони, які ви можете зробити з них, див. У розділі елементів керування та шаблонів.

Стиль

Спільні елементи керування автоматично відображають системну тему та колір акценту, працюють з усіма типами вводу та масштабуються на всіх пристроях. Таким чином, вони відображають Систему вільного дизайну - вони є адаптивними, відчутливими та красивими. Спільні елементи керування використовують світло, рух та глибину за стилями за замовчуванням, тому, використовуючи їх, ви використовуєте нашу систему вільного дизайну у своєму додатку.

Загальні елементи керування також настроюються, ви можете змінити колір переднього плану елемента керування або повністю налаштувати його зовнішній вигляд. Щоб переопределити стилі за замовчуванням в елементах керування, використовуйте легкий стиль або створюйте спеціальні елементи керування в XAML.

Back end

Написання коду

Варіанти мови програмування для вашого проекту Windows 10 в Visual Studio: Visual C ++, C #, Visual Basic і JavaScript. В Visual C ++, C # і Visual Basic можна використовувати XAML для високоякісного і природного взаємодії з призначеним для користувача інтерфейсом. В Visual C ++ ви можете вибрати DirectX замість використання XAML або разом з ним. В JavaScript вашим рівнем представлення даних буде HTML, а HTML є кросплатформним веб-стандартом. Велика частина коду і призначеного для користувача інтерфейсу будуть універсальними і будуть працювати однаково на будь-яких пристроях. Але для коду, написаного для конкретних сімейств пристроїв, і призначеного для користувача інтерфейсу, розробленого для певних форм-факторів, вам буде запропоновано використовувати адаптивний код і адаптивний користувальницький інтерфейс. Розглянемо ці випадки.

Виклик API-інтерфейсу, реалізованого цільовим сімейством пристроїв

При виклику API в додатку UWP необхідно знати, реалізований чи API-інтерфейс тим сімейством пристроїв, на яке орієнтоване вашу програму. Visual Studio Intellisense відображає тільки API-інтерфейси, які доступні для вибраного розширення SDK. Якщо ви не вибрали розширення SDK, ви побачите тільки API-інтерфейси, доступні для універсального сімейства пристроїв.

У документації по API також описано, якою частиною сімейства пристроїв є API-інтерфейс. У розділі "Вимоги" зазначено, що таке реалізовується сімейство пристроїв і яка версія цього сімейства пристроїв відображається в API-інтерфейсі.

Виклик API-інтерфейсу, НЕ реалізованого цільовим сімейством пристроїв. Сюди відносяться випадки, коли необхідно викликати API-інтерфейс в розширенні SDK, на яке ви посилалися, але API не є частиною сімейства пристроїв, на які ви орієнтовані. Наприклад, ви можете бути орієнтовані на сімейство універсальних пристроїв, але мати класичний API-інтерфейс для використання під час запуску програми на мобільному пристрої. В цьому випадку для вас може виявитися кращим написання адаптивного коду для виклику цього API-інтерфейсу.

Об'єктно-орієнтооване програмува́ння (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція

Об'єктно-орієнтоване програмування – це метод програмування, заснований на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

- все є об'єктами;
- всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення – це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії;
- кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів;
- кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів;
- у класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії;
- класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

В результаті дослідження Дебори Дж. Армстронг (англ. Deborah J. Armstrong) комп'ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об'єктно-орієнтованого програмування. До них належить.

Клас

Клас визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Наприклад, клас Собака може

характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Як правило, клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами

Об'єкт

Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об'єкта називаєтьсястаном. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класуСобака.

Метод

Можливості об'єкта. Оскільки Сірко – Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

<u>Успадкування</u> (наслідування)

Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка і т.п. Так,Сірко може бути екземпляром класу Вівчарка. Підкласи *успадковують* атрибути та поведінку своїх

батьківських класів, і можуть вводити свої власні.Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в C++ і те і інше.

Приховування інформації (інкапсуляція)

Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися дочленів об'єкта. Як наслідок, кожен об'єкт представляє кожному іншому класупевний інтерфейс – члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дозволить полегшити внесення змін, тобто, без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейсможе гарантувати, що щенята можуть додаватись лише до об'єктів класу Собака кодом самого класу.

Часто, члени класу є як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: Java використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected – лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь-яких членів.

Абстрагування

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі – як Коллі і як Тварина (можливо, батьківський клас Собака) при підрахунку тварин Петра.

Поліморфізм

Лямбда-вираз в програмуванні – спеціальний синтаксис для визначення функціональних об'єктів, запозичений з λ-числення. Застосовується як правило для оголошення анонімних функцій за місцем їх використання, і зазвичай допускає замикання на лексичний контекст, в якому цей вислів використано. Використовуючи лямбда-вирази, можна оголошувати функції в будь-якому місці коду. Лямбда-вирази приймають дві форми. Форма, яка найбільш прямо замінює анонімний метод, являє собою блок коду, укладений у фігурні дужки. Це – пряма заміна анонімних методів. Лямбда-вирази, з іншого боку, надають ще більш скорочений спосіб оголошувати анонімний метод і не вимагають ні коду в фігурних дужках, ні оператора return. Обидва типи лямбда-виразів можуть бути перетворені в делегати.У всіх лямбда-виразах використовується лямбда-оператор =>, який читається як «переходить в" (в мовах Java, F # і PascalABC.NET використовується оператор ->). Ліва частина лямбда-оператора визначає параметри введення (якщо такі є), а права частина містить вираз або блок оператора. Лямбда-вираз x => x * 5 читається як «функціях, яка переходить в x, помножене на 5».У даному додадку дані (опитування) виводяться у вигляді карток. Кожна картка містить назву опитування, гіперсилку на користувача, що додав це опитування та взалежності проголосував користувач чи ні, варіанти у вигляді рівнів або радіо кнопок.

Асинхронне програмування

Асинхронне програмування – це форма паралельного програмування, яка дозволяє працювати окремо від основного потоку додатків. Коли робота завершена, вона повідомляє про основний потік (а також про те, чи була робота завершена або не виконана). Існує безліч переваг від його використання, наприклад, покращена продуктивність роботи та покращена чуйність.Для вирішення задач, пов'язаних з вводом-висновком (наприклад, для запиту даних з мережі або доступу до бази даних), бажано використовувати асинхронне програмування. Якщо у вас є код, обмежений ресурсами процесора, наприклад, виконує складні розрахунки, то це також підходить сценарій для асинхронного програмування. В C # є модель асинхронного програмування, реалізована на рівні мови, яка дозволяє легко писати асинхронний код, не використовуючи зворотний виклик або бібліотеки, які підтримують асинхронність. Вона будується на принципах асинхронної моделі на основі задач (TAP). Загальний огляд асинхронної моделі. В основі асинхронного програмування лежать об'єкти Task та Task <T>, які моделюють асинхронні операції. Вони підтримуються ключовими словами async і await. В більшості випадків модель досить проста.

Для коду, обмеженою продуктивністю введення-виведення, за допомогою await виконується операція, яка повертає об'єкт Task або Task <T> всередині методу async.

Для коду, обмеченного ресурсами процесора, за допомогою await виконується операція, яка запускається в фоновому потоці методом Task.Run.

Іменно за допомогою ключевого слова очікування твориться вся магія. Він передає керування виклику об'єкту методу, який виконував очікування, дозволяючи тим самим користувачеві інтерфейсу або службі відповісти на запити.

Есть також і інші подходы к написанию асинхронного кода без использования ключевых слов async і await. Їх опис можна знайти в приведеній

вище статті, присвяченій ТАП. Однак далі в цьому документі будуть розглянуті конструкції на рівні мови.

Ключові моменти для розуміння:

- асинхронний код можна використовувати як за обмеженою продуктивністю введення-виведення, так і при обмежених ресурсах процесора, але по-різному в кожному випадку;

- в асинхронному коді використовуються конструкції Task <T> і Task, які служать для моделювання задач, що виконуються в фоновому режимі;

- ключове слово async робить метод асинхронним, що дозволяє використовувати в його тілі ключове слово await;

- коли застосовується ключовий слово, очікується, він призупиняє виконання виклику методу і передає керування зворотним виклику об'єкту, поки не буде завершена очікувана задача;

- await можна використовувати тільки в асинхронному методі;

Збереження даних користувача

У додатку необхідно зберігати дані у двох випадках:

- тимчасова інформація для коректної роботи гри (теперішній рівень, рахунок);

- дані рекордів користувача.

Розглянемо кожен з них з точки зору програмної реалізації.Для зберігання даних у розрізі сесії роботі додатку у мові C# прийнято використовувати статичні типи даних.

