**Форма № 17**

Міністерство освіти і науки України

**Національний університет «Запорізька політехніка»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра програмних засобів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(найменування кафедри)

**КУРСОВИЙ ПРОЄКТ**

**(РОБОТА)**

з дисципліни «Об’єктно-орієнтоване програмування»

(назва дисципліни)

на тему: «Гра віртуальна домашня тваринка «Ducky»» в

Студентів(ки) 2 курсу КНТ-129 групи

спеціальності 121 Інженерія   
програмного забезпечення

освітня програма (спеціалізація) інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_Харитонов Д. О.\_\_\_\_\_\_\_

(прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_Остапенко М. В.\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_Морозов К. \_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_Коляда М.\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент, к.т.н., Каплієнко Т. І.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   Каплієнко Т. І.

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали

2022 рік

**Форма № 25**

Міністерство освіти і науки України

**Національний університет «Запорізька політехніка»**

( повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра програмних засобів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ступінь вищої освіти      бакалавр**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Спеціальність 121 Інженерія програмного забезпечення**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення**\_\_\_\_\_\_\_**

(назва освітньої програми (спеціалізації)

**ЗАТВЕРДЖУЮ**

**Завідувач кафедри ПЗ, д.т.н, проф.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С.О. Субботін**

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_року

З А В Д А Н Н Я

**НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

  Прізвище Д. Ю., Прізвище В. Д., Прізвище А. К., Прізвище К. В.

(прізвище, ім’я, по батькові)

1. Тема проєкту (роботи) «Гра віртуальна домашня тваринка «Ducky»»

керівник проєкту (роботи)\_Каплієнко Тетяна Ігорівна, к.т.н., доцент ,

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від

2. Строк подання студентом проєкту (роботи)\_05 грудня 2022 року\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до проєкту (роботи) створити гру віртуальна домашня тваринка «Ducky» а

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області; 2. Аналіз програмних засобів; 3. Основні рішення з реалізації компонентів системи; 4. Керівництво програміста; 5. Керівництво користувача; 6. Додатки.     
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)\_\_

Слайди презентації\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультанти розділів проєкту (роботи)

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада  консультанта | Підпис, дата | |
| завдання видав | прийняв  виконане завдання |
| 1-5 Основна частина | Каплієнко Т.І., доцент |  |  |
|  |  |  |  |
|  |  |  |  |

7. Дата видачі завдання\_\_13 вересня 2021 р.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів курсового  проєкту (роботи) | Строк виконання етапів проєкту  ( роботи ) | Примітка |
| 1. | Аналіз індивідуального завдання. | 1 тиждень |  |
| 2. | Аналіз програмних засобів, що будуть використовуватись в роботі. | 2 тиждень |  |
| 3. | Аналіз структур даних, що необхідно використати в курсової роботі. | 3 тиждень |  |
| 4. | Затвердження завдання | 4 тиждень |  |
| 5. | Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача. | 5-9 тиждень |  |
| 6. | Аналіз вимог до апаратних засобів | 9 тиждень |  |
| 7. | Розробка програмного забезпечення | 9-13 тиждень |  |
| 8. | Проміжний контроль | 10 тиждень | Розділи 1-2 ПЗ |
| 9. | Оформлення, відповідних пунктів пояснювальної записки. | 10-14 тиждень | Розділи 1-5 ПЗ |
| 10. | Захист курсової роботи. | 15 тиждень |  |
|  |  |  |  |
|  |  |  |  |

**Студент (ка) \_\_\_\_\_\_\_\_\_ \_\_\_** Прізвище Д. Ю.**\_\_\_\_**

( підпис ) (прізвище та ініціали)

**Студент (ка) \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_** Прізвище В. Д.**\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

**Студент (ка) \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_** Прізвище А. К.**\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

**Студент (ка) \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_** Прізвище К. В.**\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

**Керівник проєкту (роботи) \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_** Каплієнко Т.І.**\_\_\_\_**

( підпис ) (прізвище та ініціали)

**реферат**

Мета роботи – створення гри симулятора домашнього улюбленця на основі старої ретро-гри Tamagochi.

Проведено аналіз предметної області, досліджено аналогічне програмне забезпечення, а також існуючі методи та програмні засоби для вирішення завдання.

Для реалізації гри-симулятора використовувалася мова програмування С# та середа розробки Visual Studio.

Здійснено опис прийнятих рішень, реалізованих класів, наведено опис полів та методів реалізованих класів. Також у роботі були використані контейнери List для збереження даних.

У даній роботі було проведено дослідження розробки ігор-симуляторів, розглянуто особливості мови програмування С# у середовищі розробки Visual Studio 2021. За основу для розробки графічного інтерфейсу для користувача була узята мова розмітки сторінки XAML, що використовувалася на базі графічної системи WPF. Потреба у збереженні даних реалізована у роботі зі спеціалізованими форматами збереження даних JSON та XML.

VISUAL STUDIO, WPF, JSON, XML, C#, XAML, DUCK, TAMAGOCHI.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7](#_Toc122535200)

[ВСТУП 8](#_Toc122535201)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9](#_Toc122535202)

[1.1 Аналіз відпочинку, як основи предметної області 9](#_Toc122535203)

[1.2 Огляд існуючих методів вирішення завдання 9](#_Toc122535204)

[1.3 Огляд існуючих програм та сервісів для проведення вільного часу 10](#_Toc122535205)

[1.3.1 Передмова 10](#_Toc122535206)

[1.3.2 Гра «tamagochi-xomyachok» 10](#_Toc122535207)

[1.3.3 Гра «My Pocket Pets» 12](#_Toc122535208)

[2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ 15](#_Toc122535209)

[2.1 Огляд особливостей мови програмування 15](#_Toc122535210)

[2.2 Огляд особливостей обраного компілятору 16](#_Toc122535211)

[2.3 Огляд класової ієрархії 16](#_Toc122535212)

[2.4 Висновки з розділу 17](#_Toc122535213)

[3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ 18](#_Toc122535214)

[3.1 Основні рішення щодо розроблених класів 18](#_Toc122535215)

[3.1.1 Клаc Tamagochi 20](#_Toc122535216)

[3.1.2 Клас InvertoryController 21](#_Toc122535217)

[3.1.3 Interface IFood 22](#_Toc122535218)

[3.1.4 Клас Sugar 23](#_Toc122535219)

[3.1.5 Клас TamagochiJson 24](#_Toc122535220)

[3.1.6 Клас TamagochiXML 24](#_Toc122535221)

[3.1.7 Клас ExceptionController 25](#_Toc122535222)

[3.1.8 Класс MainWindow 26](#_Toc122535223)

[3.1.9 Клас Item 29](#_Toc122535224)

[3.2 Основні розроблені алгоритми 30](#_Toc122535225)

[3.3 Основні рішення щодо розробки інтерфейсу 30](#_Toc122535226)

[3.4 Основні рішення щодо роботи з даними 31](#_Toc122535227)

[3.5 Обробка виключних ситуацій 31](#_Toc122535228)

[4 КЕРІВНИЦТВО ПРОГРАМІСТА 32](#_Toc122535229)

[4.1 Призначення та умови застосування програми 32](#_Toc122535230)

[4.1.1 Призначення програми 32](#_Toc122535231)

[4.1.2 Функції програми 32](#_Toc122535232)

[4.1.3 Умови застосування програми 32](#_Toc122535233)

[4.2 Характеристика програми 33](#_Toc122535234)

[4.2.1 Структура програми 33](#_Toc122535235)

[4.3 Звертання до програми 33](#_Toc122535236)

[4.4 Вхідні та вихідні дані 34](#_Toc122535237)

[4.4.1 Вхідні дані 34](#_Toc122535238)

[4.4.2 Вихідні дані 34](#_Toc122535239)

[5 КЕРІВНИЦТВО КОРИСТУВАЧА 35](#_Toc122535240)

[5.1 Призначення програми 35](#_Toc122535241)

[5.2 Умови виконання програми 35](#_Toc122535242)

[5.2.1 Апаратні вимоги програми 35](#_Toc122535243)

[5.2.2 Вимоги до користувача 35](#_Toc122535244)

[5.3 Виконання програми 36](#_Toc122535245)

[5.3.1 Запуск програми 36](#_Toc122535246)

[5.3.2 Виконання роботи з програмою 36](#_Toc122535247)

[5.4 Повідомлення користувачу 37](#_Toc122535248)

[5.5 Довідка програми 38](#_Toc122535249)

[ВИСНОВКИ 39](#_Toc122535250)

[ПЕРЕЛІК ПОСИЛАНЬ 40](#_Toc122535251)

# **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ**

ООП - об’єктно-орієнтоване програмування.

JSON - JavaScript Object Notation.

XML - Extensible Markup Language.

XAML - EXtensible Application Markup Language.

List – структура з простору імен System.Collections.Generic, що представляє найпростіший список однотипних об'єктів.

WPF - Windows Presentation Foundation, система для побудови клієнтських програм Windows з візуально привабливими можливостями взаємодії з користувачем.

# **ВСТУП**

Розробка ігор є одним з найпопулярніших напрямків розробки програмового забезпечення, гри з 1980-1990рр. все більше і більше набирають популярність у новій розробці.

Отже, актуальність даного програмного забезпечення полягає у створенні гри-симулятору Tamagochi на новий лад з елементами готування їжі або алхімії.

Область використання такого застосунку включає в себе найважливішу сферу людської діяльності, а саме відпочинок та розваги.

Наразі сучасна сфера розробки ігор не має аналогу із казуальних симуляторів життя на десктоп-приладах, проблема у сфері розваг для данних застосунків залишається досить гострою, тому дана гра є актуальною для розваг та відпочинку.

Отже, мета даного дослідження, а саме розробка гри «Ducky», є актуальною задачею.

# **1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

В даному розділі розглянуті існуючі методи провести вільний час та відпочити, а також розглянуті програми, що реалізують дану потребу, а саме гра «Ducky».

