

**Черкаський національний університет імені Богдана
Хмельницького**

Кафедра програмного забезпечення автоматизованих систем

КУРСОВА РОБОТА

**з дисципліни “Програмування та алгоритмічні мови”
НА ТЕМУ «Королівська балда - пошук найкращого ходу у грі»
Студента 1 курсу, групи КС-19 напряму підготовки
«Програмна інженерія»
спеціальності
«Програмне забезпечення систем»**

(прізвище та ініціали)

Керівник _____

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

Національна шкала: _____

Кількість балів: _____ **Оцінка: ECTS** _____

Члени комісії

_____ (підпис)	_____ (прізвище та ініціали)
_____ (підпис)	_____ (прізвище та ініціали)
_____ (підпис)	_____ (прізвище та ініціали)

м. Черкаси – 2020 рік

Зміст

Зміст.....	B
Вступ.....	1
Розділ 1 - Огляд інформаційних джерел та вибір алгоритмів пошуку найкращого ходу у грі “Балда”.....	3
1.2. Вступ до розділу 1.....	3
1.2. Огляд алгоритмів гри та вибір алгоритмів для гри.....	3
1.3. Висновок до 1 розділу.....	5
Розділ 2 - Проектування алгоритмів та будування схем.....	6
2.1. Вступ до 2 розділу.....	6
2.2. Проектування гри та методів.....	6
2.3. Проектування алгоритму пошуку.....	6
2.4. Будування блок-схем алгоритмів.....	7
2.5. Висновок до 2 розділу.....	8
Розділ 3 – реалізація програмного продукту.....	9
Створення користувацького інтерфейсу.....	9
Створення програмного коду та його тестування.....	11
Висновок до 3 розділу.....	18
Висновки.....	19
Використані Інформаційні джерела.....	20
Додаток А. Блок-схема виведення.....	A
Додаток Б. Блок-схема пошуку слів.....	B
Додаток В. Блок-схема введення в поле.....	C

Вступ

Балда це досить цікава гра в яку можна пограти з друзями, вам лише потрібен листок паперу та знати правила гри, які дуже прості.

Мета цієї курсової роботи: Розробити гру “Королівська Балда” на комп’ютер, для цього нам потрібно:

1. Проєктування гри (будування блок-схем алгоритму пошуку слова(згідно з усіма правилами гри) та введення/виведення)
2. реалізувати введення букв та виведення результату пошуку слів.
3. розробити алгоритм пошуку слова згідно з усіма правилами гри.
4. зробити графічний інтерфейс гри.

Балда - лінгвістична(жанр гри) настільна гра в якій необхідно скласти слова за допомогою літер, що додаються клавіатурою або вибором літер на квадратне ігрове поле.

Настільна гра - це гра в якій здійснюються маніпуляції з ігровими предметами(в нас маніпуляції з буквами) на плоскій поверхні, переважно ці ігри є інтелектуальними.



Приклад настільної гри “Монополія”

Інтерфейс - це сукупність засобів, методів та правил взаємодії між елементами системи. Інтерфейси бувають: текстовими(консольними), графічними, вебінтерфейси, сенсорні, розмовні, апаратні, інтелектуальні і так далі.

Інтерфейс - це сукупність засобів, методів та правил взаємодії між елементами системи. Інтерфейси бувають: текстовими(консольними), графічними, вебінтерфейси, сенсорні, розмовні, апаратні, інтелектуальні і так далі.



Копія екрану GNOME Shell GUI. Див. джерело 2

HUD(heads-up display) - колекція постійних елементів на екрані, що повідомляють про статус гравця(кількість очок, здоров'я, прогрес тощо).



HUD гри Wolfenstein 3D: Панель внизу екрана показує поточний рівень, кількість очок, життів, портрет героя, кількість здоров'я та боеприпасів, поточну зброю. Див. 3 джерело.

Отже, ми поставили мету, задачі які треба виконати та ознайомилися з базовими поняттями.

Розділ 1 - Огляд інформаційних джерел та вибір алгоритмів пошуку найкращого ходу у грі “Балда”

1.2. Вступ до розділу 1

У цьому розділі ми проведемо огляд алгоритмів та методів реалізації поставленого завдання, буде проводитись аналіз оглянутих алгоритмів та вибір алгоритмів і методів для реалізації програми.



Рис.1. та Рис.2. Приклад гри Балда

1.2. Огляд алгоритмів гри та вибір алгоритмів для гри

У найбільш популярному варіанті гри, який має безліч комп'ютерних реалізацій, ігрове поле це квадратна таблиця в центрі якого міститься слово яке складається з 5 літер. Розміри поля та слів можуть бути різними, слова складаються за допомогою переходів від букви до букви під прямим кутом(по горизонталі та вертикалі), варіанті гри “Королівська балда” може бути перехід по діагоналі(Як хід королем в грі “Шахи”)(Див. 1 джерело). Також треба дотримуватися наступних правил: кожна клітинка містить одну букву, щоб виграти потрібно заповнити всі клітинки, якщо слово не було знайдено в словнику(тобто не правильно введено), додається 1 бал до програшу, а в нас, якщо балів буде 7 та

більше то буде прогаш. Заборонені слова вульгаризми, діалектизми та жаргонізми. Слова не повинні повторюватись.

А щоб зробити користувацький інтерфейс будемо працювати з інструментом для розробки ігор - Unity. Для користувацького інтерфейсу нам потрібні картинки поля та основного меню, які ми знайшли в інтернеті.