Статичні класи та члени статичних класів

Статичний клас в основному такий же, як і нестатичний клас, але існує одна відмінність: не можна створювати екземпляри статичного класу. Другими словами, нельзя використовувати ключове слово new для створення змінної типу класу. Оскільки немає змінної екземпляри, доступ до членів статичного класу здійснюється з використанням самого імені класу.

Статичний клас може використовуватися як звичайний контейнер для наборів методів, що працюють на вхідних параметрах, і не повинен повертати або встановлювати які-небудь внутрішні поля екземпляра. Наприклад, в бібліотеці классов .NET Framework статичний класс System.Math містить методи, що виконують математичні операції, без збереження або вилучення даних, унікальних для конкретного экземпляра класу Math .

Как и в случае с типами всех классов, информация о типе для статического класса загружается средой CLR .NET Framework , когда загружается программа, которая ссылается на класс. Програма не може точно вказати, коли завантажується класс. Однак гарантується завантаження цього класу, ініціалізація його полів та викликів статичного конструктора перед першим зверненням до класу в програмі. Статичний конструктор викликається тільки один раз, і статичний клас залишається в пам'яті за час існування домену програми, в якому знаходиться програма.

Нижче наведені основні можливості статичного класу.

- содержит тільки статичні члени;
- создавать его экземпляры нельзя;
- являється запечатаним;
- не може вмістити конструктори екземплярів.

По суті, створення статичного класу аналогічно створенню класу, що містить тільки статичні члени та закритий конструктор.Закритий конструктор не допускає створення екземплярів класу. Перевага застосування статичних класів полягає в тому, що компілятор може перевірити відсутність випадково доданих членів екземплярів. Таким чином, компілятор гарантує неможливість створення екземплярів подібних класів.

Статичні класи запечатані, тому їх не можна наследовати. Вони не можуть наследовати ніяких класів, крім об'єкта. Статичні класи не можуть містити конструктор екземплярів, але можуть містити статичний

конструктор. Нестатичні класи також повинні визначати статичний конструктор, якщо клас містить статичні члени, для яких потрібна нетривіальна ініціалізація. Додаткові відомості см. в розділі Статичні конструктори.

Нестатичний клас може містити статичні методи, поля, властивості або події. Статичний член викликається для класу навіть у тому випадку, якщо не створено екземпляр класу. Доступ до статичних членів завжди виконується за іменем класу, а не екземпляра.Существует тільки одна копія статичного члена, незалежно від того, скільки створено екземпляри класу. Статичні методи та властивості не можуть звертатися до нестаціонарних полів і подій в їх містовому типі, і вони не можуть звертатися до змінної екземпляру об'єкта, якщо він не передається явно в параметрі методу. Більше прикольно оголошення нестатичного класу з кількома статичними членами, ніж оголошення всього класу як статичного.Статичні поля звичайно використовуються для наступних двох цілей: зберігання числа чисельних об'єктів або збереження значення, яке спільно використовується всіма екземплярами. Статичні методи можуть бути перевантажені, але не переопределені, оскільки вони відносяться до класу, а не до екземпляру класу.

Незважаючи на те, що поле не може бути оголошено як static const , поле const за своєю поведінкою є static const. Він відноситься до типу, а не до екземплярам типу. Тому к полям const можна звернутися з використанням того ж нотації ClassName.MemberName , що і для статичних полів. Экземпляр объекта не требуется.

C # не підтримує статичні локальні змінні (змінні, оголошені в області дії методу).

Виклик статичного методу генерує інструкцію виклику в проміжковій мові Microsoft (MSIL), в той час як виклик методу callvirt , яка також перевіряє наявність посилання на пусті об'єкти. Однак у більшості випадків різниця у продуктивності двох видів викликів несумлінна.

2.2 Опис програми та її алгоритмів

Готова програма дозволяє грати в гру «Що спільного?». Гравцю необхідно знайти за визначений термін часу спільну рису у картинок, які він бачить на екрані. Якщо гравець знаходить вірний рису, то при натиску ЛКМ на нього з'являється наступний рівень, де знайти спільну рису буде важче. Гра закінчується, коли гравець не встигає за певний відлік часу знайти спільну рису, або вводить не вірну відповідь.

Опис функцій системи

Функції системи зображені у вигляді UML діаграми прецедентів з одним актором – Гравець на рисунку 2.1.

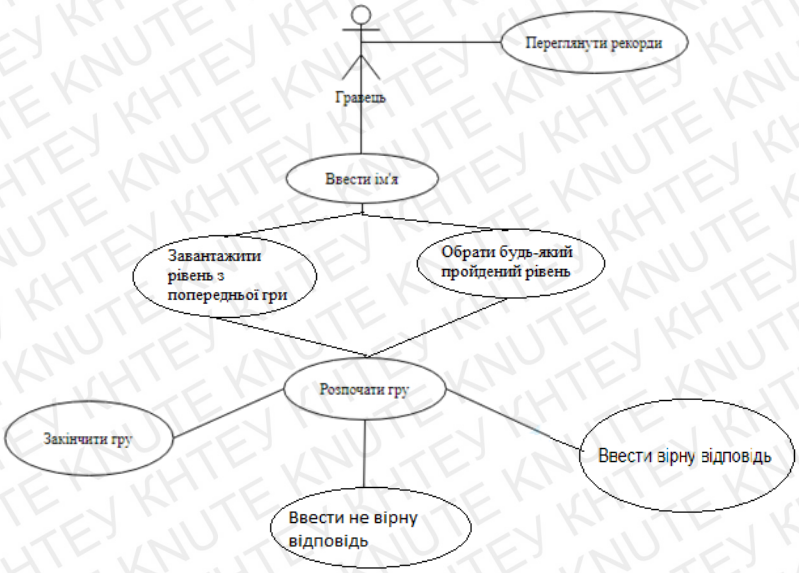

Рис 2.1 – UML діаграма прецедентів

Ця діаграма зображує, що може зробити гравець.

Гравець може:

1) Ввести ім'я, після чого запуститься гра
2) Вписати вірну відповідь
3) Вписати не вірну відповідь.
4) Добровільно закінчити гру.

5) Переглянути таблицю рекордів.

Опис ієрархії власних класів

В процесі написання гри було розроблено ієрархію класів представлену на рисунку 2.2.

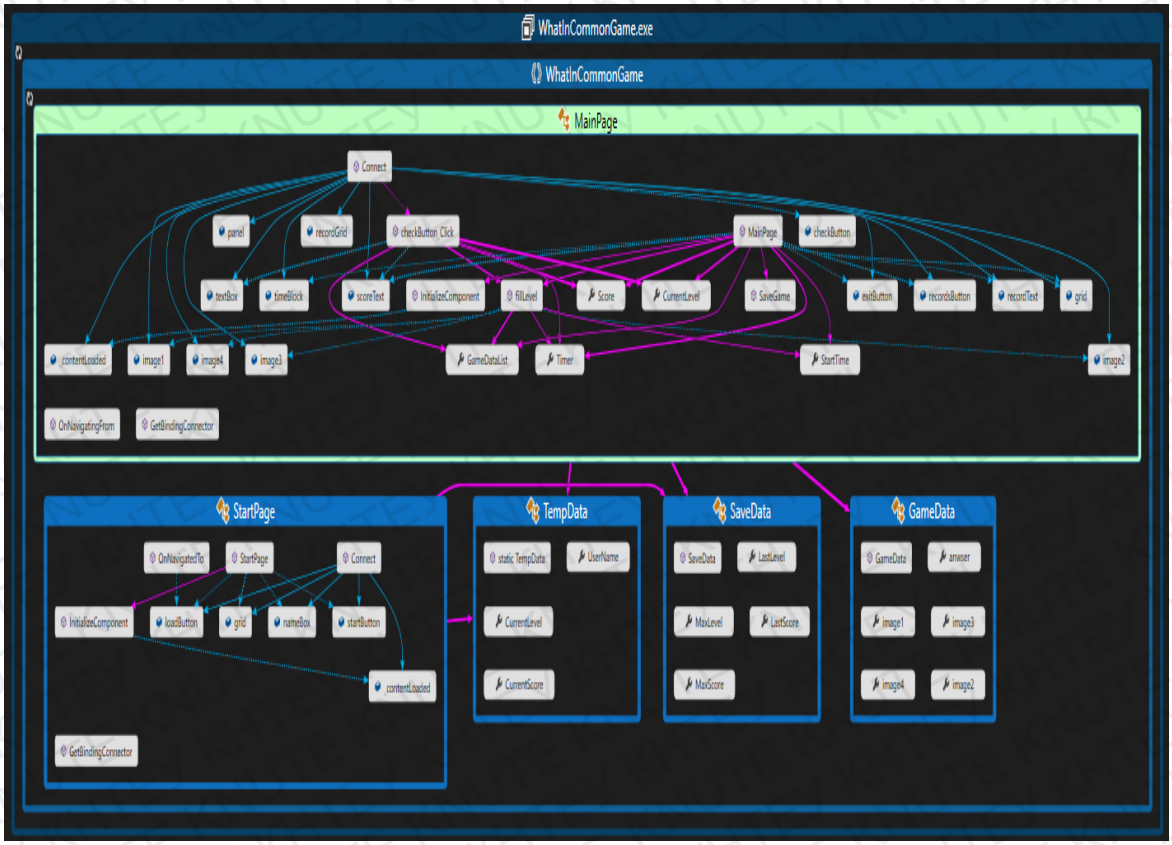

Рис 2.2 – Ієрархія класів

Нижче подано опис представлених на діаграмі класів.