## **1.1 Аналіз відпочинку, як основи предметної області**

Проблема проведення вільного часу активно вирішується багатьма розробниками ігор на різних мовах програмування, і це робиться у сфері розваг та відпочинку, тому предметна область саме така – організація розваг та відпочинку.

Основні вимоги у цій сфері наступні:

* вирішення головної мети – розважання користувача;
* певна ціль, яку протягом гри буде досягнуто;
* зрозумілий інтерфейс;
* привабливий дизайн.

## **1.2 Огляд існуючих методів вирішення завдання**

Проведення вільного часу та відпочинок можуть бути задовільнені багатьма різними способами, і один із них це проведення часу у іграх. Саме такою і є гра «Ducky». У більшості ігор є певна мета, яку потрібно досягнути. Наприклад проходження рівня, набрання певної кількості баллів, отримання найвишого балу серед інших гравців тощо. Під час досягнення цієї мети гравець витрачає вільний час, та отримує задоволення від того, що він досягає певної мети. Також, є ігри, в яких мета ніколи не буде досягнута, однак задоволення гравець отримує завдяки тому, що кожен раз він засвоює нову інформацію, з кожним разом може пройти ще далі, отримує більш кращу оцінку за проходження, має почуття задоволення від свого розвитку у грі. Гра «Ducky» саме із таких ігор. В ній реалізовано список кращих гравців, і є куди розвиватися, отримуючи більшу кількість балів. В свою чергу, повністю пройти гру немає можливості, і задоволення здобувається завдяки отриманню балів

Фактично, в даний час, не існує універсальних методів вирішення данної проблеми, так як користувач може отримувати задоволення від різних видів ігор та не всі будуть йому до вподоби

## **1.3 Огляд існуючих програм та сервісів для проведення вільного часу**

### **1.3.1 Передмова**

Слід зазначити, що програм, які вирішують проблему з вільним часом та розвагами дуже багато. Було розглянуто деякі із них, які подібні до гри «Ducky».

Перелік деяких з них:

* tamagochi-xomyachok;
* My Pocket Pets.

### **1.3.2 Гра «tamagochi-xomyachok»**

Посилання: <https://igrulez.net/game/igra-tamagochi-xomyachok>

Короткий опис: гра полягає у вирощуванні монстру. Є кількість дій, які можна зробити, та перелік дій (рис. 1.3.2.1). В залежності від того, що було за цю кількість дій обрано, та скільки разів, буде отриманий різний результат (рис. 1.3.2.2)

Переваги:

* гра має логічне завершення;
* відсутність таймеру;
* отримання різних результатів;
* анімація кожної дії.

Недоліки:

* коротка гра, швидко закінчується і потрібно починати зпочатку;
* для отримання різних результатів треба проходити гру повністю.

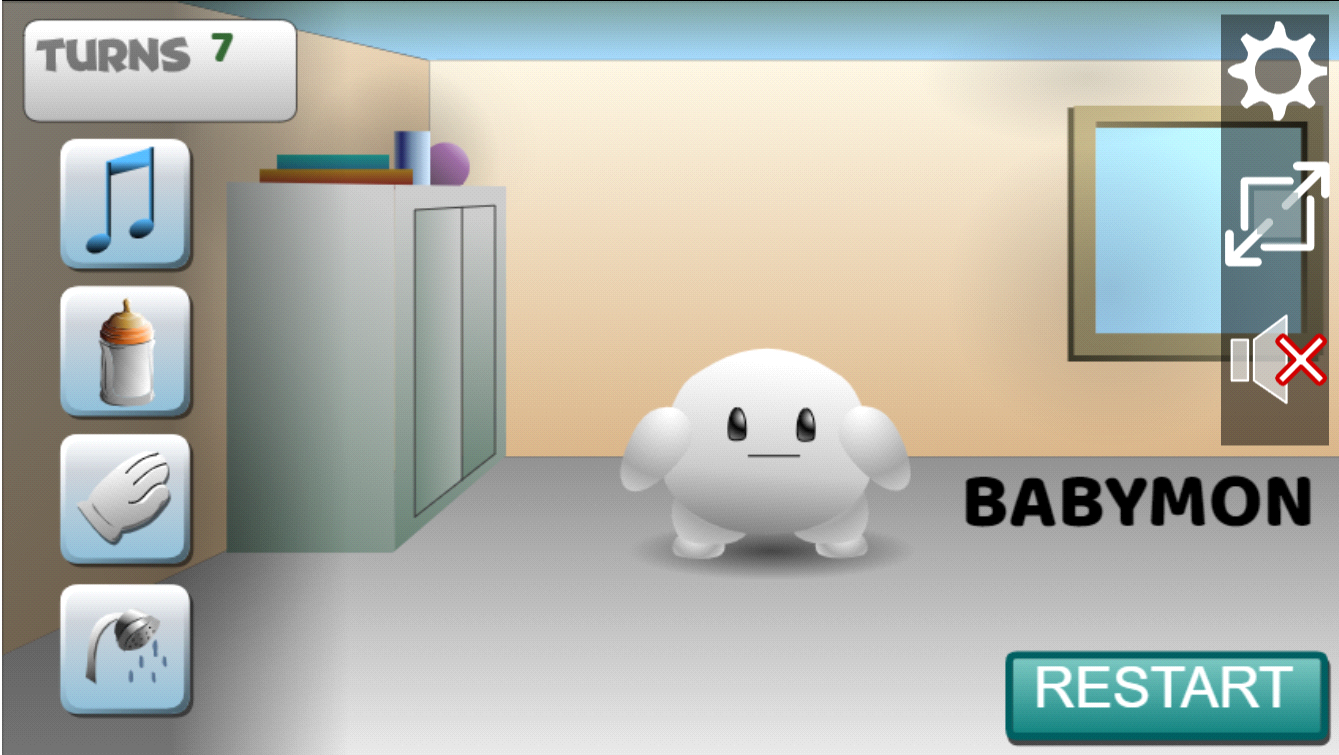


Рисунок 1.3.2.1 – Інтерфейс гри tamagochi-xomyachok



Рисунок 1.3.2.2 – Кінець гри tamagochi-xomyachok

### **1.3.3 Гра «My Pocket Pets»**

Посилання: <https://www.game-game.com.ua/191248/>

Короткий опис: гра полягає у вирощуванні кішки. З початку є яйце (рис. 1.3.3.1), і потрібно зробити певні дії, щоб отримати улюбленця (рис. 1.3.3.2). Потім потрібно робити різні дії, які підіймають різні характеристики улюбленця, і після заповнення всіх характеристик, можна перейти далі (рис. 1.3.3.3)

Переваги:

* розбиття на різні рівні з різними діями;
* гарний дизайн;
* можна вирощувати різних улюбленців;
* є прогрес розвитку персонажа (різні дії, різні показники).

Недоліки:

* гра не динамічна;
* дії відбуваються дуже повільно;
* якщо запустити гру ще раз, треба пройти всі рівні з початку.



Рисунок 1.3.3.1 – Початок гри



Рисунок 1.3.3.2 – Отриманий улюбленець



Рисунок 1.3.3.3 – Перехід до наступного рівня

# **2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ**

В даному розділі розглянуті основні особливості програмних засобів, за допомогою який реалізовано курсовий проєкт.

## **2.1 Огляд особливостей мови програмування**

C# — об'єктно-орієнтована, орієнтована на компоненти мова програмування. C# надає мовні конструкції безпосередньої підтримки такої концепції роботи. Завдяки цьому C# підходить для створення та застосування програмних компонентів. З моменту створення мова C# збагатилася функціями для підтримки нових робочих навантажень та сучасними рекомендаціями щодо розробки ПЗ. В основному C# - об'єктно-орієнтована мова. Ви визначаєте типи та їх поведінку.

Ось кілька функцій мови C#, які дозволяють створювати надійні та стійкі програми. Складання сміття автоматично звільняє пам'ять, зайняту недосяжними об'єктами, що не використовуються. Типи, що допускають значення null, забезпечують захист від змінних, які посилаються виділені об'єкти. Обробка винятків надає структурований та розширюваний підхід до виявлення помилок та відновлення після них. Лямбда-вираження підтримують прийоми функціонального програмування. Синтаксис LINQ створює загальний шаблон для роботи з даними будь-якого джерела. Підтримка мов для асинхронних операцій забезпечує синтаксис для створення розподілених систем. C# є Єдина система типів. Всі типи C#, включаючи типи-примітиви, такі як int та double, успадковують від одного кореневого типу об'єкта. Всі типи використовують загальний набір операцій, а значення будь-якого типу можна зберігати, передавати та обробляти таким чином. Більше того, C# підтримує як визначені користувачами типи посилань, так і типи значень. C# дозволяє динамічно виділяти об'єкти та зберігати спрощені структури у стеку. C# підтримує універсальні методи та типи, що забезпечують підвищену безпеку типів та продуктивність. C# надає ітератори, які дозволяють розробникам класів колекцій визначати варіанти поведінки для клієнтського коду.

## **2.2 Огляд особливостей обраного компілятору**

Для проєкту було обрано середу розробки Visual Studio. При створенні проекту середовище Visual Studio створює для нього конфігурації збірки за замовчуванням, а також рішення, що містить проект. Ці конфігурації визначають, як виконується складання та розгортання рішень та проектів. Зокрема, використовуються унікальні конфігурації проектів для різних цільових платформ (наприклад, Windows або Linux) та типів збирання (наприклад, налагодження або випуск). Ви можете як завгодно змінювати ці конфігурації та за необхідності створювати свої власні.

## **2.3 Огляд класової ієрархії**

У проєкті всі класи знаходяться у просторі імен Tamagochi\_WPF. Окрім цього, є 4 класи: TamagochiXml, TamagochiJson, Tamagochi, Item які не успадковують інші класи. Є інтерфейс IFood, від якого успадковані всі класи їжі (рис. 2.3.1)



Рисунок 2.3.1 – Діаграма класів для файлу Food

## **2.4 Висновки з розділу**

У даному розділі було описано мову програмування та її особливості. Також було описано обраний для виконання поставленої задачі компілятор та його особливості і зроблено огляд класової ієрархії

# **3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ**

В даному розділі розглянуті існуючі методи вирішення завдання розробки гри симуляторів, що реалізують дану потребу, а саме створення гри симулятору «Ducky».