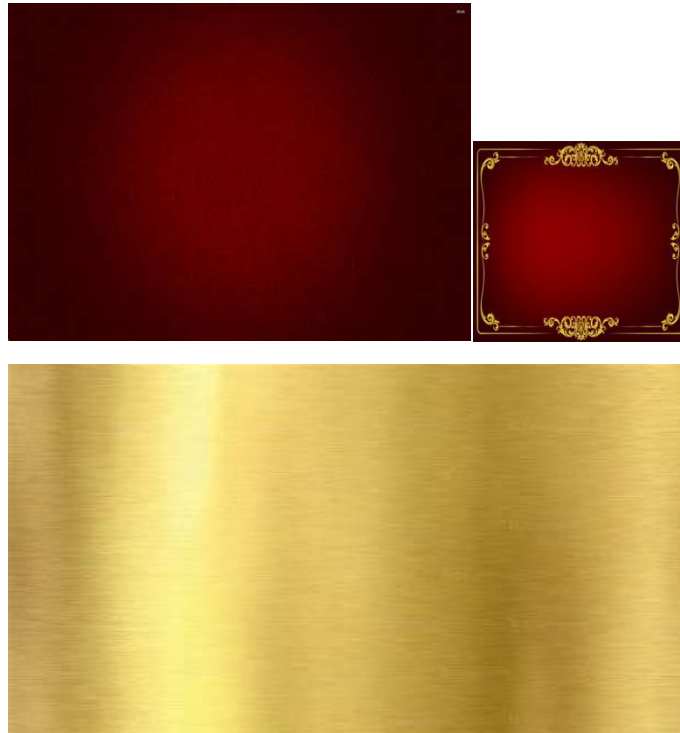


Рис.3. Картинки для меню та поля

Для того щоб знаходити слова потрібен словник слів, з якого потрібно буде брати слова та зрівнювати з тими які ми знайшли пошуком слів, тому знайдемо словник в інтернеті, для того щоб були хоча б стандартні слова нам потрібно не менше 5000 слів, а то і більше. За ідеєю слова в словнику в мене будуть англійською мовою. Я знайшов словник на гіт хабі(Див. джерело 4). В текстовому файлі слова повинні бути через рядок.

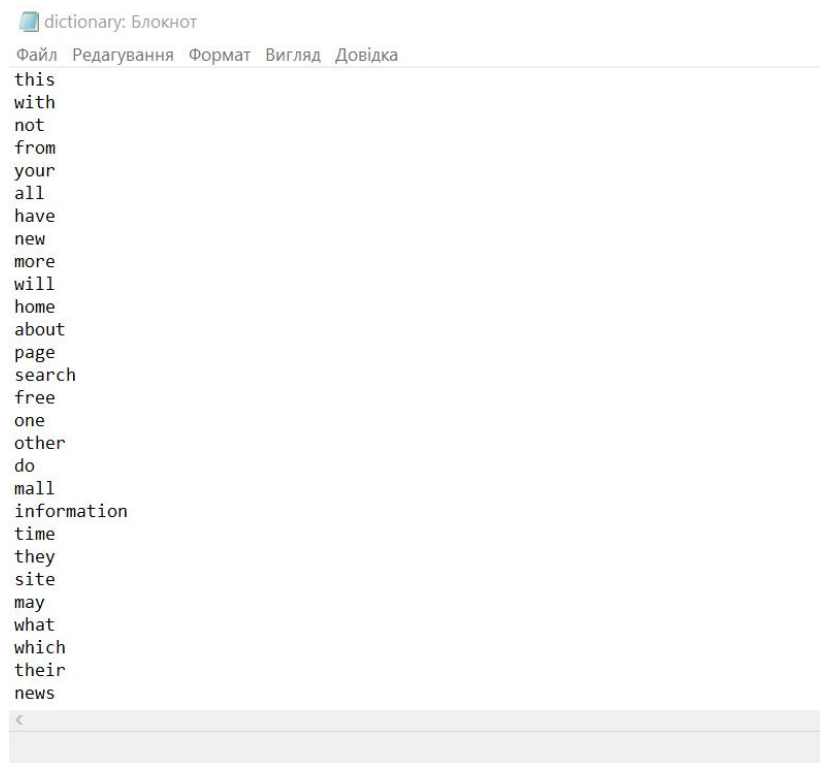


Рис.4. Приклад формату словника слів

1.3. Висновок до 1 розділу

Отже, ми ознайомились та обрали алгоритми та методи для реалізації програми, обрали інструменти якими будемо користуватися, ознайомились з правилами гри та знайшли словник слів.

Розділ 2 - Проєктування алгоритмів та будування схем

2.1. Вступ до 2 розділу

У цьому розділі буде проводитись опис алгоритмів для реалізації задачі та розроблення блок-схем алгоритмів.

2.2. Проєктування гри та методів

Отже, щоб реалізувати введення букв потрібно буде створити користувацький інтерфейс, тому що для введення нам потрібні будуть поля вводу та треба буде налаштувати щоб вводилася лише 1 буква. А для виводу результату перевірки в нас буде окреме місце в користувацькому інтерфейсі, яке буде показувати чи пошук знайшов слово, чи ні, а також де знаходиться слово в словнику. Щоб кнопки робили нам потрібне полотно на яке ми розмістимо панель, в якій будуть кнопки. Також нам потрібна система подій, яка є в юніті для дії з елементами користувацького інтерфейсу. Ще в нас буде головне меню з якого можна буде натиснути на кнопку “Play” і ми перейдемо на наступну сцену в якій буде пошук слів з введенням та виведенням. Якщо гравець програв ми переходимо на сцену де нам кажуть, що ми програди, так само з виграшем, ми перейдемо на сцену де нас вітатимуть з перемогою.