Клас «MainPage» – ініціалізація об’єктів і властивостей, а також виклик функції завантаження першого рівня.

У таблиці 2.1 представлено опис методів класу.

Таблиця 2.1

Методи класу EnterName

Властивість	Опис

1	2
loadLevel()	Завантаження указаного рівня.
OnPointerRealeased()	Спрацьовує при натиску ЛКМ на «Check». Якщоправильна відповідь, то здійснюється перехід на наступний рівень. Якщо не правильна відповідь, то виведення повідомлення про програш.
OnClick()	Спрацьовує при вводі імені після запуску програми.
OnTick()	Відповідає за роботу таймера.
SaveToFile()	Записує ім’я гравця і рахунок до файлу.
RestartGame()	Гра починається з початку.
Властивість	Опис
RestartOnClick()	Спрацьовує при натиску кнопки Restart.
RecordOnClick()	Спрацьовує при натиску на кнопку Рекорди і виводить їх.

Опис компонентів

В результаті написання програми було створено компоненти, які є окремимими файлами. А саме:

1) Папка Assets. У цій папці за задумом ви повинні зберігати ресурси вашого застосування, такі як, наприклад, картинки. За замовчуванням, дана папка містить обов'язкові картинки-заплатки, які виступають в ролі логотипу, і про них я ще поговорю докладніше в одній зі статей даного циклу. До слова, вам не обов'язково зберігати ці та інші ресурси в цій папці. Як, власне, у вас немає обмежень по кількості папок і підпапок з ресурсами.

2) Файл App.xaml і пов'язаний з ним App.xaml.cs / vb виконують практично туж функцію, що і Application в WPF. У App.xaml ви можете прописати ресурси програми, а також вказати бажану тему (світла чи темна) у властивості RequestedTheme. У файлі коду ви можете управляти життєвим циклом додатка.

Тема життєвого циклу програм UWP досить обширна і докладніше про те, що це таке, можна почитати тут. Також файл App.xaml.cs / vb можна використовувати для розміщення глобальних змінних додатки, але це вже залежить від шаблону програмування, який ви вирішите використовувати. Наприклад, для UWP більше схожий MVVM, який я використовувати не буду, але досвідченим розробникам вкрай рекомендую.

3) Файл App1_TemporaryKey.pfx є тимчасовим сертифікатом, підписує ваш додаток, і за потрібне, щоб воно могло запускатися на пристрої. Більше знати нам про нього нічого не потрібно. Працювати з ним нам не доведеться.

4) Файл MainPage.xaml і пов'язаний з ним MainPage.xaml.cs / vb це основна сторінка додатка. Відмінність від додатків WPF полягає в тому, що тут ми працюємо не з вікнами, а зі сторінками. Вікно у додатки UWP найчастіше одне, і в ньому завантажуються сторінки додатка, між якими ви перемикаєтеся, подібно переходу по сторінках сайту в браузері. Детальніше про сторінки і навігації в UWP я розповім в одній з найближчих статей.

5) І останній, автоматично створений файл проекту, - це файл

Package.appxmanifest. Це, як ви зрозуміли, маніфест проекту, який містить основні оголошення. Цей файл має зручний редактор, який викликається подвійним клацанням миші. В даному редакторі можна виставити важливі параметри програми, задати розташування логотипів, задати можливості програми і оголосити контракти, які воно буде використовувати. По ходу цього циклу я регулярно буду звертатися до різних налаштувань маніфесту і більш детально їх розглядати. Також зауважу, що в документації часто зустрічається вказівку, в яких параметрах маніфесту які ключі повинні бути відзначені. У цих випадках документація посилається на текстову версію маніфесту. Відкрити її можна, якщо натиснути правою кнопкою миші на файл маніфесту і вибрати пункт «Відкрити за допомогою» і далі – «Редактор (текстовий) XML».

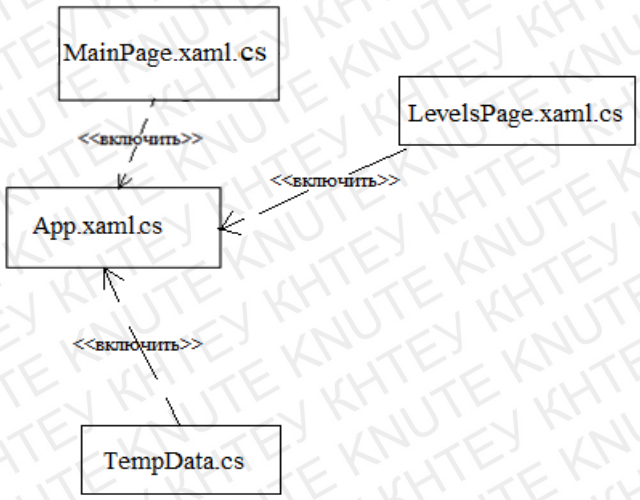

Рис. 2.3 – Діаграма компонентів.

Опис алгоритму

Користувач вводить спільну рису.

Якщо відповідь була введена правильна, то він переходить на наступний рівень, якщо ні, то виводиться повідомлення про поразку і пропонується почати гру спочатку.

На рисунку 2.4 зображено циклічний, розгалужений алгоритм роботи програми.

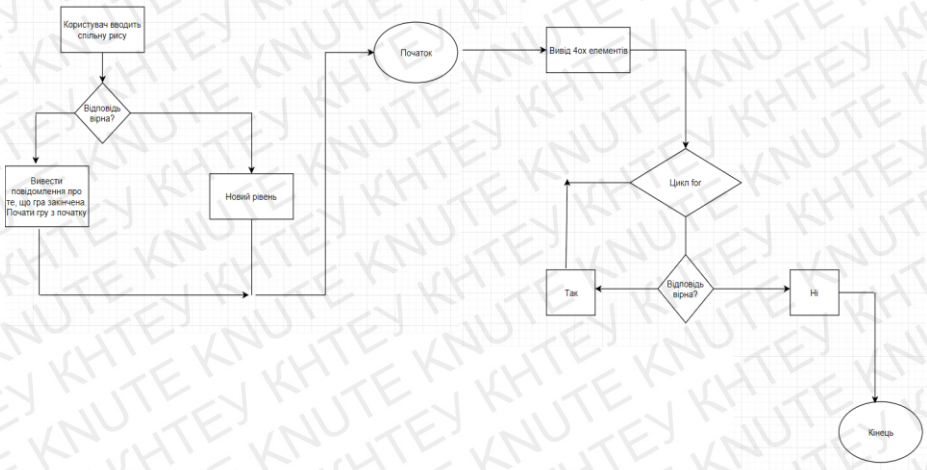

Рис. 2.4 – Алгоритм роботи програми

2.3. Інструкція програміста

Проект являє собою гру для будь-якого пристрою під керуванням ОС Windows.

- Автор: Макогонюк Максим Олегович
- Мова програмування: C++
- Середовище програмування: Microsoft Visual Studio 2017
- Вимоги до апаратного забезпечення:
- Процесор – Intel Pentium x64, не нижче 300 MHz
- Обсяг ОЗУ – не нижче 64 Мб
- Відеокарта: NVIDIA Geforce GT 630M. Вільний простір на диску - 100 мб
- Монітор з роздільною здатністю не менше 800x600, TrueColor
- Клавіатура – Windows-сумісна
- Наявність комп'ютерної миші або touch screen
- Вимоги до апаратного забезпечення:
- Операційна система - 7 / 8 / 8.1 / 10

2.4. Інструкція оператора

Для запуску гри потрібно перейти до папки з її проектом і натиснути ЛКМ x2 на WhatInCommonGame.exe.

Після запуску проекту натиснути Run->Run.

Гра запуститься і буде відображено вікно авторизації, у якому запропоноване випадкове ім'я для спрощення авторизації, проте у користувача залишається можливість ввести своє ім'я, далі необхідно натиснути Start game, щоб вибрати рівень, або Load last game для того, щоб запустити останній пройдений рівень, як зображено на рисунку 2.5.