## **3.1 Основні рішення щодо розроблених класів**

Структура програми складається з 37 класів, 36 з яких – користувацькі, 1 – клас форм (рис. 3.1.1). Один з користувацьких класів є інтерфейсом. Було використано інтерфейс замість абстрактного класу, так як у наслідуваних класах не потрібно було реалізовувати один або більше методів, а писати свої характеристики та методи. Діаграми користувацьких класів знаходяться у додатку А.

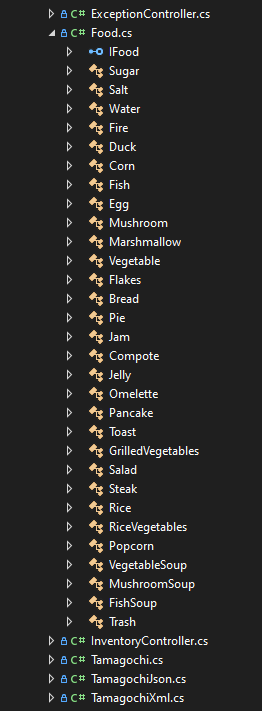


Рисунок 3.1.1 –Класи програми

### **3.1.1 Клас Tamagochi**

Цей клас містить у собі усі поля про всі ознаки, характеристики та методи персонажа гри. Клас використовується для передачі інформації до інтерфейсу та оновлення характеристик життя персонажа. У ньому перевіряється, які характеристики зменшити, або збільшити. Також перевіряється чи «живий» об’єкт. Містить поля для ініціалізації та перезапуску гри.

Дані та методи класу наведені у таблиці 3.1.

Таблиця 3.1 – Опис полів та методів класу Tamagochi

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public string Name | Містить геттери та сеттери для приватного поля name\_ |
| public int Happines | Містить геттери та сеттери для приватного поля happines\_ |
| public int Poisoning | Містить геттери та сеттери для приватного поля poisoning\_ |
| public int Saturation | Містить геттери та сеттери для приватного поля saturation\_ |
| public int Heal | Містить геттери та сеттери для приватного поля heal\_ |
| public bool IsAlive | Містить геттери та сеттери для приватного поля isAlive\_ |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| public Tamagochi(List<IFood> food\_) | Конструктор, переданий List передається для ініціалізації приватного поля об’єкту InvertoryController |
| public void StateCreate() | Задає стартові дані |
| public void StateDestroy() | Обнуляє всі доступні дані |
| public void StateUpdate() | Оновлює характеристики персонажа |
| public void Eat(IFood food\_) | Приймає об’єкт Food та передає його характеристики до об’єкту Tamagochi |

### **3.1.2 Клас InvertoryController**

Цей клас містить інформацію про об’єкти, які випадковим чином з’являються у списку продуктів, якими можна нагодувати «Ducky» або приготувати нову їжу з отриманих. У класі відбуваються всі маніпуляції з приготуванням, збереженням та видаленням їжі, яку буде приймати клас Tamagochi.

Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.2 – Опис полів та методів класу InvertoryController

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public List<Item> \_items | Поле, в якому зберігаються всі об’єкти Item, в яких є значення їжі |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| public InventoryController(List<IFood> foods\_) | Конструктор, який додає об’єкти у List |
| public IFood Craft(string ingredient\_right\_, string ingredient\_left\_) | Метод, в якому відбувається приготування їжі(змішування двох енгрідієнтів) |
| public void Add(IFood food\_, int amount\_ = 1) | Додання нових об’єктів Item у List |
| public void Remove(IFood food\_) | Видаляє елементи об’єктів Item у List |
| public bool CheckItem(IFood food\_) | Перевірка об’єкту Item в наявності у List |

### **3.1.3 Interface IFood**

Цей інтерфейс включає в себе поля з для обробки інформації про їжу, яку будуть наслідувати інші 30 класів їжі, та зберігати інформацію скільки здоров’я, насичення, щастя та отруєння, та рецепт цього елементу їжі з двох інших інгредієнтів.

Таблиця 3.3 – Опис полів та методів інтерфейсу IFood

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| string Name { get; } | Поле передає ім’я продукту |
| int Heal { get; } | Поле передає показник здоров'я |
| int Poison { get; } | Поле передає показник отруєння |

Продовження таблиці 3.3

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| int Happy { get; } | Поле передає показник щастя |
| int Satiety { get; } | Поле передає показник насичення |
| string[] Recipe { get; } | Поле передає рецепт їжі для приготування |
| bool HasRecipe(); | Метод передає наявність рецепту |

### **3.1.4 Клас Sugar**

Клас унаслідується від інтерфейсу IFood та приймає усі його поля. Дані з цього класу будуть передані до класу Tamagochi та змінять його показники. Таким чином описані класи: Salt, Water, Fire, Duck, Corn, Fish, Egg, Mushroom, Marshmallow, Vegetable, Flakes, Bread, Pie, Jam, Compote, Jelly, Omelette, Pancake, Toast, GrilledVegetables, Salad, Steak, Rice, RiceVegetables, Popcorn, VegetableSoup, MushroomSoup, FishSoup, Trash.

Таблиця 3.4 – Опис полів та методів інтерфейсу Sugar

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| string Name { get; } | Поле передає ім’я продукту |
| int Heal { get; } | Поле передає показник здоров'я |
| int Poison { get; } | Поле передає показник отруєння |
| int Happy { get; } | Поле передає показник щастя |
| int Satiety { get; } | Поле передає показник насичення |
| string[] Recipe { get; } | Поле передає рецепт їжі для приготування |

Продовження таблиці 3.4

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| bool HasRecipe(); | Метод передає наявність рецепту |

### **3.1.5 Клас TamagochiJson**

Цей клас містить дані, які приймає в себе json-файл для занесення інформації у рейтинг спроб гравця. Конструктор формулює дані, які потім конвертує у окремий файл.

Таблиця 3.5 – Опис полів та методів інтерфейсу TamagochiJson

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public string Name { get; set; } | Поле передає ім’я персонажа |
| public int Rating { get; set; } | Поле передає час життя персонажа |
| public TamagochiJson(string name, int timeOflife) | Конструктор що приймає час життя та ім’я персонажа |
| public TamagochiJson() | Первинний конструктор |

### **3.1.6 Клас TamagochiXML**

Цей клас містить дані, які приймає в себе xml-файл для збереження інформації при виході гравця з гри. Конструктор формулює дані, які потім конвертує у окремий файл.

Таблиця 3.6 – Опис полів та методів інтерфейсу TamagochiXML

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public TamagochiXml() | Первинний конструктор |
| public TamagochiXml(Tamagochi t, int timeOfLife) | Конструктор з передачею класу для збереження та часом його життя |
| public string Name { get; set; } | Поле передає та ініціалізує час ім’я продукту |
| public int Happines { get; set; } | Поле передає та ініціалізує час показник щастя |
| public int Poisoning { get; set; } | Поле передає та ініціалізує час показник отруєння |
| public int Saturation { get; set; } | Поле передає та ініціалізує час показник насичення |
| public int Heal { get; set; } | Поле передає та ініціалізує час показник здоров'я |
| public int timeOflife { get; set; } | Поле передає та ініціалізує час показник час життя |

### **3.1.7 Клас ExceptionController**

Вкладений клас для обробки виняткових ситуацій. Оброблює ситуації, коли може відбутися аварійне вимкнення програми.

Таблиця 3.7 – Опис полів та методів інтерфейсу ExceptionController

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public ExceptionController()  : base("Error") | Iніціалізує конструктор класу винятків із повідомленням про помилку. |
| public ExceptionController(string message\_)  : base(message\_) | Iніціалізує конструктор класу винятків із зазначеним повідомленням про помилку. |

### **3.1.8 Класс MainWindow**

Клас унаслідується від класу Window, який відповідає за графічний інтерфейс програми. В ньому відбуваються усі основні обробки подій, таймери, передача даних з інших класів до інтерфейсів.

Таблиця 3.8 – Опис полів та методів інтерфейсу MainWindow

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public MainWindow() | Конструктор класу, в ньому відбувається ініціалізація усіх необхідних компонентів для роботи програмного забезпечення, а саме: завантаження даних, ініціалізація списку продуктів та запуск таймерів. |

Продовження таблиці 3.8

|  |  |
| --- | --- |
| Поля та методи класу | Опис |

|  |  |
| --- | --- |
| ***private:*** | |
| private int CompareTamagochJson(TamagochiJson t1, TamagochiJson t2) | Компаратор для сортування елементів для таблиці рейтингу |
| private void LoadGame\_Click(object sender, RoutedEventArgs e) | Метод для завантаження покинутої гри |
| private void Window\_Closing(object sender, CancelEventArgs e) | Метод для закриття гри |
| private void TimerOfLife\_Tick(object sender, EventArgs e) | Метод вираховує час життя та передає його до інтерфейсу |
| private void Timer\_Eat(object sender, EventArgs e) | Метод додає параметри їжі до класу Tamagochi та додає їх до інтерфейсу. |
| private void Start\_Tick(object sender, EventArgs e) | Метод для появи стартової форми |
| private void End\_Tick(object sender, EventArgs e) | Метод для появи форми смерті персонажа гри |
| private void eat\_List\_MouseDoubleClick(object sender, MouseButtonEventArgs e) | Метод для додання обраної їжі до eat\_List |
| private void Button\_Plus\_Click(object sender, RoutedEventArgs e) | Метод для додання «+» до eat\_List |
| private void GifOfDuckStandart\_MouseDown(object sender, MouseButtonEventArgs e) | Метод для виклику gif-зображення при кліку на персонажа |