2.3. Проєктування алгоритму пошуку

Щоб зробити алгоритм пошуку треба буде спочатку знайти оптимальний алгоритм пошуку. Алгоритм пошуку у нас такий:

- 1) Програма дивиться чи змінились поля.
- 2) Якщо була введена буква ми знаходимо розташування поля.
- 3) Ми будемо блокувати шлях для знаходження поля, тобто. записувати заблокований шлях в строкову змінну, якщо він буде виходити за межі масиву(таблиці з полями).
- 4) Далі, якщо шлях не заблокований ми будемо рухатися по полях забираючи букви в строкову змінну, рухатися будемо як по координатам масиву x та

у(значення будуть протилежні, тому що масив з 0-5, 0 знаходиться зверху, а 5 внизу), рухаємося по горизонталі(праве та ліве поле $x-1$ та $x+1$), вертикалі(верхнє та нижнє поле $y-1$ та $y+1$) та діагоналі(верхнє ліве $x-1$ $y-1$, верхнє праве, нижнє праве $x+1$ $y+1$ та нижнє ліве поле). Перевірятися буде кожен шлях, який може бути. Треба зробити перевірку чи вже було перевірено це поле(якщо не буде перевірки, пошук буде рухатись туди-сюди коли не буде доступних полів). Якщо поле вже було перевірено, то ми виходимо з циклу. Також створимо масив який буде містити всі можливі комбінації слів. З кожним циклом пошуку буде створюватися нове слово, яке буде переміщено в масив.

5) Коли були отримані всі комбінації слів починаємо зрівнювати слова, які ми отримали з словами в текстовому файлі, який містить набір слів.

6) Якщо слово не було знайдено, то додається 1 до числа, якщо число набере 7, то ви програли.

7) Якщо ви заповнили всі поля без програшу, то ви виграли.

2.4. Будування блок-схем алгоритмів

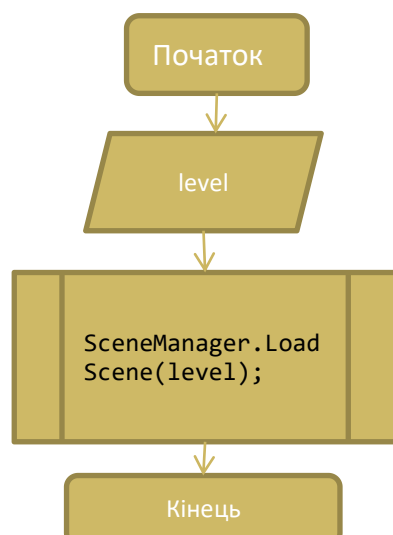
Тепер намалюємо блок-схему для введення, виведення та пошуку слів.

Блок-схеми:

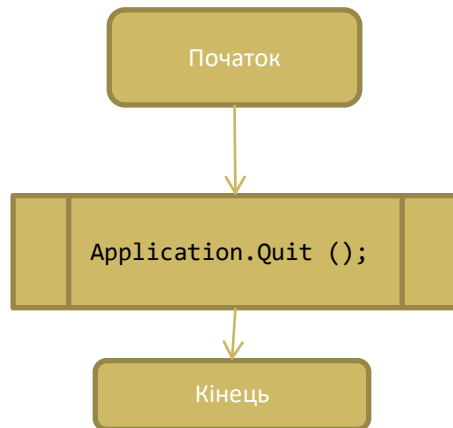
- 1) Виведення - див. Додаток А.
- 2) Пошук - див. Додаток Б.
- 3) Введення - див. Додаток В.

Також деякі схеми для меню, програшу та виграшу, виходу:

- 1) Блок-схема для програшу, перемоги та меню(level - номер сцени)



2)Блок-схема виходу з програми



2.5. Висновок до 2 розділу

Отже, ми провели опис алгоритмів для реалізації задач та розробка блок-схем, тобто ознайомилися з алгоритмом пошуку, введення, виведення та виходу з програми та побудували блок-схеми до алгоритмів і тепер ми готові для реалізації поставлених задач.

Розділ 3 – реалізація програмного продукту

В цьому розділі ми проведемо опис реалізації завдання у програмному коді, який буде доповнюватися частинами лістингу програми з коментарями, опис результатів роботи програми та тестування буде доповнюватись картинками екрану.

Створення користувацького інтерфейсу

Спочатку створимо новий 2-д проект в Unity та почнемо створювати користувацький інтерфейс для гри, для цього нам треба відкрити сцену(якщо сцена не відкрилася автоматично) та меню Hierarchy(Ієрархія) де ми створимо елементи для користувацького інтерфейсу. Нажмемо в Hierarchy ПКМ(Праву клавішу миші) далі наведемо курсор на UI(user interface) та нажмемо на Canvas(на укр. полотно) з яким також створиться EventSystem(на укр. - система подій, якщо система подій не створилася, тоді створимо систему подій власноруч, UI > EventSystem).

Тепер в полотні створимо панель, на якій буде розміщено поля для введення, для цього нажмемо на полотно та створимо панель(UI > Panel) та змінимо розмір панелі, який нам потрібен і тепер нажмемо на панель і створимо поля для введення (UI > InputField). Тепер, коли в нас кількість полів - 35 нам треба перейти в режим Scene та поставити їх, як таблицю та змінити розмір, щоб всі поля були квадратні.