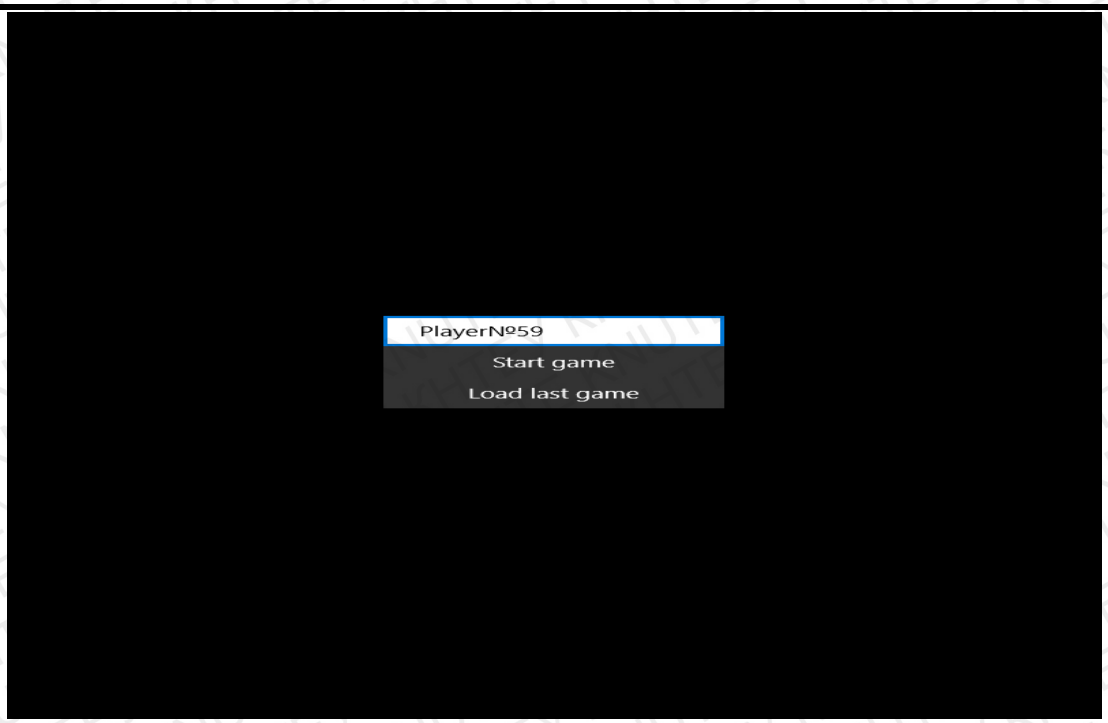

Рис. 2.5 – Вікно авторизації після запуску програми

Після того, як ім’я було введене і була натиснута кнопка «Start game користувачу надається можливість обрати рівень з якого розпочнеться, як показано на рисунку 2.6.

Рис. 2.6 – Вибір рівнів

Після того, як гравець натисне на рівень, з якого хоче розпочати, гра почнеться.

Кількість набраних очок, за проходження певної кількості рівнів. Приклад на рисунку 2.7

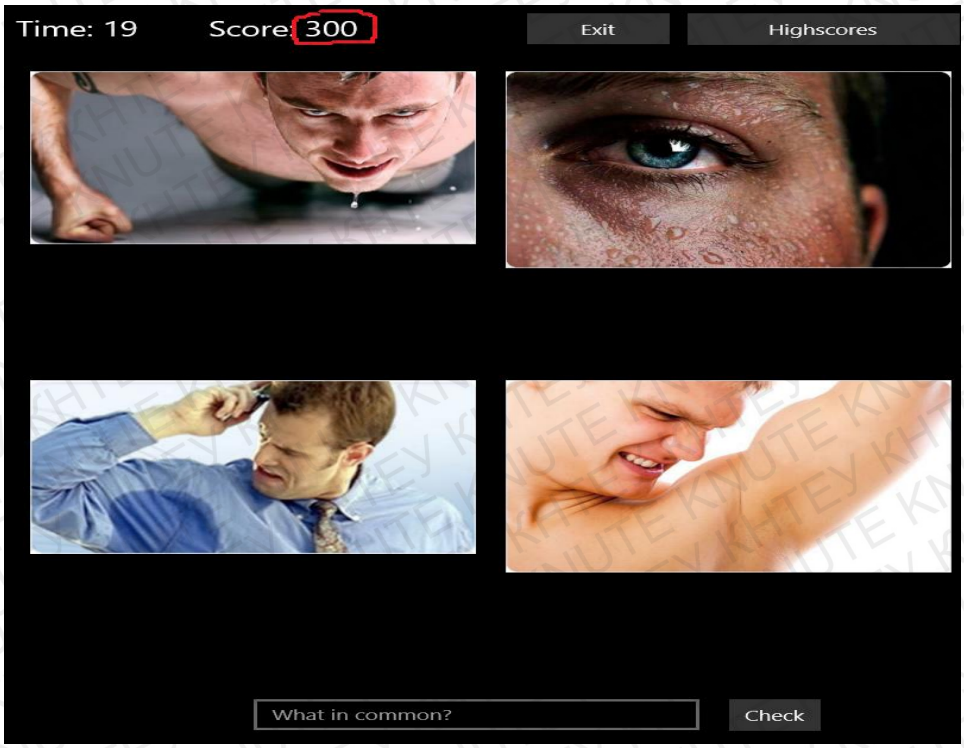

Рис. 2.7 – Кількість зібраних очок за гру.

На рисунку 2.8 показано, як зростає складність відповідно до обраного рівня.

Рис. 2.8 – Складність зростає з кожним рівнем

Також в грі реалізований таймер і на проходження кожного рівня дається лише 20 секунд, як зображено на рис. 2.9

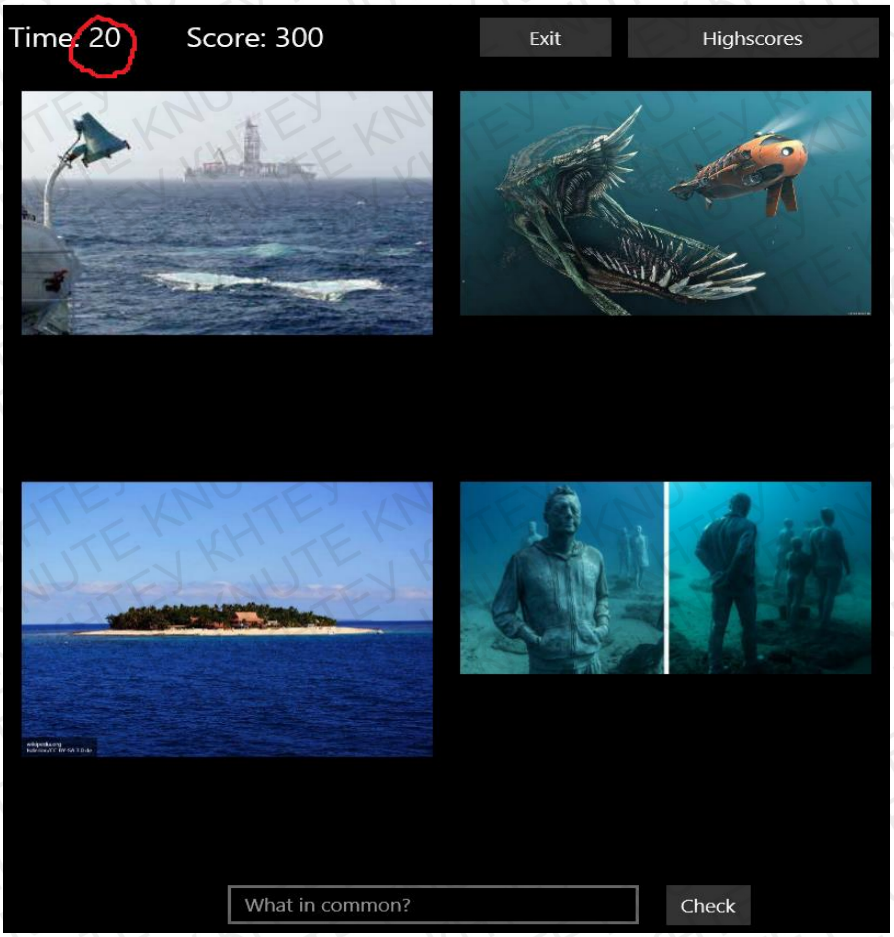

Рис. 2.9 – Таймер

Якщо гравець натискає на не правильний елемент, тобто сумний смайл, або не встигає знайти веселий смайл і час виходить, то гра закінчується і з'являється повідомлення про програш, як показано на рис. 2.10.

Рис 2.10 – Програш

Коли гравець закінчив гру- його ім’я і рахунок заноситься до списку рекордів, який можна переглянути, натиснувши на кнопку «Highscores» у верхньому правому кутку вікна. Приклад зображено на рисунку 2.11 і 2.12.

Рис 2.11 – Кнопка Highscores.