Продовження таблиці 3.8

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| private void TimerOfPetting\_Tick(object sender, EventArgs e) | Повертає стандартне gif-зображення |
| private void TimerEatGif\_Tick(object sender, EventArgs e) | Метод для виклику gif-зображення при годуванні персонажа |
| private void Timer\_Tick(object sender, EventArgs e) | Метод для появи меню |
| private void Pause(bool check) | Метод для зупинення деяких таймерів програми |
| private void UnPause() | Метод для продовження деяких таймерів програми |
| private void Developers\_Tick(object sender, EventArgs e) | Метод для появи форми з розробниками |
| private void Instruction\_Tick(object sender, EventArgs e) | Метод для появи форми з інструкцією |
| private void Top\_Tick(object sender, EventArgs e) | Метод для появи форми з рейтингом |
| private void Restart\_Game(object sender, RoutedEventArgs e) | Метод для появи для перезавантаження гри |
| private void Button\_Menu(object sender, RoutedEventArgs e) | Метод для старту таймеру для меню |
| private void Developers\_Btn(object sender, RoutedEventArgs e) | Метод для старту таймеру для форми з розробниками |
| private void instruction\_Btn(object sender, RoutedEventArgs e) | Метод для старту таймеру для форми з інструкцією |

Продовження таблиці 3.8

|  |  |
| --- | --- |
| private void Top\_Btn(object sender, RoutedEventArgs e) | Метод для старту таймеру з рейтингом спроб |
| private void Start\_Game(object sender, RoutedEventArgs e) | Метод для ініціалізації початкових полів |
| private void MenuPanel\_MouseDown(object sender, MouseButtonEventArgs e) | Метод дозволяє перетягувати вікно гри |
| private void Timer\_Take\_Eat(object sender, EventArgs e) | Метод рандомно видає один з тридцяти класів IFood |
| private void GetFood(object sender, RoutedEventArgs e) | Основний метод для приготування та поїдання їжі персонажем |

### **3.1.9 Клас Item**

Цей клас зберігає інформація про класи наслідувані від інтерфейсу IFood, та потім поміщається у InvertoryController.

Таблиця 3.9 – Опис полів та методів інтерфейсу TamagochiXML

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public Item() | Первинний конструктор |
| public Item(IFood food\_, int amount\_ = 1) | Конструктор з передачею классу IFood та кількістю |

Продовження таблиці 3.9

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| public static bool operator ==(Item this\_, Item other\_) | Перевантажений оператор для перевірки рівності |
| public static bool operator !=(Item this\_, Item other\_) | Перевантажений оператор для перевірки нерівності |

## **3.2 Основні розроблені алгоритми**

У ході написання було курсової роботи було розроблено алгоритм для роботи зі символьним рядком, що надає функціонал для приготування продуктів та запису їх у eat\_List, та алгоритм сортування рейтингу спроб.

Алгоритм реалізує основне з завдань програми, а саме приготування їжі та годування персонажа.

Вхідними даними для алгоритму є:

* Рядок String

У кінці виконання алгоритм формує вихідні данні:

* Дані у клас Tamagochi або нову страву у eat\_List, також змінюється gif-зображення в залежності від переданих даних у клас Tamagochi.

## **3.3 Основні рішення щодо розробки інтерфейсу**

Головним рішенням ,щодо розробки інтерфейсу, було вибрати нову та сучасну платформу для розробки інтерфейсу WPF (Windows Presentation Foundation). Ця платформа відзначається кросплатформеністю та більш зручним інтерфейсом для програміста.

## **3.4 Основні рішення щодо роботи з даними**

Основним рішенням щодо роботи з даними було використання XML та JSON форматів для зберігання даних у файлах.

JSON використовується під час виходу з програми, зчитує з файлу інформацію у список, додає об’єкт спеціального формату до нього, і перетворює список у JSON формат, та записує у файл. При цьому з цього файлу, під час запуску програми отримується JSON строка, яка представляє собою список об’єктів, перетворюється на список, сортирується, і об’єкти цього списку передаються у таблицю лідерів, яких можна побачити відкрив відповідне вікно

XML використовується для зберігання інформації про Тамагочі під час виходу з програми. При цьому, якщо вихід було зроблено не під час смерті, то при наступному запуску програми інформація у форматі XML буле зчитана з файлу, та з’явиться можливість завантажити Тамагочі з цього файлу, не починаючи гру з самого початку

## **3.5 Обробка виключних ситуацій**

Текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст, текст.

# **4 КЕРІВНИЦТВО ПРОГРАМІСТА**

В даному розділі розглянуті призначення, умови застосування, характеристика програми, звертання до програми, початкові та вихідні дані та представлені повідомлення.

## **4.1 Призначення та умови застосування програми**

### **4.1.1 Призначення програми**

Дане програмове забезпечення призначене для розваг у вільний час, розвитку реакції, уваги, логіки та аналітичного мислення

### **4.1.2 Функції програми**

Функції, що виконує програма:

* Симуляція домашнього улюбленця, котрого треба годувати їжею в певний час.

### **4.1.3 Умови застосування програми**

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* операційна система Windows 7, 8, 10, 11;
* наявність дисплею;
* наявність миші та клавіатури;
* до 100 мб оперативної пам’яті для використання програми.

## **4.2 Характеристика програми**

Програма виконана за допомогою мови програмування високого рівня С# в середовищі розробки Visual Studio 2022 за допомогою платформи WPF.

### **4.2.1 Структура програми**

Структура програми складається з 37 класів, 36 з яких – користувацькі, 1 – клас форм. Один з користувацьких класів є інтерфейсом. Було використано інтерфейс замість абстрактного класу, так як у наслідуваних классах не потрібно було реалізовувати один або більше методів, а писати свої характеристики та методи. Діаграми користувацьких класів знаходяться у додатку А.

## **4.3 Звертання до програми**

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми. Потрібно упевнитися у наявності всіх файлів програми.

Звертання до програми передбачене одним способом: через виконуваний файл Tamagochi\_WPF.exe. Звертання до програми через командний рядок не передбачене, адже програма розроблена саме для роботи через інтерфейс для складання розкладу.

## **4.4 Вхідні та вихідні дані**

### **4.4.1 Вхідні дані**

Вхідними даними є файл leaders.txt та tamagochi.xml. У першому файлі збережена інформація про лідерів у вигляді JSON-строки (цей файл буде використаний для заповнення таблиці лідерів), а у другому інформація про персонажа (буде використаний, якщо персонаж має ім’я і відповідну кількість здоров’я, для завантаження гри не з початку)

### **4.4.2 Вихідні дані**

Вихідними даними є файл leaders.txt та tamagochi.xml. У першому файлі буде збережено данні про персонажа після його смерті, а у другий буде збережено дані персонажа під час закриття програми

# **5 КЕРІВНИЦТВО КОРИСТУВАЧА**

В даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

## **5.1 Призначення програми**

Програма «Ducky» призначена для розваг і відпочинку, розвитку реакції, уваги, логіки та аналітичного мислення

## **5.2 Умови виконання програми**

### **5.2.1 Апаратні вимоги програми**

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* операційна система Windows 7, 8, 10, 11;
* наявність дисплею;
* наявність миші та клавіатури;

до 100 мб оперативної пам’яті для використання програми.

### **5.2.2 Вимоги до користувача**

До користувачів висуваються наступні вимоги:

* вміння використовувати клавіатуру та мишу;
* знання англійської мови;
* вік більший за 4 роки.

## **5.3 Виконання програми**

### **5.3.1 Запуск програми**

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми.

Звертання до програми передбачене одним способом: через виконуваний файл. Звертання до програми через командний рядок не передбачене.

Після запуску виконуваного файлу Tamagochi WPF.exe з’являється головна форма програми, що свідчить про початок роботи з програмою.

### **5.3.2 Виконання роботи з програмою**

Спершу користувач має відкрити програму. Після чого, якщо є кнопка “load”, за допомогою якої можна завантажити попередню гру (якщо не було смерті персонажа). Також, при кожному запуску програми буде поле вводу ім’я, де можна ввести ім’я персонажа, та підтвердити його, натиснувши на кнопку “DuckDuckGo”. Далі відбудеться запуск гри, в якій потрібно вводити їжу до відповідного поля, після чого підтверджувати її за допомогою кнопки “Feed”, ставити + щоб додати інгредієнт за допомогою кнопки “Plus” чи обирати їжу з списку подвійним натисканням миші. Є кнопка меню, яка відкривається подвійним натисканням. У меню є 5 пунктів, перші 3 з яких відкриваються за допомогою подвійного натискання кнопки миші, а 2 інших є кнопками, які виконують відповідні дії. Перший пункт меню – відкриває сторінку з розробниками, другий – інструкцію з використання, третій – відкриває таблицю з лідерами, 4 – починає гру з початку, а 5 – закриває меню. Перші три пункти меню відкривають вікна, закрити які можна за допомогою хрестика у правому верхньому куті. Так само можна закрити основну форму програми

## **5.4 Повідомлення користувачу**

В програмі є декілька повідомлень. Одне з них виникає під час смерті персонажа (рис. 5.4.1), а інше під час введення неправильного інгредієнту (рис. 5.4.2)

