ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ БІЗНЕСУ ТА СУЧАСНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО БАКАЛАВРСЬКОЇ РОБОТИ

на тему
ТРЕНАЖЕР З ТЕМИ «АЛГОРИТМИ СИНТАКСИЧНОГО АНАЛІЗУ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ»

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Соболєв Денис Андрійович
_____ «___»_____2021р.
(підпис)
Науковий керівник к.ф.-м.н., доц. Черненко Оксана Олексіївна
_____ «___»_____2021р.
(підпис)

ПОЛТАВА 2021р.

ЗМІСТ

ВСТУП 4
1 Постановка задачі 6
2 Інформаційний огляд 7
2.1 Плюси та мінуси розглянутих тренажерів 7
2.2 Актуальність роботи 12
3 Теоретична частина 13
3.1 Загальні відомості 13
3.2 Алгоритм роботи тренажера 20
3.3 Блок-схема програми-тренажера 27
4 Практична частина 28
4.1 Опис програмної реалізації 28
4.2 Перевірка валідності програмного забезпечення 33
ВИСНОВКИ 40

ВСТУП

З часів розробки першого комп'ютера в далекому 1941 процес модернізації, та полегшення роботи людини не зупинявся. І зараз, у сучасному світі, однією з великих задач людства являється автоматизація такого процесу як навчання. Саме тому було запроваджено ботів-вчителів та програми-тренажери для навчання певним дисциплінам чи темам.

Такі програми-тренажери користуються великим попитом серед людей з суворим графіком, що не мають часу на повноцінне навчання. Таке програмне забезпечення дозволяє у будь-який час та навіть без доступу в мережу інтернет отримувати доступ до навчальних матеріалів з будь-якої частини світу.

Мета роботи – створення програмного забезпечення у вигляді тренажеру з теми «Алгоритми синтаксичного аналізу» для систем дистанційного навчання.

Об'єкт роботи – процес дистанційного навчання студентів.

Предмет роботи – програма-тренажер з теми «Алгоритми синтаксичного аналізу».

Методи роботи – програмне забезпечення розроблено за допомогою MS Visual Studio та платформи Unity 2018.

Структура пояснювальної записки до бакалаврської роботи:

- титульний аркуш;
- завдання до бакалаврської роботи;
- реферат, що містить предмет, мету, методи, анотацію результатів ключові слова, словосполучення;
- зміст;
- вступ;
- інформаційний огляд;
- теоретична частина;
- практична частина;

- висновки;

- список використаних джерел;

Обсяг пояснювальної записки: 42 стор., в т.ч. основна частина 40 стор., джерел -10.

1 Постановка задачі

Метою та головним завданням бакалаврської роботи є створення програмного забезпечення у вигляді тренажеру з теми «Алгоритми синтаксичного аналізу» для систем дистанційного навчання.

Дане програмне забезпечення створено у MS Visual Studio, платформи Unity 2018 та мови програмування C#.

Головним завданням даної бакалаврської роботи є:

1. Вибір методів розробки;
2. Складання пояснювальної записки;
3. Складання алгоритму роботи програмного забезпечення;
4. Створення блок-схем роботи алгоритму;
5. Програмна реалізація тренажеру;
6. Перевірка та тестування програми;

Основними вимогами до програмного забезпечення є:

1. Наявність у тренажері як довідкового матеріалу так і практичних завданнь;
2. Можливість виходу з програми при обранні повноекранного показу;
3. Перевірку введеної відповіді з відповідним повідрмленням про правильність;
4. Інтерфейс програми не повинен відволікати увагу користувача під час роботи.

2 Інформаційний огляд

Під час пошуку та аналізу схожих за тематикою тренажерів було знайдено два тренажери, а саме: Величко, Артур Олександрович - тренажер з теми «Способи задання мов» [1] та Белінський, Олексій Борисович - тренажер з теми «Побудова предикативних синтаксичних аналізаторів» з дистанційного навчального курсу «Теорія програмування» [2].

2.1 Плюси та мінуси розглянутих тренажерів

При запуску тренажеру «Способи задання мов» автор Величко Артур Олександрович відкривається головне меню тренажеру на якому є можливість обрати теоретичну частину або практичне тестування.

Існує перевірка введеної відповіді при роботі з практичним тестуванням, реалізовано вікно оцінювання та кнопка повернення в головне меню.

При виборі неправильної відповіді виводиться повідомлення «Відповідь не вірна, оберіть інший варіант!!!», якщо відповідь вірна відбувається перехід на інший крок.

При запуску тренажеру «Побудова предикативних синтаксичних аналізаторів» автор Белінський Олексій Борисович відкривається вкладка «Головна» на якій є можливість перейти до довідкової інформації або одразу до тестів (див. рисунок 2.1)