Рис. 2.12 – Список рекордів

Під час створення гри були використані наступні класи та функції реалізовані в них(у вигляді UML - діаграмми):

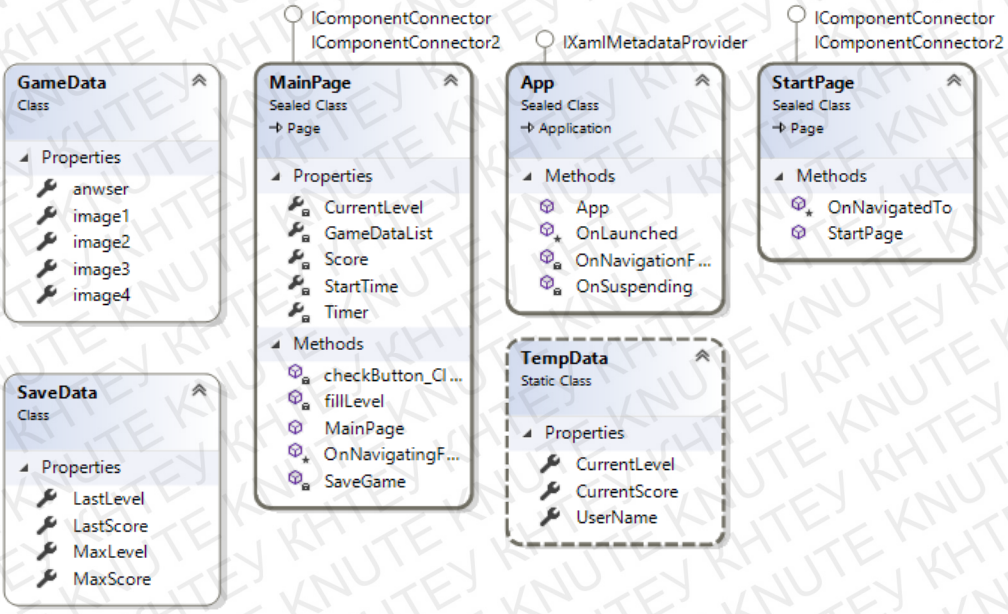

Рис. 2.13 – UML-діаграма

Висновок до розділу 2

У другому розділі була проведена робота по створенню програмного додатку, розробки його інтерфейсу та функцій, які стануть фішкою данного проекту, також наведена інструкція користувача, у якій описані усі функції додатку.

РОЗДІЛ 3.
ОРГАНІЗАЦІЙНО - ЕКОНОМІЧНА ЧАСТИНА

3.1 Оцінка вартості програмного продукту

У даному розділі дипломної роботи проводиться розрахунок затрат на розробку розширеної версії гри *«What is in common»*. Розробка гри вимагає певних інтелектуальних і трудових витрат, а також обов'язкового використання комп'ютерної техніки, що визначає особливості розрахунку собівартості програмного продукту. Оцінка вартості програми розглядається як оцінка затрат коштів і часу, необхідних для виконання етапів проекту.

3.1.1 Розрахунок часу

Загальний час на створення програми складається з різних компонентів. Структура загального часу на створення програмного продукту представлена в таблиці 3.1.

Таблиця 3.1

Структура загального часу на створення програмного продукту

№ етапу	Позначення часу даного етапу	Зміст етапу
1	**2**	**3**
1	$T_{ПО}$	Підготовка опису завдання
2	$T_{О}$	Опис завдання
3	$T_{А}$	Розробка алгоритмів
4	$T_{БС}$	Розробка блок-схеми алгоритмів
5	$T_{ПІ}$	Проектування інтерфейсу
6	$T_{Н}$	Написання програми (кодування)
7	$T_{ВТ}$	Відладка і тестування програми
8	$T_{Д}$	Оформлення документації, інструкції користувачеві, записки пояснення

Час розраховується в людино-годинах, причому $T_{ПО}$ та $T_Д$ береться по фактично відпрацьованому часу, а час останніх етапів визначається розрахунковий по умовному числу команд Q.

Умовне число команд Q визначається за формулою

$$Q = q \cdot з, \qquad (3.1)$$

де q – коефіцієнт, що враховує умовне число команд залежно від типу завдання. Значення коефіцієнта q вибирається з таблиці 3.2.

Значення коефіцієнта з вибирається з таблиці 3.3.

Таблиця 3.2

Значення коефіцієнта q

Тип завдання	Значення коефіцієнта q
1	**2**
Завдання обліку	1400…1500
Завдання оперативного управління	1500…1700
Завдання планування	3000…3500
Багатоваріантні завдання	4500…5000
Комплексні завдання	5000…5500

Для даного завдання коефіцієнт q приймається = 1650

Програмні продукти за мірою новизни можуть бути віднесені до однієї з чотирьох груп:

- група А – розробка принципово нових завдань;
- група Б – розробка оригінальних програм;
- група В – розробка програм з використанням типових рішень;
- група Г – разове типове завдання.

Для даного завдання міра новизни: В

За складністю програмні продукти можуть бути віднесені до однієї з трьох груп:

- 1 – алгоритми оптимізації і моделювання систем;

- 2 – завдання обліку, звітності і статистики;

- 3 – стандартні алгоритми.

Дане завдання може бути віднесено до другої групи складності низького рівня.

Коефіцієнт **(** з **)** визначається з таблиці 3.3 в залежності від складності і міри новизни.

Таблиця 3.3

Значення коефіцієнта з

Мова програмування	**Група складності**	**Міра новизни**			
		А	**Б**	**В**	**Г**
Високого рівня	1	1,38	1,15	1,15	0,69
	2	1,30	1,19	1,08	0,65
	3	1,20	1,10	1,00	0,60
Низького рівня	1	1,58	1,45	1,32	0,79
	2	1,49	1,37	1,24	0,74
	3	1,38	1,26	1,15	0,69

Для даного завдання коефіцієнт з = 1,00.

За формулою 3.1 визначається умовне число команд Q.

Q = 1650 · 1,00 = 1650,00 чол / годин.

Визначається час, витрачений на кожен етап створення програмного продукту:

- $T_{ПО}$ (час на підготовку опису завдання), береться по факту і для даного завдання

$T_{ПО}$ = 25 чол / годин.

- $T_О$ (час на опис завдання) визначається за формулою

$$T_О = Q \cdot B / (50 \cdot K), \qquad (3.2)$$

де В - коефіцієнт обліку змін завдання. Коефіцієнт В залежно від складності завдання і числа змін вибирається в інтервалі від 1,2 до 1,5. Для даного завдання В = 1,4

К - коефіцієнт, що враховує кваліфікацію програміста. Значення коефіцієнта вибирається з таблиці 3.4.

Таблиця 3.4

Значення коефіцієнта К

Стаж програміста	Значення коефіцієнта К
1	2
до 2-х років	0,8
від 2 до 3 років	1,0
від 3 до 5 років	1,1…1,2
від 5 до 10 років	1,2…1,3
понад 10 років	1,3…1,5

В даному випадку коефіцієнт К = 1,0.

За формулою 3.2 підраховується час на опис завдання.

То = 1650,00 *1,4/(50*1,0) = 46,2 [чол. / годин].

- T_A (час на розробку алгоритмів) розраховується за формулою

$$T_A = Q / (50 * K), \quad (3.3)$$

За формулою 3.3 підраховується час на розробку алгоритмів.

T_A =1650,00 / (50 · 1,0) = 33[чол. / годин].

- $T_{БС}$ (час на розробку блок-схеми) визначається аналогічно ТА за формулою 3.3 і складає

$T_{БС}$ = 1650,00 / (50 · 1,0) = 33[чол. / годин].

- T_H (час написання програми на мові програмування) визначається за формулою

$$T_H = Q \cdot 1,5 / (50 \cdot K), \quad (3.4)$$

За формулою 3.4 підраховується час написання програми на мові програмування.

Тн = 1650,00 *1,5/(50*1,0)=49,5 [чол. / годин].

- $Т_{ПІ}$ (час на проектування інтерфейсу) визначається за формулою

$$Т_{ПІ} = Q / 50, \quad (3.5)$$

За формулою 3.5 підраховується час на проектування інтерфейсу програми.

$Т_{ПІ}$ = 1650 / 50= 33 [чол. / годин].

- $Т_{ВТ}$ (час відладки і тестування програми) визначається за формулою

$$Т_{ВТ} = Q \cdot 4,2 / (50 \cdot К), \quad (3.6)$$

За формулою 3.6 підраховується час відладки і тестування програми.

Твт = 1650,00 ·4,2 / (50 · 1,0) = 138,6 [чол. / годин].

- $Т_Д$ (час на оформлення документації), береться по факту і для даного завдання

$Т_Д$ = 25 чол / годин.