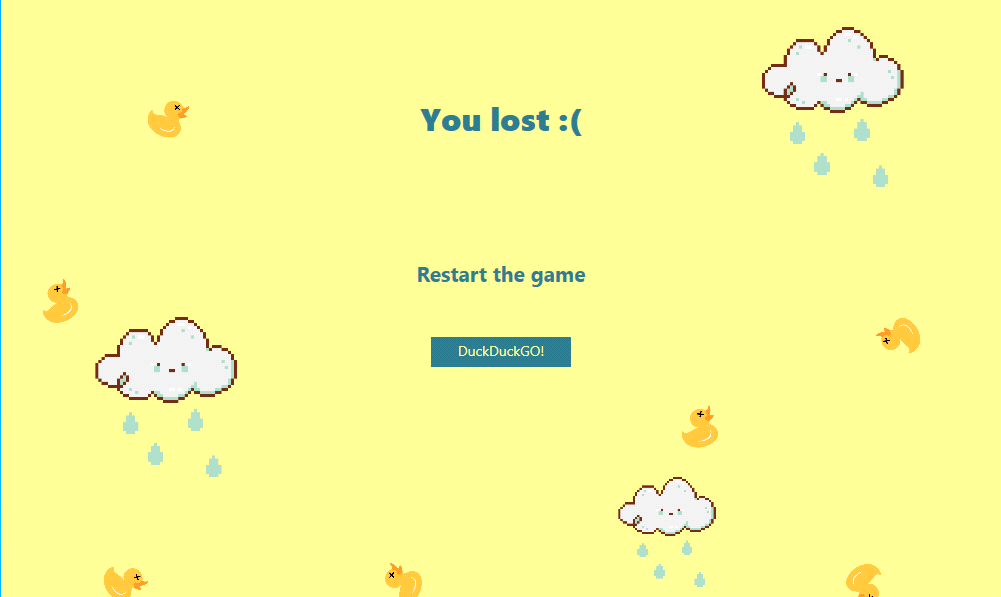


Рисунок 5.4.1 – Повідомлення про смерть персонажа

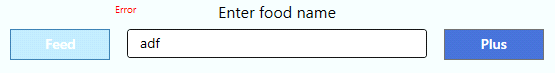


Рисунок 5.4.1 – Повідомлення про неправильне введення інгредієнту

## **5.5 Довідка програми**

Для зручності користувача та для отримання допомоги в будь-який момент часу було створено довідку для програми. Форма довідки зображена на рисунку 5.5.1

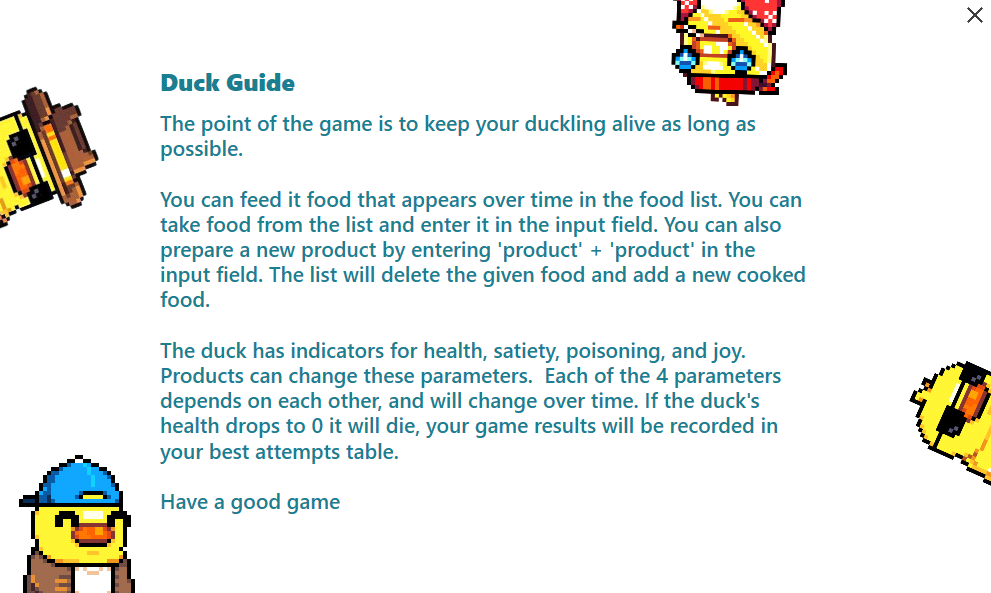


Рисунок 5.5.1 – Довідка програми

# **ВИСНОВКИ**

Під час виконання курсового проекту було розроблено гру симулятор домашнього улюбленця.

Було виконано огляд існуючих методів вирішення завдання та аналогів, проаналізовано переваги та недоліки і розроблено функціонал власного програмного продукту.

Для програми було реалізовано класи для роботи життєвого циклу персонажу та його годування.

Було розроблено алгоритм обробки рядка для годування домашнього улюбленця, основними ознаками якими є:

* результат приготування їжі або годування персонажу;
* можливість редагування рядка.

Було розроблено простий та зрозумілий інтерфейс, який підходить для людей різного віку.

Одною з ключових переваг розробленого продукту було закладення засад для подальшого розвитку та модернізації гри симулятора. Зважаючи на актуальність технологій, які були використані, якісне проектування ієрархії класів, алгоритму та інтерфейсу, цей програмний продукт має усі перспективи до подальшої розробки, а саме:

* створення повноцінної гри АА-класу;
* можливість редагування свого персонажу.

# **ПЕРЕЛІК ПОСИЛАНЬ**

1) Кормен Т., Лейзерсон Ч., Ривест Р. Глава 16. Динамическое программирование // Алгоритмы: построение и анализ(третье издание). — М.: «МІТ», 1990 — С. 299—317.

2) Кормен Т., Лейзерсон Ч., Ривест Р. Введение // Алгоритмы: построение и анализ(третье издание). — М.: «МІТ», 1990 — С. 7—17.

3) Mark Allen Weiss, Using a Table Instead of Recursion // Data Structures and Algorithm in C++(fourth edition). — Florida International University, 2013. –482 – 485p.

4) Шилдт Г. Полный справочник по C# / Ручко Н.М. — М.: «Вильямс», 2005. — 748 с.

5) «Евристичні алгоритми» [Електрон. ресурс]. – Режим доступу: [http HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html":// HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"mmsa HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html". HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"kpi HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html". HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"ua HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"/ HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"sancho HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"/ HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"ASD HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"\_ HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"HTM HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"/ HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"Arti HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"04. HYPERLINK "http://mmsa.kpi.ua/sancho/ASD\_HTM/Arti04.html"html](http://mmsa.kpi.ua/sancho/ASD_HTM/Arti04.html).

6) “What Is SQLite?” [Electronic resource]. – Access mode: <https://www.sqlite.org/index.html>.

7) “Free Spire.XLS for .NET” [Electronic resource]. – Access mode: <https://www.e-iceblue.com/Introduce/free-xls-component.html#.XfmogfxoRPY>.

8) “ADO.NET” [Electronic resource]. – Access mode: <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/>.

9) «Алгоритм лінійного пошуку» [Електрон. ресурс]. – Режим доступу: <http://kytok.org.ua/?p=311>.

**Додаток Б Код Програми**

**Б1 - Клас Tamagochi**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace Tamagochi\_WPF

{

public class Tamagochi

{

private string name\_;

public string Name { get { return name\_; } set { name\_ = value; } }

private int happines\_;

public int Happines { get { return happines\_; } set { happines\_ = value; } }

private int poisoning\_;

public int Poisoning { get { return poisoning\_; } set { poisoning\_ = value; } }

private int saturation\_;

public int Saturation { get { return saturation\_; } set { saturation\_ = value; } }

private int heal\_;

public int Heal { get { return heal\_; } set { heal\_ = value; } }

private bool is\_alive\_;

public bool IsAlive { get { return is\_alive\_; } set { is\_alive\_ = value; } }

public InventoryController Inventory;

public Tamagochi(List<IFood> food\_)

{

StateCreate();

Inventory = new InventoryController(food\_);

}

public void StateCreate()

{

try

{

heal\_ = 100;

poisoning\_ = 0;

saturation\_ = 100;

happines\_ = 100;

is\_alive\_ = true;

}

catch (ExceptionController ex)

{

Console.WriteLine(ex.Message);

return;

}

}

// викликається під час створення персонажа. (╯✧▽✧)╯

public void StateDestroy()

{

try

{

name\_ = "";

if (!is\_alive\_)

{

throw new ExceptionController("Error: You are dead");

}

heal\_ = 0;

poisoning\_ = 0;

saturation\_ = 0;

happines\_ = 0;

is\_alive\_ = false;

}

catch (ExceptionController ex)

{

Console.WriteLine(ex.Message);

return;

}

}

// викликається за смерті персонажа.(╯✧▽✧)╯

public void StateUpdate()

{

if (heal\_ <= 0)

{

is\_alive\_ = false;

//StateDestroy();

}

if (saturation\_ > 75)

{

heal\_ += 5;

happines\_ += 3;

}

if (poisoning\_ > 0)

{

heal\_ -= 4;

poisoning\_ -= 1;

}

if (happines\_ <= 25)

{ poisoning\_ += 1; heal\_ -= 5;}

//poisoning\_ = poisoning\_ + Math.Abs(happines\_);

if (saturation\_ <= 75)

heal\_ -= 10;

happines\_ -= 5;

saturation\_ -= 5;

if (heal\_ > 100) heal\_ = 100;

if (poisoning\_ > 100) poisoning\_ = 100;

if (saturation\_ > 100) saturation\_ = 100;

if (happines\_ > 100) happines\_ = 100;

if (happines\_ < 0) happines\_ = 0;

if (heal\_ < 0) heal\_ = 0;

if (poisoning\_ < 0) poisoning\_ = 0;

if (saturation\_ < 0) saturation\_ = 0;

}

//викликається на кожен тик ігрового часу (кожні 3\_сек. напевно).

public void Eat(IFood food\_)

{

try

{

if (!is\_alive\_)

{

throw new ExceptionController("Error: You are dead");

}

if ((heal\_ + food\_.Heal) < 100)

heal\_ += food\_.Heal;

else

heal\_ = 100;

if ((saturation\_ + food\_.Satiety) < 100)

saturation\_ += food\_.Satiety;

else

saturation\_ = 100;

if ((poisoning\_ + food\_.Poison) < 100)

poisoning\_ += food\_.Poison;

else

poisoning\_ = 100;

if ((happines\_ + food\_.Happy) < 100)

happines\_ += food\_.Happy;

else

happines\_ = 100;

Console.WriteLine("Вы покушац");

}

catch (ExceptionController ex)

{

Console.WriteLine(ex.Message);

return;

}

}

// їм. (o･ω･o)

}

}

**Б2 - Клас InvertoryCiontroller та Item**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.InteropServices;

using Tamagochi\_WPF;

namespace Tamagochi\_WPF

{

public class Item

{

public IFood food;

public int amount;

public Item() { }

public Item(IFood food\_, int amount\_ = 1)

{

food = food\_;

amount = amount\_;