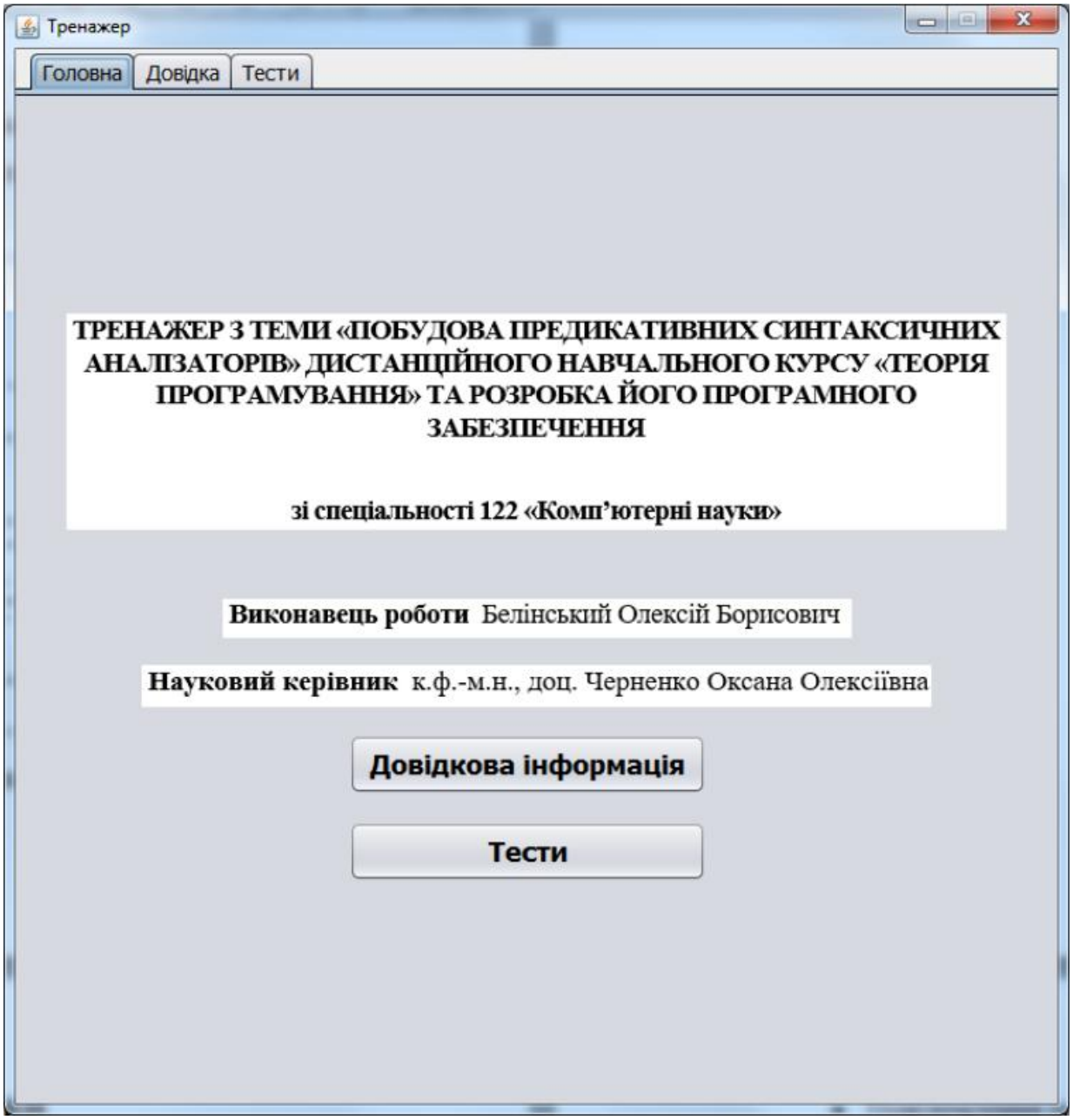


Рисунок 2.1 – Перша вкладка тренажеру «Головна»

При виборі довідкової інформації користувач через натиснення на кнопку на головному екрані, або при виборі вкладки «Довідка» вгорі тренажеру користувач отримує довідкову інформацію (див. Рисунок 2.2)