Підраховується загальний час на створення програмного продукту за формулою

$$Т = Т_{ПО} + Т_О + Т_А + Т_{БС} + Т_Н + Т_{ПІ} + Т_{ВТ} + Т_Д$$

Т = 25 + 46,2 + 33 + 33 + 48,8 + 49,5 + 138,6 + 25= 399,1 [чол. / годин].

3.1.2 Розрахунок заробітної плати виконавця робіт із створення програмного продукту

Основна ЗП визначається за формулою

$$ЗПосн.=(ЗП\ 1р \cdot Кт \cdot Т)/(Чр \cdot tр.д.)\cdot(1 + П/100), \quad (3.8)$$

де ЗП 1р - місячна зарплата 1-го розряду, грн.;

Кт – тарифний коефіцієнт, відповідний розряду тарифної сітки, за яким працює виконавець;

Т - загальний час на створення програмного продукту, чол. / годин;

Чр - кількість робочих днів в місяць;

tр.д. - тривалість робочого дня в годинах;

П - відсоток премії.

Для даного завдання:

- ЗП 1р = 450,00 грн.;
- виконавець має 14 розряд;
- для 14 розряду тарифний коефіцієнт Кт = 3,36;
- загальний час створення програмного продукту Т=399,1 чол. / годин;
- кількість робочих днів Чр = 21;
- тривалість робочого дня tр.д. = 8 годин;
- відсоток премії П=15%.

Таким чином, визначаємо основну заробітну плату виконавця робіт із створення програмного продукту

ЗП осн. = (450,00 · 3,36· 399,1)/(21·8)·(1+15/100)=4130,68 [грн].

Додаткова заробітна плата береться у розмірі 15 % від основної і розраховується за формулою

ЗП дод. = ЗПосн. · 15 / 100, (3.9)

ЗП дод. = 4130,68 · 15 / 100=619,6 [грн].

Загальна заробітна плата розраховується за формулою

ЗПзагальна = ЗПосн + ЗПдод., (3.10)

ЗПзагальна = 4130,68 + 619,6 =4750,28 [грн].

3.1.3 Розрахунок нарахувань на заробітну плату (єдиного соціального внеску)

Єдиний соціальний внесок нараховується на заробітну плату за формулою 3.11 і складає 22 % від загальної заробітної плати

ЄСВ = ЗПзагальна · 22 / 100, (3.11)

ЄСВ = 4750,28 · 22 / 100 = 1045,06 [грн].

3.1.4 Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту

Витрати на збереження та експлуатацію ПЕОМ визначаються у розмірі 130% від основної заробітної плати працівників, що забезпечують функціонування ПЕОМ.

До таких витрат відносяться витрати на:

- основну заробітну плату працівників, що забезпечують функціонування ПЕОМ (інженер-електронник; системний програміст; оператор);
- основна ЗП адміністративного і допоміжного персоналу;
- амортизаційні відрахування;
- витрати на електричну енергію;
- витрати на матеріали (до їх числа входять диски, картриджі і папір для принтерів та інше);
- витрати на профілактику ПЕОМ з периферією;
- витрати на опалювання виробничих площ;
- витрати на обслуговування виробничих площ;
- інші виробничі витрати.

Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту за формулою

В зб. експл. = ЗП осн. · 130 / 100, (3.12)

В зб. експл. = 4130,68 · 130 / 100 = 5369,88 [грн].

3.2 Розрахунок собівартості програмного продукту

У собівартість програмного продукту входять наступні елементи:

- основна заробітна плата виконавця робіт із створення програмного продукту;

- додаткова заробітна плата виконавця робіт із створення програмного продукту;

- нарахування на заробітну плату (єдиний соціальний внесок);

- витрати на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту;

- інші витрати.

Інші витрати складають 10% від суми перших чотирьох елементів і розраховуються за формулою

І.вит.=(ЗП осн. + ЗП дод. + ЄСВ + В зб. експл.) ·10 / 100 (3.13)

І.вит.= (4130,68 + 619,6 + 1045,06 + 5369,88) ·10 / 100 = 1116,52 [грн].

Визначається собівартість програмного продукту за формулою

Сп.п. = ЗП осн. + ЗП дод. + ЄСВ + В зб.експл. + І.вит. (3.14)

Сп.п. = 4130,68 + 619,6 + 1045,06 + 5369,88+ 1116,52 = 12281,74[грн].

Структура собівартості програмного продукту відображається в таблиці 3.5.

Таблиця 3.5

Структура собівартості програмного продукту

Елементи структури	Сума грн.	Питома вага %
1	**2**	**3**
1 Основна заробітна плата виконавця	4130,68	33.7
2 Додаткова заробітна плата виконавця	619,6	5
3 Нарахування на заробітну плату (ЄСВ)	1045,06	8.5
4 Витрати збереження та експлуатацію ПП	5369,88	43.7
5 Інші витрати	1116,52	9.1
Собівартість програмного продукту	12281,74	100

3.3 Розрахунок ціни програмного продукту

Ціна програмного продукту складається з декількох компонентів і розраховуються за формулою

Ц = Сп.п. + П + ПДВ, (3.15)

де Сп.п.- собівартість програмного продукту, грн.;

П - плановий прибуток, грн.;

ПДВ - податок на додану вартість, грн.

П – прибуток складає 40% від повної собівартості програмного продукту і розраховуються за формулою

П = Сп.п. · 40 / 100 (3.16)

П = 12281,74·40 / 100 = 4912,69 [грн].

ПДВ - податок на додану вартість, який береться у розмірі 20% від суми собівартості і прибутку розраховуються за формулою

ПДВ = (Сп.п. + П) ·20 / 100, (3.17)

ПДВ = (12281,74+4912,69) · 20 /100 = 3438,88 [грн].

За формулою 3.15 визначається ціна програмного продукту.

Ц = 12281,74+ 4912,69 + 3438,88 = 20633,31 [грн].

3.4 Зведена таблиця показників

Результати розрахунків відображаються в підсумкової таблиці 3.6.

Таблиця 3.6

Результати розрахунків

Найменування показника	**Сума, грн.**
1	**2**
Собівартість програмного продукту	12281,74
Прибуток	4912,69
ПДВ	3438,88
Ціна програмного продукту	20633,3

Розроблена гра має всі базові функції наявних аналогів на світових ринках, а також доповнює деякі з них. При цьому розроблена гра має конкурентно

спроможну ціну. Гра заслуговує уваги від користувачів мережі. Результати розрахунку показані в діаграмі.

Собівартість програмного продукту відображається у вигляді діаграми (Рис. 3.1).

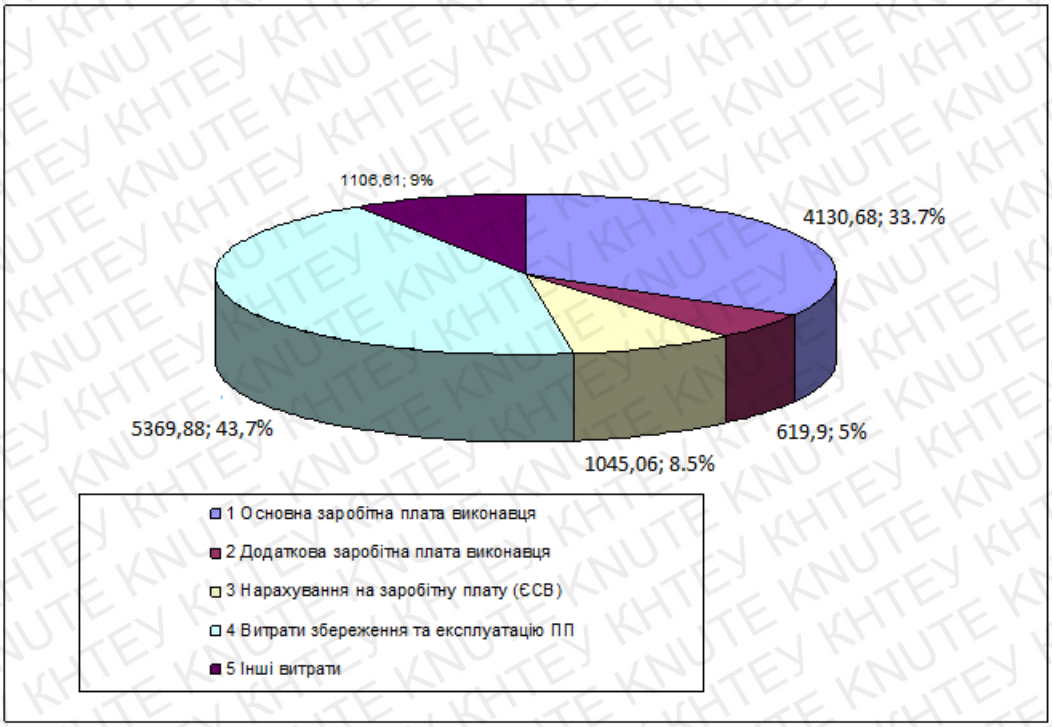

Рис. 3.1. Діаграма собівартості гри *«What is in common»*

Висновок до розділу 3

У третьому розділі були наведені економічні витрати, які будуть супроводжувати розробку данного продукту, під назвою гра *«What is in common»*. Наведена діаграма собі вартості, за формулами були розраховані усі можливі витрати.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході написання дипломної роботи була розроблена розширена версія гри «Що спільного?» для Windows мовою C#.Ознайомився з правилами гри «Що спільного?» та з різними її реалізаціями. Сформував технічне завдання та план його виконання. Розробив готовий проект, який реалізує логіку гри та має необхідний для гравця функціонал.В якості середовища програмування було обрано набір засобів розробки додатків, який дозволяє створювати додатки з графічним користувацьким інтерфейсом Mictrosoft Visual Studio 2017 і включає в себе всі інструменти для реалізації поставленої задачі.