}

public static bool operator ==(Item this\_, Item other\_)

{

return this\_.food == other\_.food;

}

public static bool operator !=(Item this\_, Item other\_)

{

return this\_.food != other\_.food;

}

}

public class InventoryController // (-\_\*)

{

public List<Item> \_items = new List<Item>();

// Список продуктів у інвентарі

public InventoryController(List<IFood> foods\_)

{

foreach (IFood food in foods\_)

{

\_items.Add(new Item(food));

}

}

// Test U( t \_ t )U

public IFood Craft(string ingredient\_right\_, string ingredient\_left\_)

{

if (!CheckItem(ingredient\_right\_) || !CheckItem(ingredient\_left\_))

{

return new Trash();

}

foreach (Item item in this.\_items)

{

if (item.food.Recipe == null)

{

continue;

}

if ((item.food.Recipe[0] == ingredient\_right\_ && item.food.Recipe[1] == ingredient\_left\_) || (item.food.Recipe[1] == ingredient\_right\_ && item.food.Recipe[0] == ingredient\_left\_))

{

foreach (string item\_name in item.food.Recipe)

{

this.\_items.Find(obj => obj.food.Name == item\_name).amount -= 1;

if (\_items.Find(obj => obj.food.Name == item\_name).amount == 0)

{

\_items.Remove(\_items.Find(obj => obj.food.Name == item\_name));

}

}

return item.food;

}

}

return new Trash();

}

/\* private bool CheckItem(IFood item\_, int amount\_ = 1)

{

foreach (Item item in this.\_items)

{

if (item\_.Name == item.food.Name && item.amount >= amount\_)

{

return true;

}

}

return false;

}

\*/

public void Add(IFood food\_, int amount\_ = 1)

{

if (CheckItem(food\_))

{

for (int index = 0; index < amount\_; index++)

{

\_items.Find(obj => obj.food.Name == food\_.Name).amount += 1;

}

}

else

{

\_items.Add(new Item(food\_, amount\_));

}

}

public void Remove(IFood food\_)

{

try

{

if (\_items.Count == 0)

{

throw new ExceptionController("Error: Product list is empty.");

}

// помилка список продуктів порожній.

\_items.Remove(\_items.Find(obj => obj.food == food\_));

//видалення продукту із інвентарю

}

catch (ExceptionController ex)

{

Console.WriteLine(ex.Message);

return;

}

}

public bool CheckItem(IFood food\_)

{

foreach (Item item in \_items)

if (item.food.Name == food\_.Name && item.amount > 0) return true;

return false;

}

public bool CheckItem(string name\_)

{

foreach (Item item in \_items)

if (item.food.Name == name\_ && item.amount > 0) return true;

return false;

}

private bool CheckCrafting(IFood food\_)

{

foreach (string name in food\_.Recipe)

if (CheckItem(name)) return false;

return true;

}

// перевірка наявності всіх інгредієнтів для крафту продукту

public void Clear()

{

\_items.Clear();

}

}

}

**Б3 – Інтерфейс Food та його похідні класи**

public interface IFood

{

string Name { get; } // имя продукта

int Heal { get; } // количество здоровья которое востанавливает продукт

int Poison { get; } // количество отравление продуктом

int Happy { get; } // количество счастья

int Satiety { get; } // количество насыщения

string[] Recipe { get; } // рецепт а точнеее список необходимых ингредиентов

bool HasRecipe(); // является ли объект приготовляемый(true) или базовым(false) продуктом

}

public class Sugar : IFood

{

private string \_name = "sugar";

string IFood.Name { get { return \_name; } }

private int \_heal = -10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 5;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 5;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Salt : IFood

{

private string \_name = "salt";

string IFood.Name { get { return \_name; } }

private int \_heal = -10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 5;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 5;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Water : IFood

{

private string \_name = "water";

string IFood.Name { get { return \_name; } }

private int \_heal = 0;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -25;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 5;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 0;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Fire : IFood

{

private string \_name = "fire";

string IFood.Name { get { return \_name; } }

private int \_heal = -40;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 20;

int IFood.Poison { get { return \_poison; } }

private int \_happy = -30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 0;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Duck : IFood

{

private string \_name = "duck";

string IFood.Name { get { return \_name; } }

private int \_heal = 30;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 50;

int IFood.Poison { get { return \_poison; } }

private int \_happy = -30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 40;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Corn : IFood

{

private string \_name = "corn";

string IFood.Name { get { return \_name; } }

private int \_heal = 5;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 10;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 5;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Fish : IFood

{

private string \_name = "fish";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 20;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Egg : IFood

{

private string \_name = "egg";

string IFood.Name { get { return \_name; } }

private int \_heal = 10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 5;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Mushroom : IFood

{

private string \_name = "mushroom";

string IFood.Name { get { return \_name; } }

private int \_heal = 10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 15;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Marshmallow : IFood

{

private string \_name = "marshmallow";

string IFood.Name { get { return \_name; } }

private int \_heal = 10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Vegetable : IFood

{

private string \_name = "vegetable";

string IFood.Name { get { return \_name; } }

private int \_heal = 15;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 15;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 15;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Flakes : IFood

{

private string \_name = "flakes";

string IFood.Name { get { return \_name; } }

private int \_heal = 25;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "corn", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Bread : IFood

{

private string \_name = "bread";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 5;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "corn", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Pie : IFood

{

private string \_name = "pie";

string IFood.Name { get { return \_name; } }

private int \_heal = 35;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 50;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 25;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "bread", "berry" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Jam : IFood

{

private string \_name = "jam";

string IFood.Name { get { return \_name; } }

private int \_heal = 15;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "berry", "sugar" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Compote : IFood

{

private string \_name = "compote";

string IFood.Name { get { return \_name; } }

private int \_heal = 5;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -20;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 20;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "berry", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Jelly : IFood

{

private string \_name = "jelly";

string IFood.Name { get { return \_name; } }

private int \_heal = 15;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -15;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 55;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 40;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "jam", "marshmallow" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Omelette : IFood

{

private string \_name = "omelette";

string IFood.Name { get { return \_name; } }

private int \_heal = 10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "egg", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Pancake : IFood

{

private string \_name = "pancake";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -10;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 40;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 40;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "jelly", "bread" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Toast : IFood

{

private string \_name = "toast";

string IFood.Name { get { return \_name; } }

private int \_heal = 10;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 10;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 10;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "corn", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class GrilledVegetables : IFood

{

private string \_name = "grilled\_vegetables";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -10;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 15;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "vegetable", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Salad : IFood

{

private string \_name = "salad";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 15;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 15;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "vegetable", "vegetable" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Steak : IFood

{

private string \_name = "steak";

string IFood.Name { get { return \_name; } }

private int \_heal = 35;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -5;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 40;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 35;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "duck", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Rice : IFood

{

private string \_name = "rice";

string IFood.Name { get { return \_name; } }

private int \_heal = 20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 20;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 20;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "corn", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class RiceVegetables : IFood

{

private string \_name = "rice\_vegetables";

string IFood.Name { get { return \_name; } }

private int \_heal = 30;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -10;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 35;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 30;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "rice", "vegetable" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Popcorn : IFood

{

private string \_name = "popcorn";

string IFood.Name { get { return \_name; } }

private int \_heal = 25;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 0;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 25;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 25;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "rice", "fire" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class VegetableSoup : IFood

{

private string \_name = "vegetable\_soup";

string IFood.Name { get { return \_name; } }

private int \_heal = 25;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -30;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 40;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 40;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "vegetable", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class MushroomSoup : IFood

{

private string \_name = "mushroom\_soup";

string IFood.Name { get { return \_name; } }

private int \_heal = 30;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -30;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 45;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 45;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "mushroom", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class FishSoup : IFood

{

private string \_name = "fish\_soup";

string IFood.Name { get { return \_name; } }

private int \_heal = 35;

int IFood.Heal { get { return \_heal; } }

private int \_poison = -30;

int IFood.Poison { get { return \_poison; } }

private int \_happy = 50;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 50;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = { "fish", "water" };

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

public class Trash : IFood

{

private string \_name = "trash";

string IFood.Name { get { return \_name; } }

private int \_heal = -20;

int IFood.Heal { get { return \_heal; } }

private int \_poison = 20;

int IFood.Poison { get { return \_poison; } }

private int \_happy = -30;

int IFood.Happy { get { return \_happy; } }

private int \_satiety = 5;

int IFood.Satiety { get { return \_satiety; } }

private string[] \_recipe = null;

string[] IFood.Recipe { get { return \_recipe; } }

public bool HasRecipe()

{

return \_recipe != null;

}

}

**Б4 – клас ExceprionController**

internal class ExceptionController : Exception

{

public ExceptionController()

: base("Error")

{

}

public ExceptionController(string message\_)

: base(message\_)

{

}

}

**Б5 – клас TamagochiJson**

class TamagochiJson

{

public string Name { get; set; }

public int Rating { get; set; }

public TamagochiJson(string name, int timeOflife)

{

Name = name;

Rating = timeOflife;

}

public TamagochiJson()

{

Name = "";

Rating = 0;

}

}

**Б6 – клас TamagochiJson**

public partial class MainWindow : Window

{

DispatcherTimer timer, timerstart, timerEnd, timerTop , timerDevelopers, timerInstruction, timerForTamagochi, timerForTakeEat, timerOfLife, timerForGifPetting, timerForEatGif;

double MenuWidth , StartGameHeight, EndGameHeight, DelevopersGameHeight , InstructionGameHeight, TopGameHeight;

bool hiddenMenu , StartGamehidden , EndGamehidden, DelevopersGamehidden , InstructionGamehidden, TopGamehidden;

Tamagochi tamagochi;

InventoryController inventoryController;

int timeOflife;

List<IFood> food;

Random randomIndex;

TamagochiXml tamagochiXml;

string tamagochiFileName = "tamagochi.xml";

public MainWindow()