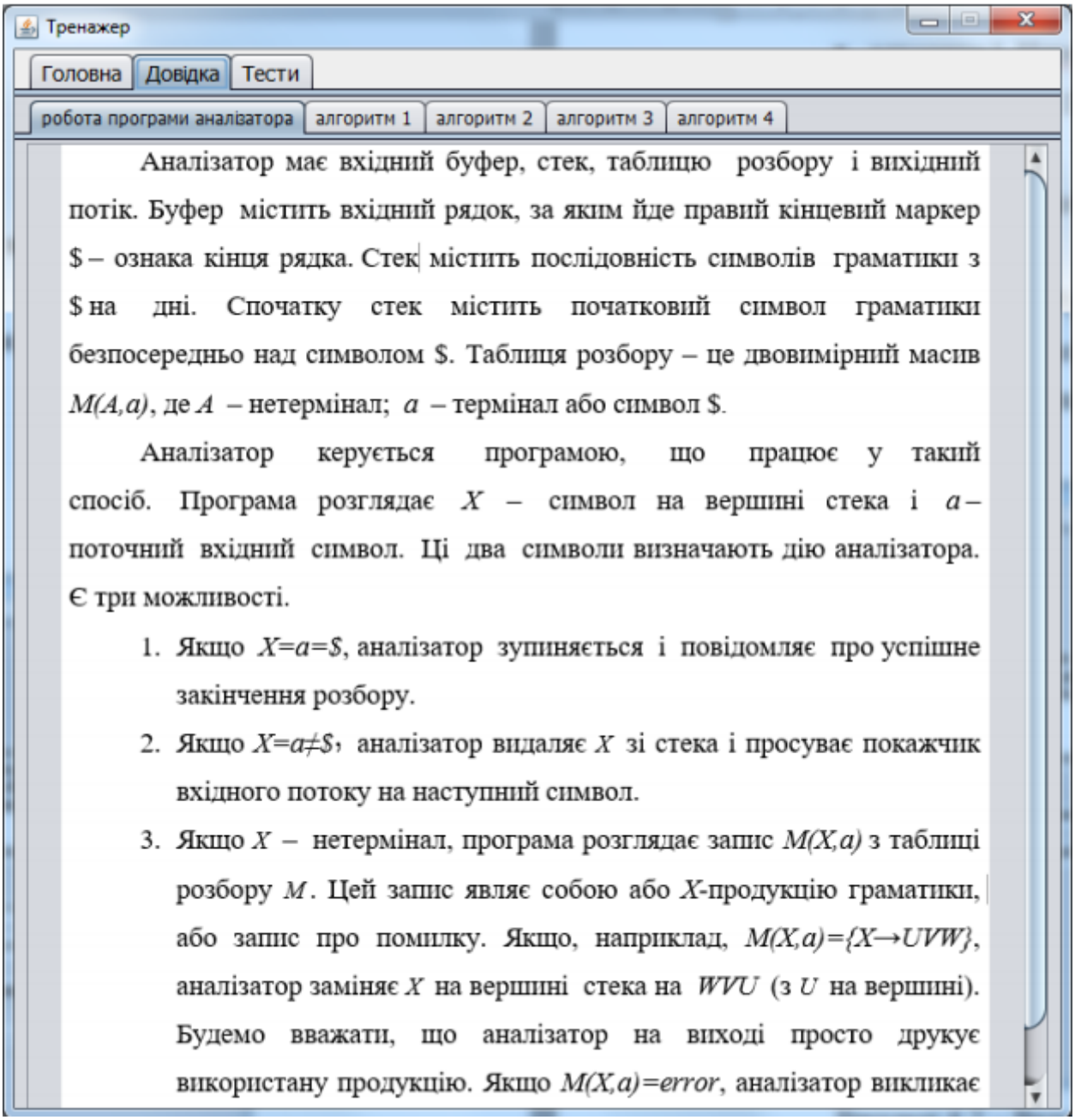


Рисунок 2.2 –Вкладка тренажеру «Довідка»

На вкладці «Довідка» є можливість перейти до кожної теми довідкового матеріалу з будь-якої частини роботи з тренажером.

При виборі вкладки «Тести» корисутвач отримує доступ до тестового матеріалу по кожному кроку роботи для побудови синтаксичного аналізатора (див. рисунок 2.3)

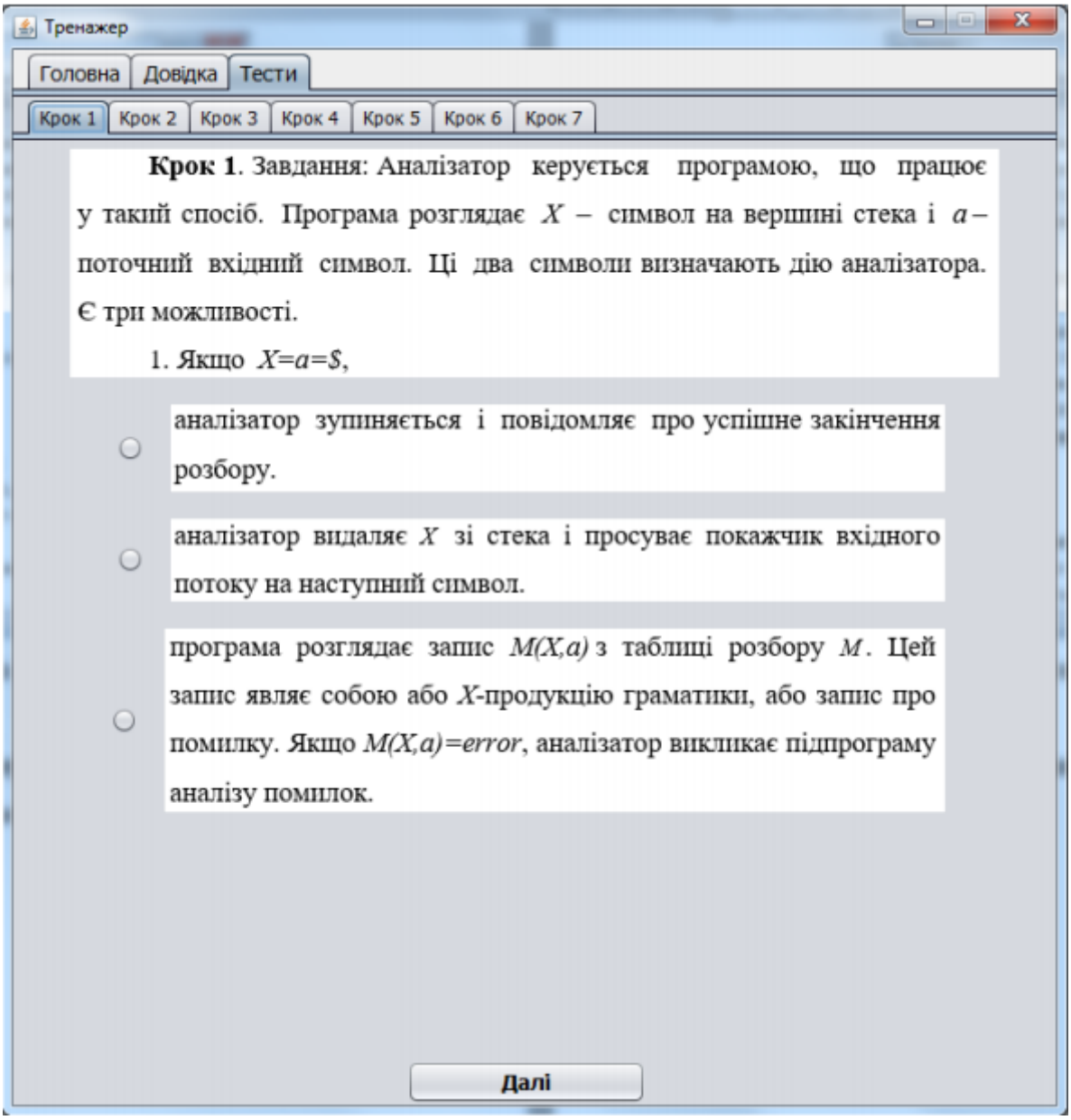


Рисунок 2.2 –Вкладка тренажеру «Довідка»

Після вибору правильного варіанту відповіді на останній тест тренажеру виводиться повідомлення про успішне завершення (див. Рисунок 2.3).