Коли в нас є вже таблиця з полями ми змінимо всі поля, щоб туди вводилася тільки одна буква. Клацаємо на 1 поле в списку в Hierarchy та зажмемо кнопку Shift та клацнемо на останнє поле в списку, так ми вибрали всі поля. Тепер переходимо в меню Inspector, де ми змінимо параметри Character Limit на 1 як показано на рис.5., і тепер в середину таблиці запишемо будь-яке слово англійською(я просто ввів BALDA, тобто з 16-21 поле я ввів по одній букві і не забудемо виключити параметр interactable для того щоб ми не змогли змінити

поле). Також змінимо вид полотна, панелі та полів для цього клацнемо на Target Graphic в Inspector та оберемо картинку, можна побачити на рис.5.

Ми зробили елементи для введення, тепер зробимо елементи для виведення. Створимо поле в якому буде показуватись потрібна інформація та можна буде прокручувати поле вверх або вниз, якщо інформація виходить за межі поля, щоб це зробити нам потрібно створити Scroll View(UI > Scroll View). Я зробив Scroll View прозорим, завдяки зміні альфи кольору в параметрі Color.

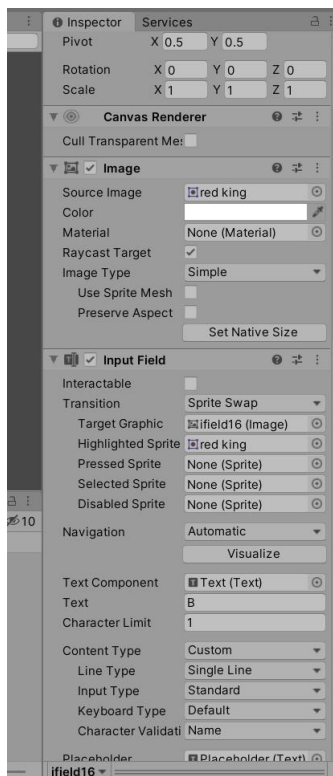


Рис.5. меню Inspector в Unity

Щоб зробити меню гри треба створити нову сцену, в якій ми зробимо меню. Скопіюємо з 1 сцени полотно та панель, але видалимо поля які скопіювалися, вони нам не потрібні і створимо EventSystem, якщо цього елементу в нас не має.

Тепер створимо кнопки (UI > Button) для того, щоб зіграти і вийти з гри. Поставимо кнопки по середині та можемо поміняти колір та вид в інспекторі. Я зробив кнопки прозорими, поміняв шрифт, який можна змінити в інспекторі та ввів текст в кнопку. Також хотілося б, щоб в меню була назва. Я створив текст(UI > Text) по середині і ввів назву.

Щоб створити сцену програшу ми створюємо нову сцену і вставимо туди полотно з панеллю та систему подій. Створимо дві кнопки для переходу в меню та для того, щоб заново пограти в гру та налаштуємо їх. Створимо текст, в якому напишемо слова програшу. Для сцени виграшу створимо сцену та вставимо всі елементи сцени програшу та змінимо текст.

Створимо HUD нарахування для очок програшу. Створимо 2 текстових елементи, в першому просто напишемо failed word count, а в другому нічого змінювати не потрібно. Тепер користувацький інтерфейс створено.

Створення програмного коду та його тестування

Створимо код, в якому ми створимо масив, в якому будуть зберігатися букви з полів, щоб передавати букви в масив я скористаюсь методом Update()(Рис.6.), який буде оновлювати масив кожен кадр, метод Start() виконує код на початку запуску програми.

```
public void Start()
{
    Update();
}

public void Update()
{
    arrIF[0, 0] = iField1;
    arrIF[0, 1] = iField2;
    arrIF[0, 2] = iField3;
    arrIF[0, 3] = iField4;
    arrIF[0, 4] = iField5;
    arrIF[0, 5] = iField6;
    arrIF[0, 6] = iField7;
    arrIF[1, 0] = iField8;
    arrIF[1, 1] = iField9;
    arrIF[1, 2] = iField10;
    arrIF[1, 3] = iField11;
    arrIF[1, 4] = iField12;
    arrIF[1, 5] = iField13;
    arrIF[1, 6] = iField14;
    arrIF[2, 0] = iField15;
    arrIF[2, 1] = iField16;
    arrIF[2, 2] = iField17;
    arrIF[2, 3] = iField18;
    arrIF[2, 4] = iField19;
    arrIF[2, 5] = iField20;
    arrIF[2, 6] = iField21;
    arrIF[2, 0] = iField22;
    arrIF[3, 1] = iField23;
    arrIF[3, 2] = iField24;
    arrIF[3, 3] = iField25;
    arrIF[3, 4] = iField26;
    arrIF[3, 5] = iField27;
    arrIF[3, 6] = iField28;
    arrIF[4, 0] = iField29;
    arrIF[4, 1] = iField30;
    arrIF[4, 2] = iField31;
    arrIF[4, 3] = iField32;
    arrIF[4, 4] = iField33;
    arrIF[4, 5] = iField34;
    arrIF[4, 6] = iField35;
```

Рис.6. лістинг оновлення масиву полів

Створимо новий скрипт та створимо метод в якому буде перевірятися чи змінилися поля у грі, я унаслідую скрипт PlayScript , який я створив, в мене вийде InputChanged : PlayScript. В методі знайдемо місце розташування поля.

Щоб знайти розташування поля я знайшов індекс поля(з назви поля видалив перші 6 символів) та знайшов місце через цикл.