У роботі були проведені дослідження предметної частини і наведені аналоги ігор на уважність. Також був проведений аналіз ігор, який представлений у зведеній таблиці.

В якості практичної частини була розроблена відповідно до попередньо складеного технічного завдання програма, яка реалізує гру «Що спільного?». В ході розробки були вивчені вимоги, що пред'являються до процесу розглянуто питання щодо документації програмного забезпечення.

В даний час, зв'язку з появою нових інформаційних технологій мова

програмування C# є популярною. Цілі, поставлені перед початком роботи вважаю досягнутими, а тему, світлену в рамках даної роботи – повністю розкритою.

					КНТЕУ 121 06-02.БР		
Зм.	Аркуш	№ докум	Підпис	Дата			
Зав. кафедри		Криворучко О.В.			Розробка програмного забезпечення «What is in common»	Стадія	Аркуш	Аркушів
Керівник		Пашорін В.І.				ВП	59	62
Гарант		Цензура М.О.				Факультет інформаційних технологій, 4 курс, 6 група		
Розроб.		Баркар О.І..			Висновки та пропозиції			

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Законодавчі та нормативні документи

1. Про акціонерні товариства : Закон України від 17.09.2008 № 514-VI.
2. Про Концепцію вдосконалення державного регулювання природних монополій : Указ Президента України від 27.09.2007 № 921/2007.
3. Податковий кодекс України : за станом на 02.12.2010 № 2755-VI // Верховна Рада України. – К. : Парлам. вид-во, 2010. – 207 с.
4. Положення про Міністерство юстиції України : затв. постановою Кабінету Міністрів України від 14.11.2006 № 1577.
5. Про затвердження Положення про Державну комісію з регулювання ринків фінансових послуг України: постанова Кабінету Міністрів України від 03.02.2010 № 157.
6. Інструкція зі статистики заробітної плати: наказ Держ. комітету статистики України від 13.01.2004 № 5.
7. Про деякі питання практики застосування конкурентного законодавства : Інформ. лист Вищого господарського суду України від 13.04.2007 № 01-8/229.
8. Закон України Про оплату праці.
9. Закон України Про підприємництво.
10. Закон України Про оподаткування прибутку підприємств.
11. ГОСТ 19.001-77 ЄСПД. Загальні положення.
12. ГОСТ 19.002-80 ЄСПД. Схеми алгоритмів і програм. Правила виконання.
13. ГОСТ 19.003-80 ЄСПД. Схеми алгоритмів і програм. Позначення умовні графічні.
14. ГОСТ 19.105-78 ЄСПД. Загальні вимоги до програмних документів.

15. ГОСТ 19.401-78 ЄСПД. Текст програми. Вимоги до змісту і оформлення.

16. ГОСТ 19.402-78 ЄСПД. Опис програми.

17. ГОСТ 19.504-79 ЄСПД. Інструкція програміста.

18. ГОСТ 19.505-79 ЄСПД. Інструкція оператора.

19. ГОСТ 19.506-79 ЄСПД. Опис мови. Вимоги до змісту і оформлення

20. ДСТУ – 2293 – 93. Система стандартів безпеки праці. Охорона праці.

Книги колективу авторів

21. Р. Шелдон, Д. Мойе, MySQL: базовый курс - Beginning MySQL-533с.

22. М. Кузнецов, И. Симдянов, MySQL на примерах. – Спб.: БХВ Петербург, 2008.-211 с.

Електронні ресурси

23. Сайт мови MySql - https://www.mysql.com/

ДОДАТКИ

Додаток А

Код програми

MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using System.ServiceModel.Channels;
using System.Threading;
using System.Threading.Tasks;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.Storage;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Media.Imaging;
using Windows.UI.Xaml.Navigation;
using Newtonsoft.Json;

  namespace WhatInCommonGame
```

```
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private List<GameData> GameDataList { get; set; }
        private int CurrentLevel { get; set; } = TempData.CurrentLevel - 1;
        private int Score { get; set; } = TempData.CurrentScore;

        /// <summary>
        ///
        /// </summary>
        private DispatcherTimer Timer { get; set; } = new DispatcherTimer
        {
            Interval = new TimeSpan(1000L)
        };

        /// <summary>
        ///
        /// </summary>
        private DateTime StartTime { get; set; } = DateTime.Now;

        public MainPage()
        {
            InitializeComponent();


            GameDataList = new List<KeyValuePair<string, string>>
```

```
                {
                    new KeyValuePair<string, string>("sniper",
@"Снайпер"),
                    new KeyValuePair<string, string>("roll",
@"Пленка"),
                    new KeyValuePair<string, string>("sweat", @"Пот"),
                    new KeyValuePair<string, string>("atm",
@"Банкомат"),
                    new KeyValuePair<string, string>("bottom",
@"Дно"),
                    new KeyValuePair<string, string>("car",
@"Машина"),
                    new KeyValuePair<string, string>("game", @"Игра"),
                    new KeyValuePair<string, string>("lion", @"Лев"),
                    new KeyValuePair<string, string>("plastic",
@"Пластик"),
                    new KeyValuePair<string, string>("rag",
@"Тряпка"),
                    new KeyValuePair<string, string>("scotch",
@"Скотч"),
                    new KeyValuePair<string, string>("track", @"След"),
                    new KeyValuePair<string, string>("weapon",
@"Оружие")
                }.Select(key => new GameData
                {
                    image1 = new BitmapImage(new Uri($"ms-
appx:///Assets/{key.Key}1.jpg")),
                    image2 = new BitmapImage(new Uri($"ms-
appx:///Assets/{key.Key}2.jpg")),
```

```
                    image3 = new BitmapImage(new Uri($"ms-
appx:///Assets/{key.Key}3.jpg")),
                    image4 = new BitmapImage(new Uri($"ms-
appx:///Assets/{key.Key}4.jpg")),
                    anwser = key.Value
                }).ToList();

                fillLevel(CurrentLevel);

                scoreText.Text = $"Score: {Score}";

                recordsButton.Click += async (sender, args) =>
                {
                    if (recordGrid.Visibility == Visibility.Collapsed)
                    {
                        recordsButton.Content = "Back";

                        scoreText.Visibility = Visibility.Collapsed;

                        grid.Visibility = Visibility.Collapsed;

                        recordGrid.Visibility = Visibility.Visible;

                        recordText.Text = await
FileIO.ReadTextAsync(
                            await
ApplicationData.Current.LocalFolder.GetFileAsync("Highscores.txt"));

                    }
                    else
```

```
            {
                recordsButton.Content = "Highscores";

                scoreText.Visibility = Visibility.Visible;

                grid.Visibility = Visibility.Visible;

                recordGrid.Visibility = Visibility.Collapsed;
            }
        };

        exitButton.Click += (sender, args) =>
        {
            Timer.Stop();
            TempData.CurrentLevel = CurrentLevel + 1;
            TempData.CurrentScore = Score;
            Frame.Navigate(typeof(StartPage));
        };

        Timer.Tick += async (sender, o) =>
        {
            var remainTime = 20 - (DateTime.Now -
StartTime).Seconds;
            timeBlock.Text = $"Time: {remainTime}";
            if (remainTime == 0)
            {
                Timer.Stop();
                var msg = new MessageDialog("Sorry", "You
lose.");
                CurrentLevel = 1;
```

```
                    Score = 0;
                    scoreText.Text = $"Score: {Score}";
                    await msg.ShowAsync();
                    TempData.CurrentLevel = CurrentLevel + 1;
                    TempData.CurrentScore = Score;
                    await SaveGame();
                    fillLevel(1);
                }
            };
        }

        private async Task SaveGame()
        {
            var storageFolder = ApplicationData.Current.LocalFolder;
            var storageFile = await
storageFolder.CreateFileAsync(@"save.txt",
                CreationCollisionOption.OpenIfExists);

            var savedJsonString = await
FileIO.ReadTextAsync(storageFile);

            if (!string.IsNullOrEmpty(savedJsonString))
            {
                var savedData =
JsonConvert.DeserializeObject<SaveData>(savedJsonString);
                if (savedData != null)
                {
                    await FileIO.WriteTextAsync(storageFile,
JsonConvert.SerializeObject(new SaveData
                    {
```