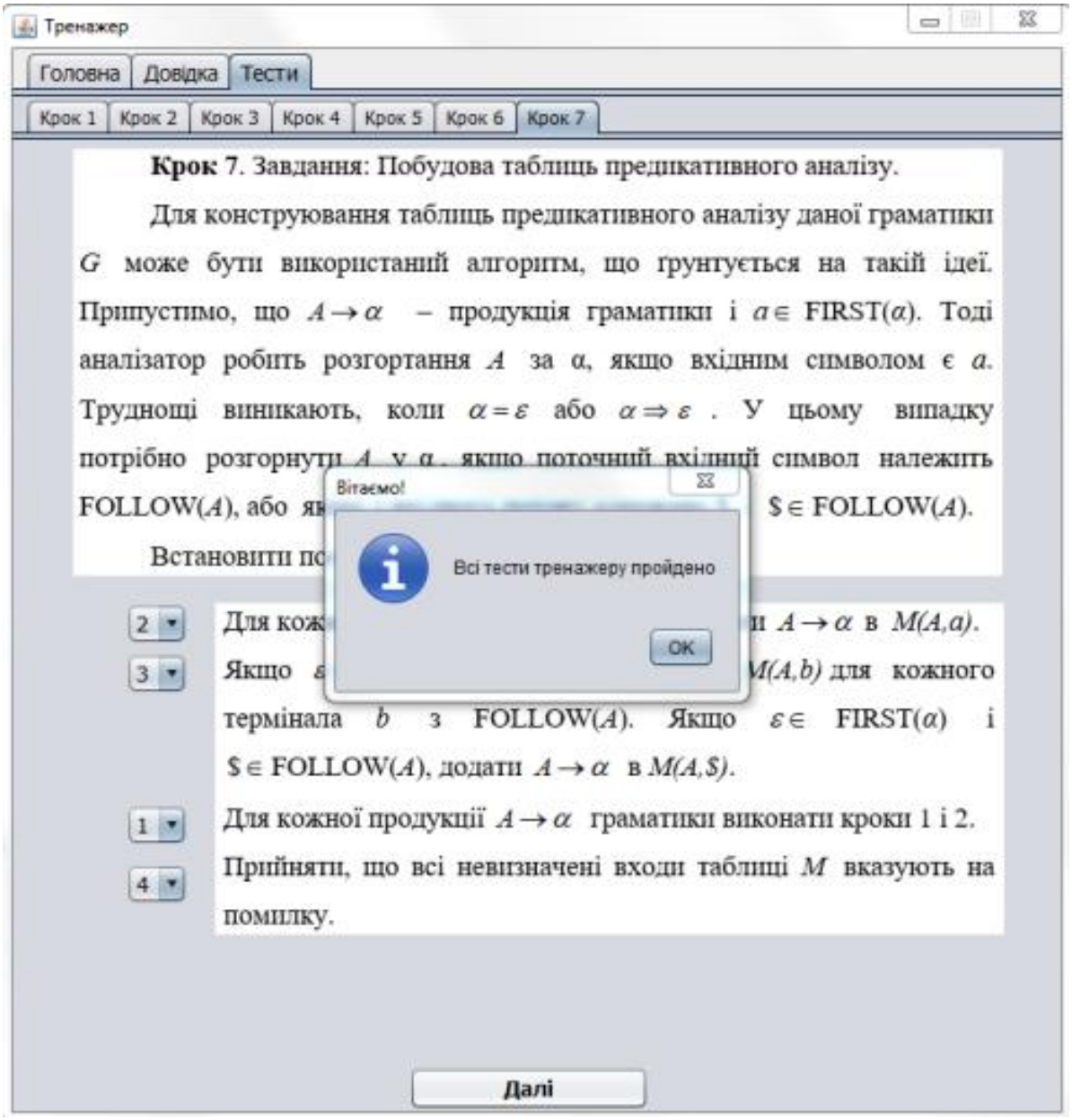


Рисунок 2.3 – Завершення роботи з тренажером

Мінусом даного тренажеру є відсутність повтору роботи без закриття тренажеру та маленьке робоче вікно.

2.2 Актуальність роботи

За неможливістю повноційного стаціонарного навчання, програми-тренажери одні з найкращих методів самостійного навчання у будь-який зручний для користувача час.

Тренажерів або схожих навчальних програм за темою «Алгоритми синтаксичного аналізу» знайдено не було, вважається потрібним та актуальним розробка програмного забезпечення за даною темою.

3 Теоретична частина

3.1 Загальні відомості

Синтаксичний аналіз – це процес зіставлення лінійної послідовності слів (лексем, токенів) мови з формальною граматикою, результатом якого є дерево розбору. Використовується спільно з лексичним аналізом. У процесі синтаксичного аналізу вхідний текст перетворюється в структуру даних (дерево) і добре підходить для подальшої обробки. Розрізняють два типи алгоритмів синтаксичного аналізу: **спадний** – будує дерево розбору від кореня, сворюючи вузли в порядку обходу; **висхідний** – будує дерево розбору для вхідного рядка, починаючи з листя і до кореня. [3]

Синтаксичний аналізатор (англ. parser) — це програма або частина програми, яка виконує синтаксичний аналіз.