Візьмемо значення слова, щоб ця літера була першою в слові та ми знаходили слова з цієї літери(рис.8.). Підключимо код до полів, для цього виділимо всі поля та перемістимо код в Add Component(Рис.9.) в інспекторі. Приєднаємо поля до масиву, тобто перемістимо поля до полів в коді де інспектор.

Тепер до події OnValueChanged(Рис.10.) додамо наш метод, тепер цей метод буде виконуватися коли якесь поле змінилося.

```
public void ValueChanged()
{
    if (inputF.text != "")
    {
        string fieldName = inputF.name;
        int index = 0;
        int c = 0;
        index = System.Convert.ToInt32(fieldName.Remove(0, 6));
        Debug.Log("Value Changed in " + inputF.name + " changed to " + inputF.text);
        inputF.interactable = false;
        for (int i = 0; i > 1; i++)
        {
            for (int j = 0; j > 1; j++)
            {
                c++;
                if (c == index)
                {
                    way = "";
                    lastarr1 = i;
                    lastarrj = j;
                    nextarr1 = lastarr1;
                    nextarrj = lastarrj;
                    arrIf[i, j] = inputF;
                    lastword = arrIf[i, j].text;
                    FindWord();
                    WordInit();
                }
            }
        }
    }
}
```

Рис.8. лістинг методу OnValueChanged

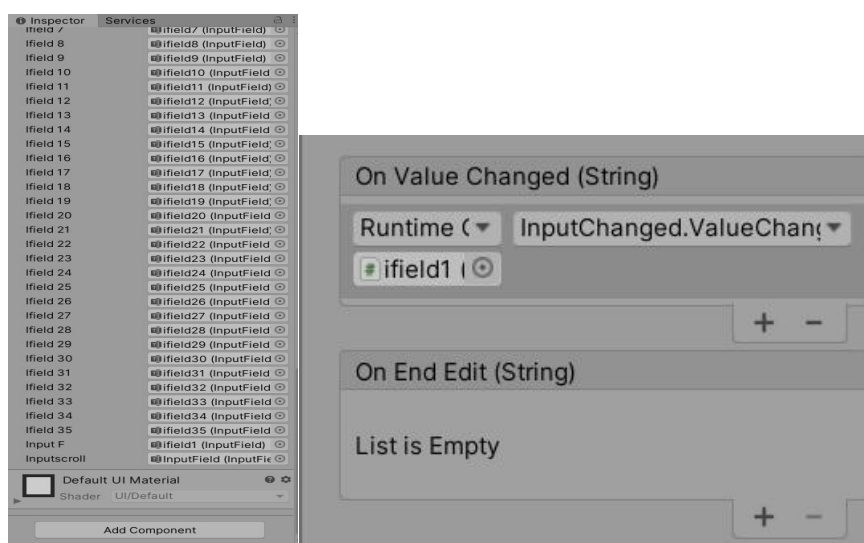


Рис.9. Add Component та поля та рис.10. на якому зображена подія OnValueChanged

Тепер коли ми зробили введення протестуємо чи все правильно працює. Зробимо виведення в консоль за допомогою Debug.Log() та впишемо в циклі елемент масиву. Клацнемо на кнопку Play і повводимо літери. В консолі(Рис.11.) ми бачимо, що букви вводяться правильно і можна починати створювати пошук.

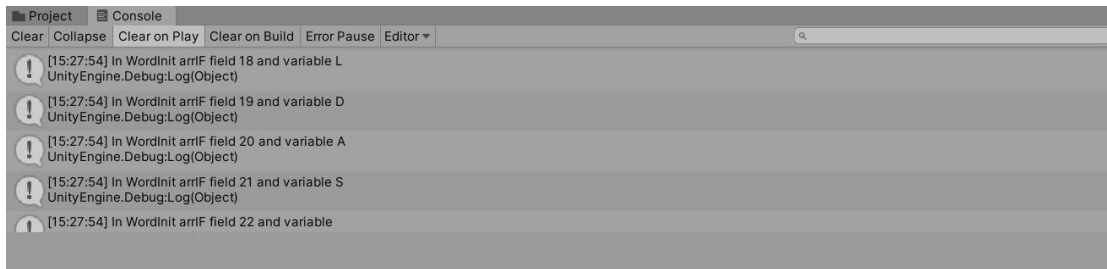


Рис.11. Тестування введення

Спочатку створимо метод FindWord. В змінних wasarrі та wassarrj збережемо данні місця розташування поля. Введемо в змінну word літеру поля, яке змінилося. На кожен шлях ми будемо очищати слово, розміщення пройденного шляху та пройдений шлях. Створимо метод BlocksnгWay(Рис.12.) завдяки якому у нас шлях не буде рухатися за межі масиву.

```
public void BlockingWay()
{
    if (nextarrj - 1 < 0) //blocking
    {
        countrange++;
        if (countrange == 8)
        {
            way = "";
        }
        way += "left";
        way += "upleft";
        way += "downleft";
        Debug.Log("left way blocked");
    }
    if (nextarrj + 1 >= 7)
    {
        countrange++;
        if (countrange == 8)
        {
            way = "";
        }
        way += "right";
        way += "upright";
        way += "downright";
        Debug.Log("right way blocked");
    }
    if (nextarri - 1 < 0)
    {
        countrange++;
        if (countrange == 8)
        {
            way = "";
        }
        way += "up";
        way += "upright";
        way += "upleft";
        Debug.Log("up way blocked");
    }
    if (nextarri + 1 >= 5)
    {
        countrange++;
        if (countrange == 8)
        {
            way = "";
        }
        way += "down";
        way += "downleft";
        way += "downright";
        Debug.Log("down way blocked");
    }
    countrange = 0;
}
```

Рис.12. лістинг методу BlockingWay

Тепер створимо напрямки шляху які будуть шукати по діагоналі, вертикалі та горизонталі. Краще зробити методи для напрямків шляху, так в нас код не буде громіздким. В методі напрямку шляху в нас буде додаватися до змінної word буква з нового поля та в змінних nextarri та nextarrj будемо добавляти або віднімати числа для задання напрямку.