```
                                    LastLevel = 1,
                                    LastScore = 1,
                                    MaxLevel = savedData.MaxLevel <
TempData.CurrentLevel ? TempData.CurrentLevel : savedData.MaxLevel,
                                    MaxScore = savedData.MaxScore <
TempData.CurrentScore ? TempData.CurrentScore : savedData.MaxScore
                                }));
                        }
                    }
                    else
                    {
                        await FileIO.WriteTextAsync(storageFile,
JsonConvert.SerializeObject(new SaveData
                        {
                            LastLevel = TempData.CurrentLevel,
                            LastScore = TempData.CurrentScore,
                            MaxLevel = TempData.CurrentLevel
                        }));
                    }
                }

                protected override async void
OnNavigatingFrom(NavigatingCancelEventArgs e)
                {
                    var storageFolder = ApplicationData.Current.LocalFolder;
                    var storageFile = await
storageFolder.CreateFileAsync(@"save.txt",
                        CreationCollisionOption.OpenIfExists);
```

```
                var savedJsonString = await
FileIO.ReadTextAsync(storageFile);

                if (!string.IsNullOrEmpty(savedJsonString))
                {
                    var savedData =
JsonConvert.DeserializeObject<SaveData>(savedJsonString);
                    if (savedData != null)
                    {
                        await FileIO.WriteTextAsync(storageFile,
JsonConvert.SerializeObject(new SaveData
                        {
                            LastLevel = TempData.CurrentLevel,
                            LastScore = TempData.CurrentScore,
                            MaxLevel = savedData.MaxLevel <
TempData.CurrentLevel ? TempData.CurrentLevel: savedData.MaxLevel,
                            MaxScore = savedData.MaxScore <
TempData.CurrentScore ? TempData.CurrentScore : savedData.MaxScore
                        }));
                    }
                }
                else
                {
                    await FileIO.WriteTextAsync(storageFile,
JsonConvert.SerializeObject(new SaveData
                    {
                        LastLevel = TempData.CurrentLevel,
                        LastScore = TempData.CurrentScore,
                        MaxLevel = TempData.CurrentLevel
                    }));
```

```
            }

            base.OnNavigatingFrom(e);
        }

        private void fillLevel(int level)
        {
            image1.Source = GameDataList[level].image1;
            image2.Source = GameDataList[level].image2;
            image3.Source = GameDataList[level].image3;
            image4.Source = GameDataList[level].image4;

            StartTime = DateTime.Now;
            Timer.Start();
        }

        private async void checkButton_Click(object sender,
RoutedEventArgs e)
        {
            if (CurrentLevel != GameDataList.Count - 1)
            {
                if (textBox.Text.Trim().ToUpper() ==
GameDataList[CurrentLevel].anwser.Trim().ToUpper())
                {
                    textBox.Text = "";
                    Score += 100;
                    scoreText.Text = $"Score: {Score}";
                    fillLevel(++CurrentLevel);
                }
                else
```

```
                {

                    await FileIO.AppendTextAsync(
                        await
ApplicationData.Current.LocalFolder.GetFileAsync("Highscores.txt"),
                        $"{TempData.UserName} : {Score}");

                    textBox.Text = "";

                    Score = 0;
                    CurrentLevel = 0;
                    scoreText.Text = $"Score: {Score}";
                    fillLevel(CurrentLevel);

                    await new MessageDialog("You
lose.").ShowAsync();
                }
            }
            else
            {
                await FileIO.AppendTextAsync(
                    await
ApplicationData.Current.LocalFolder.GetFileAsync("Highscores.txt"),
                    $"{TempData.UserName} : {Score}");

                Timer.Stop();

                textBox.Text = "";

                Score = 0;
```

```
                CurrentLevel = 0;
                scoreText.Text = $"Score: {Score}";
                fillLevel(CurrentLevel);

                await new MessageDialog("You win!").ShowAsync();
            }

        }
    }
}
```

StartPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.Storage;
using Windows.UI;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Newtonsoft.Json;
```

```
        // The Blank Page item template is documented at
https://go.microsoft.com/fwlink/?LinkId=234238

        namespace WhatInCommonGame
        {
            /// <summary>
            /// An empty page that can be used on its own or navigated to within a
Frame.
            /// </summary>
            public sealed partial class StartPage : Page
            {
                public StartPage()
                {
                    this.InitializeComponent();
                    nameBox.Text = TempData.UserName;

                    startButton.Click += (obj, a) =>
                    {
                        TempData.UserName = nameBox.Text;
                        grid.Children.Clear();

                        for (int i = 0; i < 3; i++)
                        {
                            for (int j = 0; j < 4; j++)
                            {
                                var button = new Button
                                {
                                    Content = j * 3 + i + 1,
                                    Height = grid.Height / 4,
                                    Width = grid.Width / 3,
```

```
                                        Margin = new
Thickness(grid.Width / 3 * i, grid.Height / 4 * j, 0, 0),
                                        HorizontalAlignment =
HorizontalAlignment.Left,
                                        VerticalAlignment =
VerticalAlignment.Top,
                                        BorderBrush = new
SolidColorBrush(Colors.DarkBlue),
                                        IsEnabled = i == 0 && j == 0 ||
TempData.CurrentLevel >= j * 3 + i + 1,
                                        Visibility = Visibility.Visible
                                    };
                                    button.Click += (sender, args) =>
                                    {
                                        var bt = sender as Button;
                                        TempData.CurrentLevel =
Convert.ToInt32(bt?.Content);
                                        TempData.CurrentScore =
Convert.ToInt32(bt?.Content) * 100;

    Frame.Navigate(typeof(MainPage));
                                    };
                                    grid.Children.Add(button);
                                }
                            }
                        };

                        loadButton.Click += async (sender, args) =>
                        {
                            var storageFile =
```

```
                    await
ApplicationData.Current.LocalFolder.CreateFileAsync(@"save.txt",
CreationCollisionOption.OpenIfExists);

                    var savedJsonString = await
FileIO.ReadTextAsync(storageFile);

                    if (!string.IsNullOrEmpty(savedJsonString))
                    {
                        var savedData =
JsonConvert.DeserializeObject<SaveData>(savedJsonString);
                        if (savedData != null)
                        {
                            TempData.CurrentLevel =
savedData.LastLevel;
                            TempData.CurrentScore =
savedData.LastScore;
                            Frame.Navigate(typeof(MainPage));
                        }
                    }
                };
            }

            /// <summary>
            ///
            /// </summary>
            /// <param name="e"></param>
            protected override async void
OnNavigatedTo(NavigationEventArgs e)
            {
```

```
                await
ApplicationData.Current.LocalFolder.CreateFileAsync(@"Highscores.txt",
CreationCollisionOption.OpenIfExists);
                var storageFile =
                    await
ApplicationData.Current.LocalFolder.CreateFileAsync(@"save.txt",
CreationCollisionOption.OpenIfExists);

                var savedJsonString = await
FileIO.ReadTextAsync(storageFile);

                loadButton.Visibility =
!string.IsNullOrEmpty(savedJsonString) ? Visibility.Visible : Visibility.Collapsed;

                if (!string.IsNullOrEmpty(savedJsonString))
                {
                    var savedData =
JsonConvert.DeserializeObject<SaveData>(savedJsonString);
                    if (savedData != null)
                    {
                        TempData.CurrentLevel =
savedData.MaxLevel;
                        TempData.CurrentScore =
savedData.MaxScore;
                    }
                }

                base.OnNavigatedTo(e);
            }
        }
```

```
}
```

TempData.cs

```
using System;

namespace WhatInCommonGame
{
    public static class TempData
    {
        public static int CurrentLevel { get; set; } = 0;

        public static int CurrentScore { get; set; } = 0;

        public static string UserName { get; set; } = $"Player№{new
Random(DateTime.Now.Millisecond).Next(0, 100)}";
    }
}
```

SaveData.cs

```
namespace WhatInCommonGame
{
    public class SaveData
    {
        /// <summary>
        ///
        /// </summary>
        public int LastLevel { get; set; } = 1;

        /// <summary>
        ///
        /// </summary>
        public int LastScore { get; set; } = 0;
```

```
        /// <summary>
        ///
        /// </summary>
        public int MaxLevel { get; set; } = 1;

        /// <summary>
        ///
        /// </summary>
        public int MaxScore { get; set; } = 0;
    }
}
```