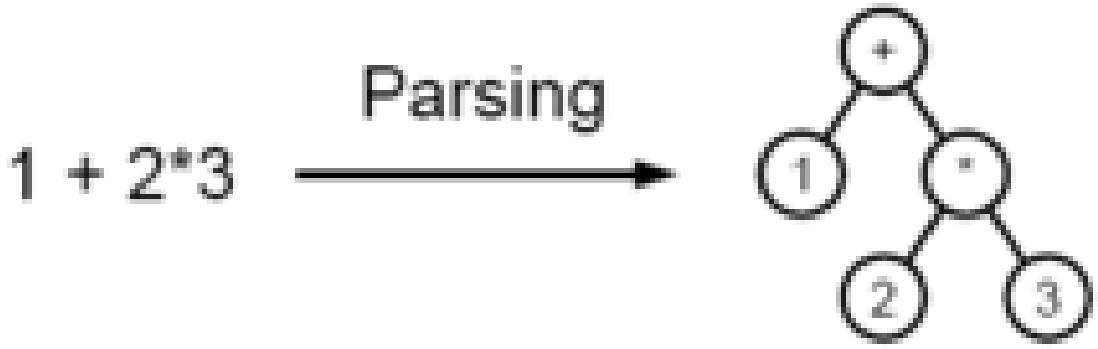


Рисунок 3.1 - Приклад розбору виразу в дерево

Під час синтаксичного аналізу текст оформлюється у структуру даних, зазвичай — в дерево, яке відповідає синтаксичній структурі вхідної послідовності і добре підходить для подальшої обробки. Зазвичай синтаксичні аналізатори працюють в два етапи: на першому ідентифікуються осмислені токени (виконується лексичний аналіз), на другому створюється дерево розбору. [4]

Кожна мова програмування має точні правила, які задають синтаксичну структуру коректних програм. Синтаксис конструкцій мови програмування може бути описаний за допомогою контекстно-вільних граматик або форм

Бекуса-Наура, Граматики забезпечують наступні значні переваги розробникам компіляторів та інтерпретаторів мов програмування: граматика дає точну і просту для розуміння синтаксичну специфікацію мови програмування; для деяких класів граматик можливо автоматично побудувати ефективний синтаксичний аналізатор, який визначає синтаксичну структуру вихідної програми. Додатковою перевагою автоматичного створення аналізатора є можливість виявлення синтаксичних неоднозначностей та інших складностей, які інакше могли б залишитися непоміченими на початкових фазах створення мови та її транслятора; правильно побудована граматика структурує мову програмування, що полегшує трансляцію вихідної програми в коректний об'єктний код і виявлення помилок; граматика дозволяє мови ітеративно еволюціонувати, збагачуючись новими конструкціями для розв’язання нових задач. Додавання цих нових конструкцій у мові виявляється більш простим завданням, якщо існуюча реалізація мови заснована на її граматичному описі. [5]

Предиктивний синтаксичний аналізатор – це синтаксичний аналізатор, що працює методом рекурсивного спуску без повернень (відкотів). Це можливо, якщо з граматики видалена ліва рекурсія і вона лівофакторизована.

У багатьох випадках акуратне розроблення граматики, видалення з неї лівої рекурсії та її ліва факторизація дозволяють одержати граматику, яка може бути проаналізована синтаксичним аналізатором, що використовує метод рекурсивного спуску і не потребує відкоту (предиктивним аналізатором).

Нерекурсивний предиктивний аналізатор можна побудувати за допомогою явного використання стека замість неявного при рекурсивних викликах. Ключова проблема предиктивного аналізу полягає у визначенні продукції, яку потрібно застосувати до нетермінала. Для пошуку продукції може бути використана таблиця розбору.

Модель предиктивного синтаксичного аналізатора, що керується таблицею, показана на рис. 4.

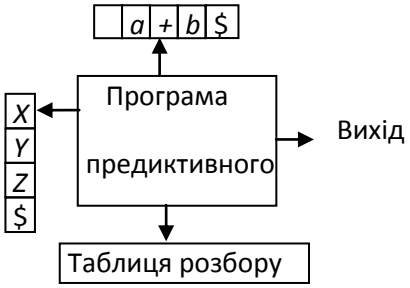


Рисунок 3.2 - Схема предиктивного аналізатора

Аналізатор має вхідний буфер, стек, таблицю розбору і вихідний потік. Буфер містить вхідний рядок, за яким йде правий кінцевий маркер $– ознака кінця рядка. Стек містить послідовність символів граматики з $ на дні. Спочатку стек містить початковий символ граматики безпосередньо над символом $. Таблиця розбору – це двовимірний масив $M[A,a]$, де A – не термінал; a – термінал або символ $.

Аналізатор керується програмою, що працює у такий спосіб. Програма розглядає X – символ на вершині стека і a – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три можливості.

1 Якщо $X = a = \$$, аналізатор зупиняється і повідомляє про успішне закінчення розбору.

2 Якщо $X = a \neq \$$, аналізатор видаляє X зі стека і просуває покажчик вхідного потоку на наступний символ.

3 Якщо X – нетермінал, програма розглядає запис $M[X,a]$ з таблиці розбору M. Цей запис являє собою або *X*-продукцію граматики, або запис про помилку. Якщо, наприклад, $M[X,a] = \{X \to UVW\}$, аналізатор заміняє X на вершині стека на WVU (з U на вершині). Будемо вважати, що аналізатор на виході просто друкує використану продукцію. Якщо $M[X,a] = error$, аналізатор викликає підпрограму аналізу помилок. [3]

Найбільш загальний метод ПЗ-аналізу без відкоту - $\boldsymbol{LR(k)}$ **- аналіз,** пристрій, що його реалізує називається $\boldsymbol{LR(k)}$ **-аналізатор.** У назві **с**имвол L означає, що розбір здійснюється зліва направо, R – що будується праве породження у оберненому порядку, k – число вхідних символів, які можуть бути переглянуті для прийняття рішення про спосіб проведення розбору. Коли k опущено, мають на увазі 1.

Переваги LR-аналізу:

1. LR -аналізатори можуть бути побудовані для розпізнавання практично всіх конструкцій мов програмування.

2. Метод LR -аналізу – найбільш загальний метод ПЗ-аналізу без відкоту, який, крім того, може бути реалізований досить ефективно.