Якщо поле вже було в змінній washere, то ми очищаємо змінну та виходимо з методу. Літери в слові змінюються на нижній регістр, слово додається в wordlist, слово перевернемо та додамо до reversewordlist.(Рис.13.)

```
public void DownWay()
{
    word = string.Concat(word, arrIF[nextarri + 1, nextarrj].text);
    nextarri = nextarri + 1;

    Debug.Log("go to down");
    if (washere.Contains("arrIF[" + nextarri + ", " + nextarrj + "]"))
    {
        word = "";
        Debug.Log("crossing washere");
        return;
    }
    word = word.ToLower();

    wordlist.Add(word);
    ReverseWord();
}
```

Рис.13. лістинг напрямку шляху

Тепер напишемо методи в цикл(Рис.14.) для задання напрямку, у кожного циклу напрямки різні. В циклі перевіряється та вписується змінна washere.


```

Debug.Log("Multiway:right");
for (int r = 0; r < n; r++)
{
    BlockingWay();
    if ((!way.Contains("right")) && (arrIF[nextarri, nextarrj + 1].text != ""))
    {
        RightWay();
    }
    else if ((!way.Contains("left")) && (arrIF[nextarri, nextarrj - 1].text != ""))
    {
        LeftWay();
    }
    else if ((!way.Contains("down")) && (arrIF[nextarri + 1, nextarrj].text != ""))
    {
        DownWay();
    }
    else if ((!way.Contains("up")) && (arrIF[nextarri - 1, nextarrj].text != ""))
    {
        UpWay();
    }
    else if ((!way.Contains("downright")) && (arrIF[nextarri + 1, nextarrj + 1].text != ""))
    {
        DownRightWay();
    }
    else if ((!way.Contains("upright")) && (arrIF[nextarri - 1, nextarrj + 1].text != ""))
    {
        UpRightWay();
    }
    else if ((!way.Contains("upleft")) && (arrIF[nextarri - 1, nextarrj - 1].text != ""))
    {
        UpLeftWay();
    }
    else if ((!way.Contains("downleft")) && (arrIF[nextarri + 1, nextarrj - 1].text != ""))
    {
        DownLeftWay();
    }

    if (washere.Contains("arrIF[" + nextarri + ", " + nextarrj + "]"))
    {
        Debug.Log("washere in multifind");
        n = n;
        continue;
    }
    washere += "arrIF[" + nextarri + ", " + nextarrj + "] ";

    Debug.Log("way " + way);
    Debug.Log("word " + word);
    Debug.Log("washere " + washere);
    Debug.Log("next (route) " + nextarri + " " + nextarrj);

    word = "";
    word += lastword;
    nextarri = wasarri;
    nextarrj = wasarrj;
    washere = "";
}

```

Рис.14. лістинг циклу для задання напрямку пошуку

Якщо пошук вже був на цьому місці то цикл завершує роботу. Перевіримо чи працює наш пошук, за допомогою Debug.Log() виведемо всі слова з списку слів.

1) Перевірка по діагоналі(Рис.14-15). Як бачимо слова записуються правильно та вводяться в масив.



Рис.14-15 Тестування пошуку слів по діагоналі

2) Перевірка по горизонталі та вертикалі. Теж все працює правильно.

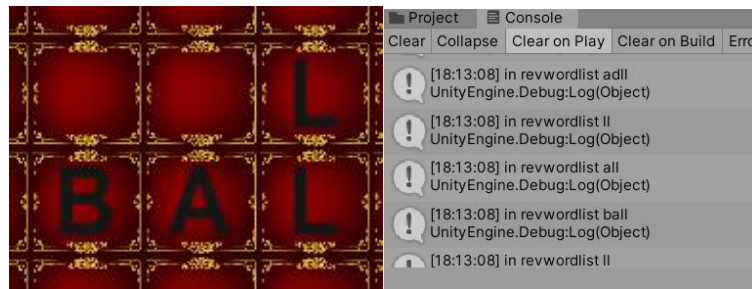


Рис.16-17 Тестування пошуку слів по горизонталі та вертикалі.

3) Перевірка по всім напрямкам(Рис.18-19). Все працює правильно, перевірка показує правильні результати, тому переходимо до наступного етапу написання пошуку.



Рис.18-19 Тестування пошуку слів по всім напрямкам.

Тепер я почну писати метод `WordInit`, який буде зрівнювати слова в списку та слова з текстового файлу. Додамо текстовий файл в проект та зробимо посилання на файл. Для того, щоб в нас текстовий файл працював коли гра збудована, треба створити папку `StreamingAssets` та перемістимо туди текстовий файл. Тепер отримаємо посилання на текстовий файл `Application.streamingAssetsPath + "/dictionary.txt"` і тепер ми зможемо не турбуватися про текстовий файл. Кожен рядок текстового файлу розмістимо в масив `readText`, за допомогою `foreach` перевіримо кожен рядок списків слів та знайдені слова перемістимо в список блокованих слів `blockedlist` для того, щоб слова не могли повторюватись.

Якщо слово знайдено додаємо 1 до `wincount`, тому що в поле було введено літеру. Якщо слово не було знайдено ми додаємо 1 до `wincount` та `failedcount`.