{

food = new List<IFood>();

food.Add(new Sugar());

food.Add(new Salt());

food.Add(new Water());

food.Add(new Fire());

food.Add(new Duck());

food.Add(new Corn());

food.Add(new Fish());

food.Add(new Egg());

food.Add(new Mushroom());

food.Add(new Marshmallow());

food.Add(new Vegetable());

food.Add(new Flakes());

food.Add(new Bread());

food.Add(new Pie());

food.Add(new Jam());

food.Add(new Compote());

food.Add(new Jelly());

food.Add(new Omelette());

food.Add(new Pancake());

food.Add(new Toast());

food.Add(new GrilledVegetables());

food.Add(new Salad());

food.Add(new Steak());

food.Add(new Rice());

food.Add(new RiceVegetables());

food.Add(new Popcorn());

food.Add(new VegetableSoup());

food.Add(new MushroomSoup());

food.Add(new FishSoup());

food.Add(new Trash());

inventoryController = new InventoryController(food);

tamagochi = new Tamagochi(food);

InitializeComponent();

FileInfo info = new FileInfo(leadersFileName);

if (info.Exists)

{

using (FileStream fs = new FileStream(leadersFileName, FileMode.Open))

{

List<TamagochiJson> ls = JsonSerializer.Deserialize<List<TamagochiJson>>(fs);

ls.Sort(CompareTamagochJson);

for (int i = 0; i < 8; i++)

{

if (i == ls.Count) break;

Grid grid = new Grid();

grid.HorizontalAlignment = HorizontalAlignment.Stretch;

grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.ColumnDefinitions[0].MinWidth = 550;

//grid.ShowGridLines = true;

Label name = new Label();

name.Content = ls[i].Name;

Label rating = new Label();

rating.Content = ls[i].Rating.ToString();

Grid.SetColumn(rating, 1);

grid.Children.Add(name);

grid.Children.Add(rating);

TopListBox.Items.Add(grid);

}

}

}

info = new FileInfo(tamagochiFileName);

if (info.Exists)

{

using (FileStream fs = new FileStream(tamagochiFileName, FileMode.Open))

{

XmlSerializer serializer = new XmlSerializer(typeof(TamagochiXml));

tamagochiXml = serializer.Deserialize(fs) as TamagochiXml;

if (tamagochiXml.Name != null)

{

if (tamagochiXml.Name.Length >= 2) loadGame.Visibility = Visibility.Visible;

}

}

}

timer = new DispatcherTimer();

timer.Interval = new TimeSpan(0, 0, 0, 0 , 1);

timer.Tick += Timer\_Tick;

timerstart = new DispatcherTimer();

timerstart.Interval = new TimeSpan (0, 0, 0, 0 , 1);

timerstart.Tick += Start\_Tick;

timerEnd = new DispatcherTimer();

timerEnd.Interval = new TimeSpan(0, 0, 0, 0, 1);

timerEnd.Tick += End\_Tick;

timerDevelopers = new DispatcherTimer();

timerDevelopers.Interval = new TimeSpan(0, 0, 0, 0, 1);

timerDevelopers.Tick += Developers\_Tick;

timerInstruction = new DispatcherTimer();

timerInstruction.Interval = new TimeSpan(0, 0, 0, 0, 1);

timerInstruction.Tick += Instruction\_Tick;

timerTop = new DispatcherTimer();

timerTop.Interval = new TimeSpan(0, 0, 0, 0, 1);

timerTop.Tick += Top\_Tick;

EndGameHeight = EndGamePanel.Height;

StartGameHeight = StartGamePanel.Height;

DelevopersGameHeight = DevelopersGamePanel.Height;

InstructionGameHeight = InstructionGamePanel.Height;

TopGameHeight = TopGamePanel.Height;

MenuWidth = sideMenu.Width;

timerForTamagochi = new DispatcherTimer();

timerForTamagochi.Interval = new TimeSpan(0, 0, 0, 2);

timerForTamagochi.Tick += Timer\_Eat;

timerForTakeEat = new DispatcherTimer();

timerForTakeEat.Interval = new TimeSpan(0, 0, 0, 5);

timerForTakeEat.Tick += Timer\_Take\_Eat;

timerOfLife = new DispatcherTimer();

timerOfLife.Interval = new TimeSpan(0, 0, 0, 15);

timerOfLife.Tick += TimerOfLife\_Tick;

timerForGifPetting = new DispatcherTimer();

timerForGifPetting.Interval = new TimeSpan(0, 0, 0, 3);

timerForGifPetting.Tick += TimerOfPetting\_Tick;

timerForEatGif = new DispatcherTimer();

timerForEatGif.Interval = new TimeSpan(0, 0, 0, 3);

timerForEatGif.Tick += TimerEatGif\_Tick;

randomIndex = new Random();

}

private int CompareTamagochJson(TamagochiJson t1, TamagochiJson t2)

{

return t2.Rating - t1.Rating;

}

private void LoadGame\_Click(object sender, RoutedEventArgs e)

{

loadGame.Visibility = Visibility.Hidden;

tamagochi.Name = tamagochiXml.Name;

tamagochi.Poisoning = tamagochiXml.Poisoning;

tamagochi.Saturation = tamagochiXml.Saturation;

tamagochi.Heal = tamagochiXml.Heal;

tamagochi.Happines = tamagochiXml.Happines;

NameOfDuck.Text = tamagochi.Name;

timeOflife = tamagochiXml.timeOflife;

Label\_AgeText.Content = "Age: " + Convert.ToString(tamagochiXml.timeOflife) + " years";

Start\_Game(sender, e);

//timerstart.Start();

}

private void Window\_Closing(object sender, CancelEventArgs e)

{

using (FileStream fs = new FileStream(tamagochiFileName, FileMode.Create))

{

TamagochiXml tXml = new TamagochiXml(tamagochi, timeOflife);

XmlSerializer serializer = new XmlSerializer(typeof(TamagochiXml));

serializer.Serialize(fs, tXml);

}

}

private void TimerOfLife\_Tick(object sender, EventArgs e)

{

timeOflife++;

Label\_AgeText.Content = "Age: " + Convert.ToString(timeOflife) + " years";

}

string leadersFileName = "leaders.txt";

private void Timer\_Eat(object sender, EventArgs e)

{

/\*tamagochi.ProgressBarOfHappy.Value;\*/

tamagochi.StateUpdate();

ProgressBarOfHungry.Value = tamagochi.Saturation;

Label\_HugerIndex.Content = tamagochi.Saturation;

ProgressBarOfHappy.Value = tamagochi.Happines;

Label\_HappinessIndex.Content = tamagochi.Happines;

ProgressBarOfHeal.Value = tamagochi.Heal;

Label\_healIndex.Content = tamagochi.Heal;

ProgressBarOfPoison.Value = tamagochi.Poisoning;

Label\_PoisoningIndex.Content = tamagochi.Poisoning;

if (!tamagochi.IsAlive)

{

FileInfo fileInfo = new FileInfo(leadersFileName);

if (fileInfo.Exists)

{

List<TamagochiJson> ls;

using (StreamReader sr = new StreamReader(leadersFileName))

{

ls = JsonSerializer.Deserialize<List<TamagochiJson>>(sr.ReadToEnd());

ls.Add(new TamagochiJson(tamagochi.Name, timeOflife));

}

using (StreamWriter sw = new StreamWriter(leadersFileName))

{

sw.Write(JsonSerializer.Serialize(ls));

}

}

else

{

using (StreamWriter sw = new StreamWriter(leadersFileName))

{

List<TamagochiJson> ls = new List<TamagochiJson>();

ls.Add(new TamagochiJson(tamagochi.Name, timeOflife));

sw.Write(JsonSerializer.Serialize(ls));

}

}

tamagochi.StateDestroy();

eat\_List.Items.Clear();

EndGamehidden = true;

timerForTamagochi.Stop();

timerForTakeEat.Stop();

timerEnd.Start();

}

}

private void Start\_Tick(object sender, EventArgs e)

{

if (StartGamehidden)

{

StartGamePanel.Visibility = Visibility.Visible;

StartGamePanel.Height += StartGameHeight / 5;

if (StartGamePanel.Height >= StartGameHeight)

{

timerstart.Stop();

StartGamehidden = false;

}

}

else

{

StartGamePanel.Height -= StartGameHeight / 5;

if (StartGamePanel.Height <= 0)

{

timerstart.Stop();

StartGamehidden = true;

StartGamePanel.Visibility = Visibility.Hidden;

timerForTamagochi.Start();

timerOfLife.Start();

timerForTakeEat.Start();

}

}

}

//таймер для виведення панелі на початку гри для задання імені персонажу

private void End\_Tick(object sender, EventArgs e)

{

if (EndGamehidden)

{

EndGamePanel.Visibility = Visibility.Visible;

EndGamePanel.Height += EndGameHeight / 5;

if (EndGamePanel.Height >= EndGameHeight)

{

timerEnd.Stop();

EndGamehidden = false;

}

}

else

{

EndGamePanel.Height -= EndGameHeight / 5;

if (EndGamePanel.Height <= 0)

{

timerEnd.Stop();

EndGamehidden = true;

EndGamePanel.Visibility = Visibility.Hidden;

tamagochi.StateCreate();

timerstart.Start();

StartGamehidden = true;

}

}

}

//рестарт гри (якщо захочемо, можна сюди ще допиляти панель з результатами, скільки їжі сів і скільки хвилин прожив

private void eat\_List\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

string s = eat\_List.SelectedItem.ToString();

if (foodText.Text == "")

{

foodText.Text = s;

} else if (foodText.Text.EndsWith("+ ") || foodText.Text.EndsWith("+"))

{

foodText.Text = foodText.Text + s;

}

}

private void Button\_Plus\_Click(object sender, RoutedEventArgs e)

{

if (foodText.Text != "")

{

if (!foodText.Text.EndsWith("+ "))

{

foodText.Text = foodText.Text + " + ";

}

else if (foodText.Text.EndsWith("+"))

{

foodText.Text = foodText.Text + " + ";

}

}

}

bool isPetting = false;

private void GifOfDuckStandart\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (!isPetting)

{

isPetting = true;

GifOfDuckPat.Visibility = Visibility.Visible;