Найбільш відомі три методи побудови таблиць LR -аналізу для граматик. Перший метод, *простий* LR *(*$simple\ \ LR$*,* SLR*)*, є найбільш легким в реалізації, але найменш потужним з них. Він не може побудувати таблицю для розбору деяких граматик, які успішно обробляються іншими методами. Другий метод, *канонічний* LR*,* – найбільш потужний, але він потребує найбільших ресурсів. Третій метод – *метод з попереднім переглядом (*$lookaheadLR$*,* $LALR$*)* за потужністю і споживаними ресурсами займає проміжне положення. Метод $LALR$ працює з більшістю граматик і може бути ефективно реалізованим.

Схематично структура LR -аналізатора зображена на рис. 9.1.

LR -аналізатор складається з входу, виходу, стека, керуючої програми і таблиці синтаксичного аналізу, що складається з двох частин – дії $(action)$ і переходу $(goto)$. Керуюча програма одна й та сама для всіх аналізаторів, різні аналізатори розрізняються таблицями аналізу. Програма синтаксичного аналізу читає символи з вхідного буфера по одному за крок.

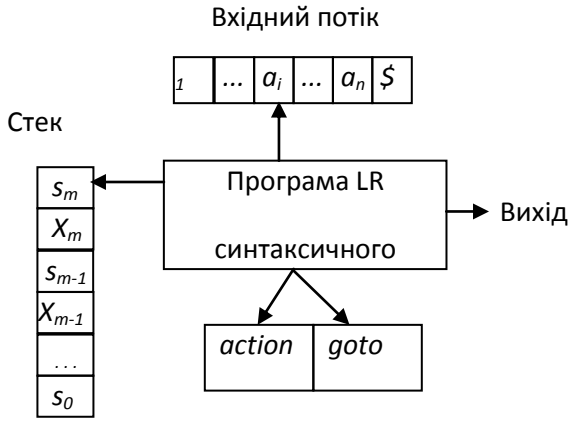


Рисунок 3.3 - Структура LR -аналізатора

У процесі аналізу використовується стек, у якому зберігаються рядки вигляду $s_0X_1s_1X_2s_2...X_ms_m$ (s_m знаходиться на вершині стека). Кожен X_i є символом граматики (термінальним або нетермінальним), а s_i – символ, який називається станом. Кожен символ стану виражає інформацію, що утримується в стеку нижче його, а комбінація символа стану на вершині стека і поточного вхідного символа використовується для індексації таблиці аналізу і визначає рішення про перенесення або згортання.

Конфігурація LR аналізатора – це пара, перший компонент якої – вміст стека, а другий – непереглянута частина вхідного потоку ($s_0X_1s_1X_2s_2...X_ms_m$, $a_ia_{i+1}...a_n\$$). Керуюча програма синтаксичного аналізатора функціонує таким чином. Вона визначає s_m, поточний стан на вершині стека і поточний вхідний символ a_i. Потім програма звертається до $action[s_m, a_i]$, елемента таблиці дій синтаксичного аналізу, який може мати одне з чотирьох значень:

1) перенесення s, де s – стан;
2) згортання у відповідності з продукцією $A \to \beta$;
3) допуск;
4) помилку.

Функція $goto$ одержує як аргументи стан і символ граматики і повертає новий стан.

Конфігурації, що виходять після кожного з чотирьох типів дій, такі.

1 Якщо $action[s_m, a_i] =$ "перенесення s ", то синтаксичний аналізатор виконує перенесення, переходячи у конфігурацію ($s_0 X_1 s_1 X_2 s_2 ... X_m s_m a_i s, a_{i+1} ... a_n \$)$. Синтаксичний аналізатор переносить у стек поточний вхідний символ a_i і черговий стан s, який визначається значенням $action[s_m, a_i]$; поточним вхідним станом стає a_{i+1}.

2 Якщо $action[s_m, a_i] =$ "згортання $A \to \beta$ ", синтаксичний аналізатор виконує згортання, переходячи у конфігурацію $(s_0 X_1 s_1 X_2 s_2 ... X_{m-r} s_{m-r} A s,\ a_{i+1} ... a_n \$)$, де $s = goto[s_{m-r}, A]$, а r – довжина β, правої частини продукції. Тут синтаксичний аналізатор спочатку видаляє із стека $2r$ символів (r символів стану і r символів граматики), так що на вершині виявляється стан s_{m-r}. Потім аналізатор поміщає в стек A (ліву частину продукції) і s – вміст елемента таблиці $goto[s_{m-r}, A]$. Поточний вхідний символ не змінюється. Для LR -аналізаторів послідовність символів $X_{m-r+1} ... X_m$, що видаляються із стека, завжди відповідає β – правій частині продукції згортання.

3 Якщо $action[s_m, a_i] =$ "допуск", то синтаксичний аналіз завершено.

4 Якщо $action[s_m, a_i] =$ = "помилка", аналізатор знайшов помилку і виконуються дії з діагностицки і відновлення.

Нижче наведено алгоритм LR -аналізу. Усі LR -аналізатори ведуть себе однаково. Різниця між ними полягає у різному змісті таблиць дій і переходів.

Граматики, для яких можна побудувати таблицю LR -розбору, називаються ***LR* -граматиками.** Існують контекстовільні граматики, що не є LR -граматиками, однак практично для опису мов типових конструкцій мов їх можна уникнути.

Щоб граматика була LR -граматикою, аналізатор, що працює зліва направо по типу перенесення-згортання, повинен уміти розпізнавати основи на вершині стека. LR -аналізатор не повинен сканувати весь стек, щоб розпізнати появу основи на вершині стека. Більше того, символ стану на вершині стека містить всю необхідну інформацію. Але якщо можна розпізнати основу, знаючи тільки символи граматики в стеку, то існує скінченний автомат, що може, читаючи символи граматики в стеку зверху вниз, визначити цю основу. Функцією переходів цього скінченного автомата є таблиця переходів LR -аналізатора. Щоб не переглядати стек на кожному кроці аналізу, на вершині стека завжди зберігається той стан, у якому повинен виявитися цей скінченний автомат після того, як він прочитав символи граматики в стеку від дна до вершини.