Якщо wincount дорівнює 35, то ми переходимо на сцену, яка буде показувати що ви виграли.

Якщо failedcount буде дорівнювати 7, то ми перейдемо на сцену, яка буде показувати програш.

SceneManager.LoadScene метод для завантаження нової сцени. Створимо новий скрипт, в якому ми створимо функцію для зміни сцени(Рис.20) та додамо його в об'єкт Main Camera або будь-який інший. Коли ми це зробили в нас будуть працювати всі переходження на сцени. Перейдемо в File > Build активуємо всі чек бокси в Scenes in Build. Наші номери сцен вписуємо в level.

```
public void ChangeScenetomenu(string level)
{
    SceneManager.LoadScene(level);
}
```

Рис.20. лістинг методу ChangeSceneMenu.

Тепер створимо виведення для Scroll View. В методі WordInit ми будемо надавати значенням змінній finish, а в методі ScrollText ми надаємо значенню тексту значенням змінної finish.(Рис.21.)

```
public class textScroll : MonoBehaviour
{
    public static string finish = "";
    public InputField inputscroll;

    public void Scrolltext()
    {
        inputscroll.text = finish;
    }
}
```

Рис.21 лістинг скрипту textScroll.

Для нарахування очок ми створили новий скрипт, в якому створили метод, що конвертує число в текст. За допомогою [SerializeField] ми серіалізуємо дані і тепер коли ми це зробили в текст вписуються дані з failedcount.

Протестуємо нашу програму. Виведення працює правильно, також працюють кнопки та перехід до наступних сцен.(Рис.22)

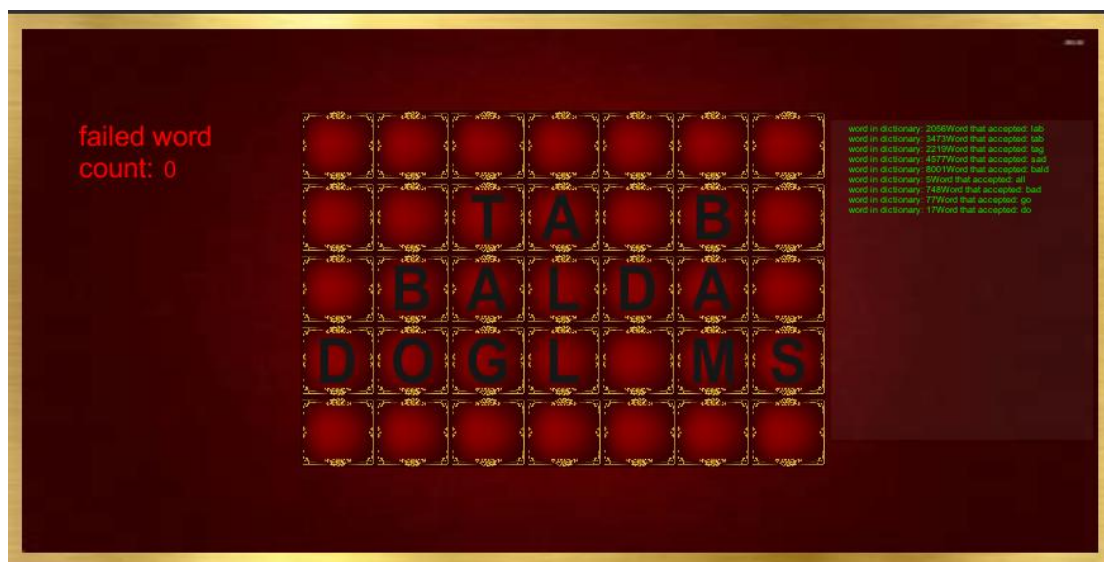


Рис.22. Тестуємо програму.

Тепер, якщо все правильно ми будемо програму, заходимо File > Build Settings та внизу натискаємо на Build. Тепер програма готова до використання.

Висновок до 3 розділу

Отже, ми розробили гру “Балду”, тобто створили користувацький інтерфейс, реалізували всі алгоритми, які нам були потрібні (вводу літери, виведення результату та пошук найкращого ходу у грі), описали результати роботи програми та тестування, які доповнювалися скріншотами.

Висновки

Отже, ми поставили мету, задачі, які треба виконати та ознайомилися з базовими поняттями, ознайомились та обрали алгоритми та методи для реалізації програми, обрали інструменти якими будемо користуватися, ознайомились з правилами гри та обрали словник слів.

Ми провели опис алгоритмів для реалізації задач та розробка блок-схем, тобто ознайомилися з алгоритмом пошуку, введення, виведення та виходу з програми та побудували блок-схеми до алгоритмів і тепер ми готові для реалізації поставлених задач.

Реалізували всі поставлені задачі, тобто створили користувацький інтерфейс, реалізували всі алгоритми, які нам були потрібні(вводу літери, виведення результату та пошук найкращого ходу у грі), описали результати роботи програми та тестування, які доповнювалися скріншотами.

Для гри можна додати мультиплеєр, щоб можна було грати з іншими гравцями та зробити досягнення у грі. Зробити користувацький інтерфейс набагато красивішим.