GifOfDuckStandart.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

timerForGifPetting.Start();

}

}

private void TimerOfPetting\_Tick(object sender, EventArgs e)

{

timerForGifPetting.Stop();

GifOfDuckStandart.Visibility = Visibility.Visible;

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

isPetting =false;

}

private void TimerEatGif\_Tick(object sender, EventArgs e)

{

timerForGifPetting.Stop();

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckStandart.Visibility = Visibility.Visible;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

}

private void Timer\_Tick(object sender, EventArgs e)

{

if (hiddenMenu)

{

sideMenu.Visibility = Visibility.Visible;

sideMenu.Width += MenuWidth/5;

if (sideMenu.Width >= MenuWidth)

{

timer.Stop();

hiddenMenu = false;

}

}

else

{

sideMenu.Width -= MenuWidth/5;

if (sideMenu.Width <= 0)

{

timer.Stop();

hiddenMenu = true;

sideMenu.Visibility = Visibility.Hidden;

}

}

}

private bool checkPause = false;

private void Pause(bool check)

{

if (check)

{

timerForTamagochi.Stop();

timerForTakeEat.Stop();

timerOfLife.Stop();

}

}

private void UnPause()

{

timerForTamagochi.Start();

timerForTakeEat.Start();

timerOfLife.Start();

}

private void Developers\_Tick(object sender, EventArgs e)

{

if (DelevopersGamehidden)

{

if (checkPause == false) { checkPause = true; Pause(checkPause); }

DevelopersGamePanel.Visibility = Visibility.Visible;

DevelopersGamePanel.Height += DelevopersGameHeight / 5;

if (DevelopersGamePanel.Height >= DelevopersGameHeight)

{

timerDevelopers.Stop();

DelevopersGamehidden = false;

}

}

else

{

DevelopersGamePanel.Height -= DelevopersGameHeight / 5;

if (DevelopersGamePanel.Height <= 0)

{

checkPause = false;

UnPause();

timerDevelopers.Stop();

DelevopersGamehidden = true;

DevelopersGamePanel.Visibility = Visibility.Hidden;

}

}

}

private void Instruction\_Tick(object sender, EventArgs e)

{

if (InstructionGamehidden)

{

if (checkPause == false) { checkPause = true; Pause(checkPause); }

InstructionGamePanel.Visibility = Visibility.Visible;

InstructionGamePanel.Height += InstructionGameHeight / 5;

if (InstructionGamePanel.Height >= InstructionGameHeight)

{

timerInstruction.Stop();

InstructionGamehidden = false;

}

}

else

{

InstructionGamePanel.Height -= InstructionGameHeight / 5;

if (InstructionGamePanel.Height <= 0)

{

checkPause = false;

UnPause();

timerInstruction.Stop();

InstructionGamehidden = true;

InstructionGamePanel.Visibility = Visibility.Hidden;

}

}

}

private void Top\_Tick(object sender, EventArgs e)

{

if (TopGamehidden)

{

if (checkPause == false) { checkPause = true; Pause(checkPause); }

TopGamePanel.Visibility = Visibility.Visible;

TopGamePanel.Height += TopGameHeight / 5;

if (TopGamePanel.Height >= TopGameHeight)

{

timerTop.Stop();

TopGamehidden = false;

}

}

else

{

TopGamePanel.Height -= TopGameHeight / 5;

if (TopGamePanel.Height <= 0)

{

checkPause = false;

UnPause();

timerTop.Stop();

TopGamehidden = true;

TopGamePanel.Visibility = Visibility.Hidden;

}

}

}

private void Restart\_Game(object sender, RoutedEventArgs e)

{

// loadGame.Visibility = Visibility.Visible;

timerEnd.Start();

Label\_AgeText.Content = "Age: " + Convert.ToString(timeOflife) + " years";

Label\_Name.Content = "Name: " + tamagochi.Name;

ProgressBarOfHeal.Value = tamagochi.Heal;

ProgressBarOfHappy.Value = tamagochi.Happines;

ProgressBarOfHungry.Value = tamagochi.Saturation;

ProgressBarOfPoison.Value = tamagochi.Poisoning;

Label\_healIndex.Content = tamagochi.Heal;

Label\_HappinessIndex.Content = tamagochi.Happines;

Label\_HugerIndex.Content = tamagochi.Saturation;

Label\_PoisoningIndex.Content = tamagochi.Poisoning;

}

private void Button\_Menu(object sender, RoutedEventArgs e)

{

timer.Start();

}

private void Developers\_Btn(object sender, RoutedEventArgs e)

{

timerDevelopers.Start();

}

private void instruction\_Btn(object sender, RoutedEventArgs e)

{

timerInstruction.Start();

}

private void Top\_Btn(object sender, RoutedEventArgs e)

{

timerTop.Start();

}

private void Start\_Game(object sender, RoutedEventArgs e)

{

if (NameOfDuck.Text.Length >= 3 && NameOfDuck.Text.Length <= 20)

{

inventoryController.Clear();

timerstart.Start();

Take\_Eat();

tamagochi.Happines = 100;

tamagochi.Heal = 100;

tamagochi.Name = "";

tamagochi.Poisoning = 0;

tamagochi.Saturation = 100;

timeOflife = 0;

Label\_AgeText.Content = "Age: " + Convert.ToString(timeOflife) + " years";

tamagochi.Name = NameOfDuck.Text;

Label\_Name.Content = "Name: " + tamagochi.Name;

ProgressBarOfHeal.Value = tamagochi.Heal;

ProgressBarOfHappy.Value = tamagochi.Happines;

ProgressBarOfHungry.Value = tamagochi.Saturation;

ProgressBarOfPoison.Value = tamagochi.Poisoning;

Label\_healIndex.Content = tamagochi.Heal;

Label\_HappinessIndex.Content = tamagochi.Happines;

Label\_HugerIndex.Content = tamagochi.Saturation;

Label\_PoisoningIndex.Content = tamagochi.Poisoning;

}

}

//передача початкових параметрів в інтерфейс

private void leftDown(object sender, MouseButtonEventArgs e)

{

Application.Current.Shutdown();

}

//вихід з проги

private void MenuPanel\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (e.LeftButton == MouseButtonState.Pressed)

{

DragMove();

}

}

private void Timer\_Take\_Eat(object sender, EventArgs e)

{

int index = randomIndex.Next(0,29);

eat\_List.Items.Add(food[index].Name);

inventoryController.Add(food[index]);

}

private void Take\_Eat()

{

int index;

eat\_List.Items.Add(food[3].Name);

inventoryController.Add(food[3]);

eat\_List.Items.Add(food[4].Name);

inventoryController.Add(food[4]);

for (int i = 0; i < 5; i++)

{

index = randomIndex.Next(0, 29);

eat\_List.Items.Add(food[index].Name);

inventoryController.Add(food[index]);

}

}

//видає їжу

private void GetFood(object sender, RoutedEventArgs e)

{

String str = foodText.Text.ToLower();

if (str == "")

{

labelErrorsWithList.Content = "Error";

return;

}

if (str[str.Length - 1] == '+')

{

labelErrorsWithList.Content = "Error";

return;

}

String[] masStr = str.Split('+');

if (masStr.Length >= 3)

{

labelErrorsWithList.Content = "Error";

return;

}

if (masStr.Length == 2)

{

for (int i = 0; i < masStr.Length; i++)

{

if (masStr[i].Substring(0, 1) == " ")

{

masStr[i] = masStr[i].Remove(0, 1);

}

int ind = masStr[i].LastIndexOf(" ");

if (ind == masStr[i].Length - 1)

{

masStr[i] = masStr[i].Remove(masStr[i].Length - 1, 1);

}

}

if (!inventoryController.CheckItem(masStr[1]) || !inventoryController.CheckItem(masStr[0]))

{

labelErrorsWithList.Content = "Error";

return;

}

if (masStr[0] == masStr[1])

{

labelErrorsWithList.Content = "Product are same";

return;

}

string newFood = inventoryController.Craft(masStr[0], masStr[1]).Name;

eat\_List.Items.Remove(masStr[0]);

eat\_List.Items.Remove(masStr[1]);

eat\_List.Items.Add(newFood);

IFood dish = null;

for (int i = 0; i < food.Count; i++)

{

if (newFood == food[i].Name) { dish = food[i]; break; }

}

inventoryController.Add(dish);

foodText.Text = "";

}

else if (masStr.Length == 1)

{

if (masStr[0].Substring(0, 1) == " ")

{

masStr[0] = masStr[0].Remove(0, 1);

}

int ind = masStr[0].LastIndexOf(" ");

if (ind == masStr[0].Length - 1)

{

masStr[0] = masStr[0].Remove(masStr[0].Length - 1, 1);

}

if (!inventoryController.CheckItem(masStr[0]))

{

labelErrorsWithList.Content = "Error";

return;

}

IFood dish = null;

int index = 30;

for (int i = 0; i < food.Count; i++)

{

if (masStr[0] == food[i].Name) { dish = food[i]; index = i; break; }

}

if (index >= 0 && index <=10)

{

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckStandart.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Visible;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

timerForEatGif.Start();

} else if (index >10 && index <= 20)

{

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckStandart.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Visible;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

timerForEatGif.Start();

} else if (index > 20 && index <= 28)

{

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckStandart.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Visible;

GifOfDuckEatTrash.Visibility = Visibility.Hidden;

timerForEatGif.Start();

} else

{

GifOfDuckPat.Visibility = Visibility.Hidden;

GifOfDuckStandart.Visibility = Visibility.Hidden;

GifOfDuckEat1.Visibility = Visibility.Hidden;

GifOfDuckEat2.Visibility = Visibility.Hidden;

GifOfDuckEat3.Visibility = Visibility.Hidden;

GifOfDuckEatTrash.Visibility = Visibility.Visible;

timerForEatGif.Start();

}

inventoryController.Remove(dish);

tamagochi.Eat(dish);

eat\_List.Items.Remove(masStr[0]);

foodText.Text = "";

}

}

//їсть їжу

}