Для вибору рішення про перенесення або згортання аналізатор переглядає чергові k вхідних символів. Практичний інтерес становлять випадки $k=0$ і $k=1$. Наприклад, у колонках *action* табл. 9.3 використовується тільки один символ. Граматика, для розбору якої LR -аналізатору потрібен перегляд до k вхідних символів на кожному кроці, називається $\boldsymbol{LR(k)}$**-граматикою.**

Основна різниця між LL - і LR -граматиками полягає у наступному. Щоб граматика була $LR(k)$, необхідно вміти розпізнавати появу правої частини продукції, бачачи усе, що породжено з цієї правої частини, а також k вхідних символів. Ця вимога істотно менш строга, ніж вимога для $LL(k)$-граматики, коли необхідно визначити застосовну продукцію, бачачи тільки перші k символів, виведених з її правої частини. [6]

3.2 Алгоритм роботи тренажера

Під час початку роботи з тренажером користувач бачить перед собою головне вікно тренажеру з інформацією про тему та виконавця роботи та кнопку для початку роботи з тренажером.

Інформація в тренажері розташована таким чином, що довідкова інформація розташовується перед тестовими завданнями з певної теми.

Реалізовано перевірку правильної відповіді під час роботи з тестовими завданнями. У випадку вибору неправильної відповіді користувач отримує повідомлення «Відповідь не вірна!». У іншому випадку - повідомлення «Відповідь вірна!» та доступ до наступного завдання.

Алгоритм роботи з навчальними матеріалами тренажеру можна описати наступними кроками:

Крок 1: Запуск тренажера та ознайомлення з темою роботи;

Крок 2: Перехід до навчальних матеріалів тренажера шляхом натиснення кнопки «Почати роботу»;

Крок 3: Отримання довідкового матеріалу з теми:

«**Синтаксичний аналіз** – це процес зіставлення лінійної послідовності слів (лексем, токенів) мови з формальною граматикою, результатом якого є дерево розбору. Використовується спільно з лексичним аналізом. У процесі синтаксичного аналізу вхідний текст перетворюється в структуру даних (дерево) і добре підходить для подальшої обробки. Розрізняють два типи алгоритмів синтаксичного аналізу: **спадний** – будує дерево розбору від кореня, сворюючи вузли в порядку обходу; **висхідний** – будує дерево розбору для вхідного рядка, починаючи з листя і до кореня.»

Крок 4: Отримання практичного завдання:

«**Приклад 1.**

Розглянемо граматику з продукціями $S \to cAd, A \to ab|a$ та вхідний рядок $w = cad$.

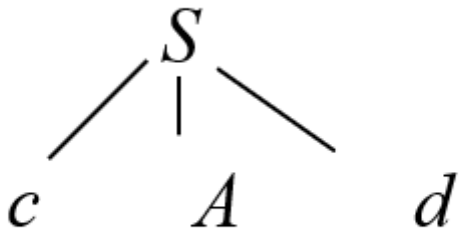
При спадній побудові дерева розбору для цього рядка ми спочатку створюємо дерево, що складається з одного вузла, позначеного як S. Покажчик входу вказує на c, перший символ рядка w.»

Крок 5: Користувач отримує практичне завдання у вигляді тесту:

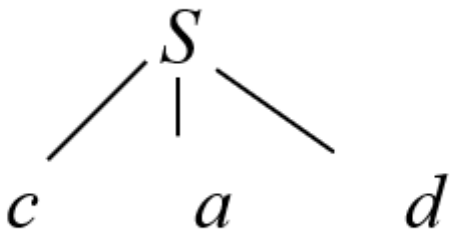
«Тепер скористаємося першою продукцією для S щоб одержати дерево

Завдання: Оберіть правильне початкове дерево для синтаксичного розбору ланцюжка $S \to cAd$.

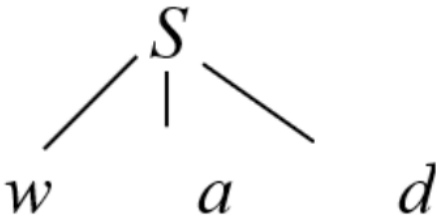
Варіанти відповіді:



1.



2.



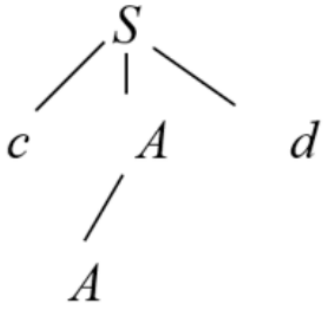
3.

Правильна відповідь – 1.»

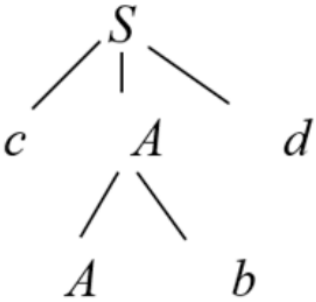
Крок 6: Користувач отримує практичне завдання у вигляді тесту:

«Крайній ліворуч лист c відповідає першому символу w, перемістимо покажчик входу до a, другого символу рядка w, і розглянемо наступний лист дерева, позначений A.

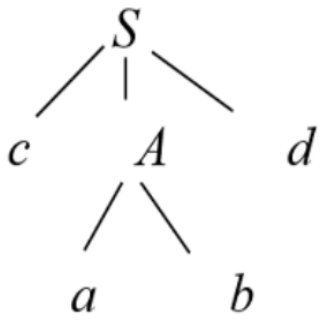
Завдання : Оберіть правильне дерево для наступного кроку синтаксичного розбору ланцюжка $S \to cAd$.