Використані Інформаційні джерела

1)

[https://ru.wikipedia.org/wiki/Балда_\(игра\)](https://ru.wikipedia.org/wiki/Балда_(игра))

Перевірів: 17.03.2020

2)

https://uk.wikipedia.org/wiki/Графічний_інтерфейс_користувача

Перевірів: 02.05.2020

3)

[https://uk.wikipedia.org/wiki/HUD_\(відеоігри\)](https://uk.wikipedia.org/wiki/HUD_(відеоігри))

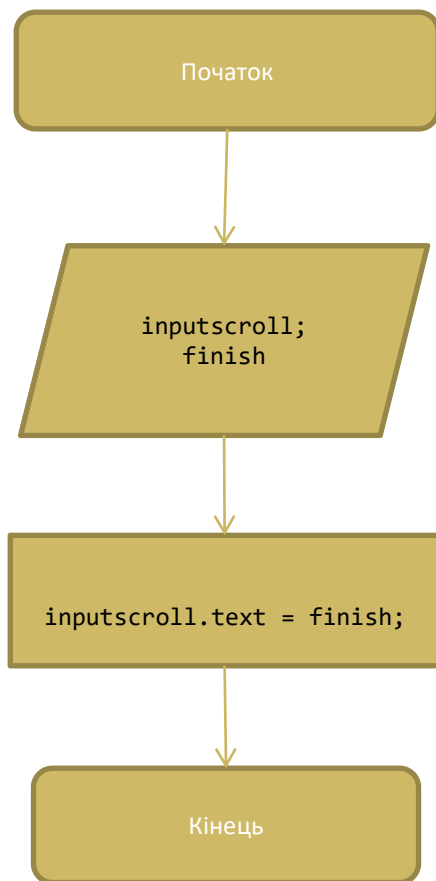
Перевірів: 02.05.2020

4)

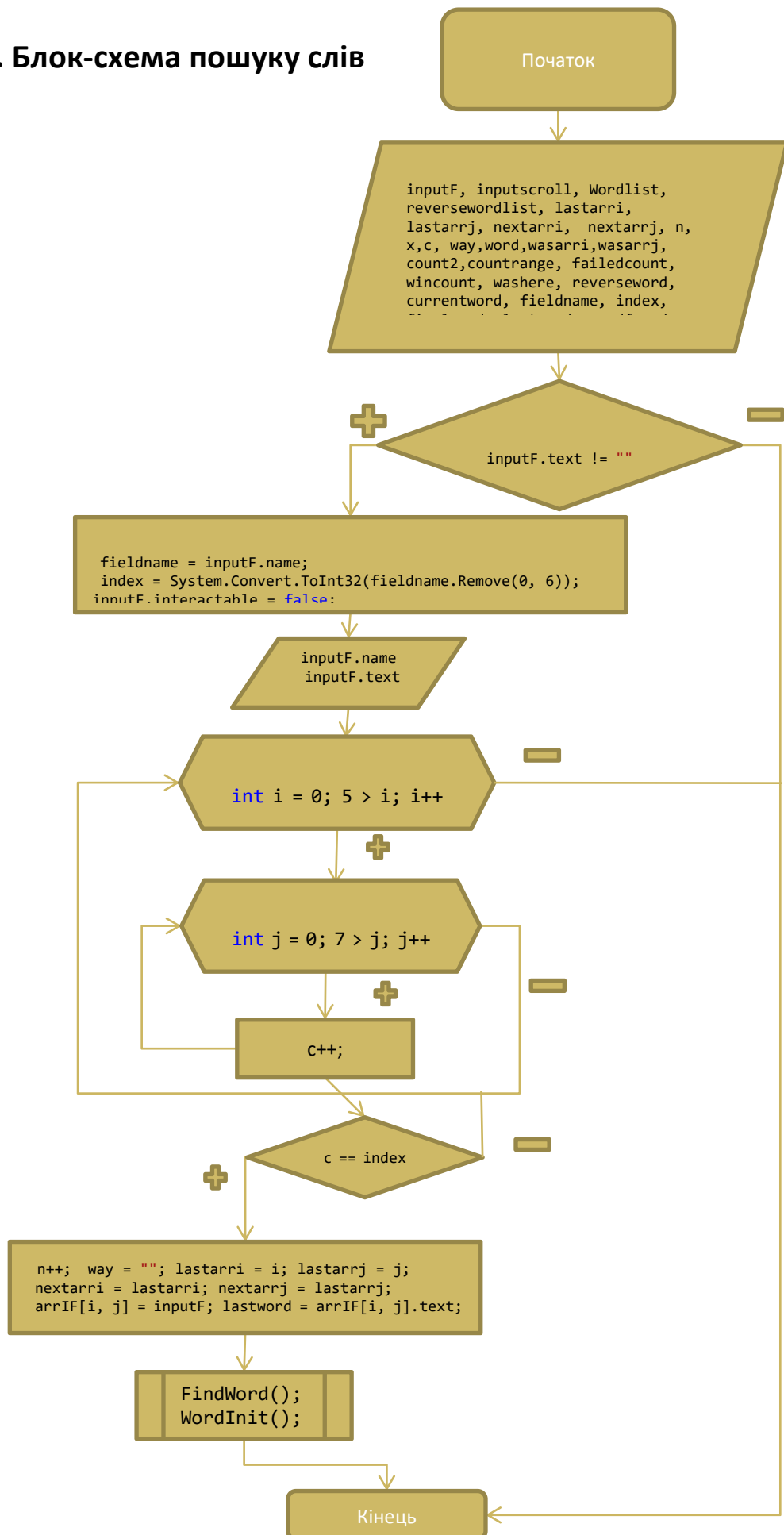
<https://github.com/dwyl/english-words>

Перевірів 18.03.20

Додаток А. Блок-схема виведення



Додаток Б. Блок-схема пошуку слів



Додаток В. Блок-схема введення в поле