1.



2.



3.

Правильна відповідь – 3.»

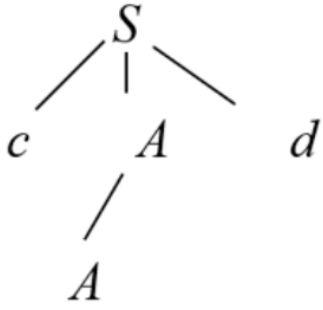
Крок 7: Користувач отримує практичне завдання у вигляді тесту:

«Виявлена відповідність зчитаного символа a ^{a} листу дерева, переходимо до наступного символа d. Однак d не відповідає листу дерева b, а отже, необхідно повернутися до A ^{A} для того, щоб вибрати нову альтернативу для роботи.

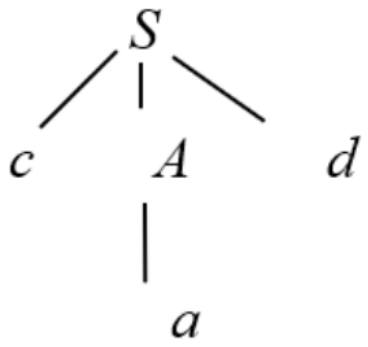
Повертаючись до A, необхідно повернути покажчик у позицію 2, у якій він був, коли ми вперше прийшли до розкладання A. Це означає, що процедура для A повинна зберігати покажчик входу в локальній змінній.

Завдання : Оберіть правильне дерево для наступного кроку синтаксичного розбору ланцюжка $S \rightarrow cAd$.

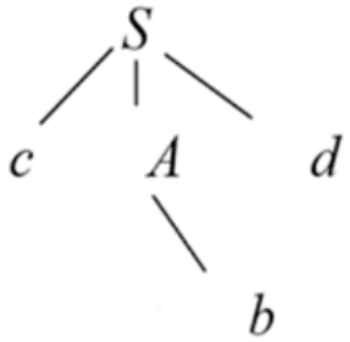
Варіанти відповіді:



1.



2.



3.

Правильна відповідь 2.»

Крок 8: Користувач отримує повідомлення про виконане завдання:

«Лист a відповідає другому символу w, а лист d – третьому. Оскільки в цей момент побудоване дерево розбору для w, припиняємо роботу і повідомляємо про успішне завершення розбору.»

Крок 9: Отримання довідкового матеріалу з теми:

«Розглянемо основний метод висхідного синтаксичного аналізу, відомий як “перенесення-згортання”. Далі будемо коротко називати його ПЗ-аналіз.

У процесі розбору типу “перенесення-згортання” будується дерево розбору вхідного рядка, починаючи з листів (знизу) до кореня (вгору). Цей процес можна розглядати як “згортання” рядка w до початкового символа граматики. На кожному кроці процесу згортання підрядок, який можна зіставити правій частині деякої продукції, заміняються символом з лівої

частини цієї продукції, і якщо на кожному кроці вибирається правильний підрядок, то у зворотному порядку простежується праве породження.»

Крок 10: Користувач отримує практичне завдання у вигляді тесту:

«Розглянемо граматику з продукціями $S \to aABe, A \to Abc|b, B \to d$.

Скануємо рядок $abbcde$ у пошуках підрядка, що відповідає правій частині якоїсь продукції. Такими підрядками є b і d. Виберемо крайнє зліва b і замінимо його нетерміналом A, що являє собою ліву частину продукції $A \to b$;

Завдання: Звести рядок $abbcde$ до S. Оберіть перший крок.

Варіанти відповіді:

1. a(b -> A -> Abc)bcde = aAbcde

2. a(b -> B)bcde = aBbcde

3. a(b -> B -> d)bcde = adbcde

Правильна відповідь – 1.»

Крок 11: Користувач отримує практичне завдання у вигляді тесту:

«Розглянемо граматику з продукціями $S \to aABe, A \to Abc|b, B \to d$.

Тепер правим частинам продукцій відповідають підрядки Abc, b і d. Хоча b і є крайнім зліва підрядком, що відповідає правій частині однієї з продукцій, виберемо для заміни підрядок Abc і замінимо його нетерміналом A відповідно до продукції $A \to Abc$.

Завдання: Звести рядок $abbcde$ до S. Оберіть другий крок.

Варіанти відповіді:

1. a(Abc -> A)de = aAde

2. (a -> A)(Abc -> A)de = AAde

3. a(A - > Abc)de = aAbcbcde

Правильна відповідь -1.»

Крок 12: Користувач отримує практичне завдання у вигляді тесту:

«Розглянемо граматику з продукціями $S \to aABe, A \to Abc|b, B \to d$.

Заміняючи d на B, ліву частину продукції $B \to d$, одержуємо $aABe$, що відповідно до першої продукції заміняється стартовим символом S. Отже, послідовність з чотирьох згортань дозволяє привести рядок $abbcde$ до стартового символа S. Ці скорочення являють собою обернене праве породження.

Завдання: Звести рядок $abbcde$ до S. Оберіть третій крок.

Варіанти відповіді:

1. aA(d -> b)e = aAbe

2. aA(d -> A -> Abc)e = aAAbce

3. aA(d -> B)e = aABe

Правильна відповідь -3.»

Крок 13: Користувач отримує повідомлення про виконане завдання:

«Розглянемо граматику з продукціями $S \to aABe, A \to Abc|b, B \to d$.

Рядок $abbcde$ зводиться до S за допомогою таких кроків:

$$abbcde$$

$$aAbcde$$

$$aAde$$

$$aABe$$

$$S$$.»

3.3 Блок-схема програми-тренажера

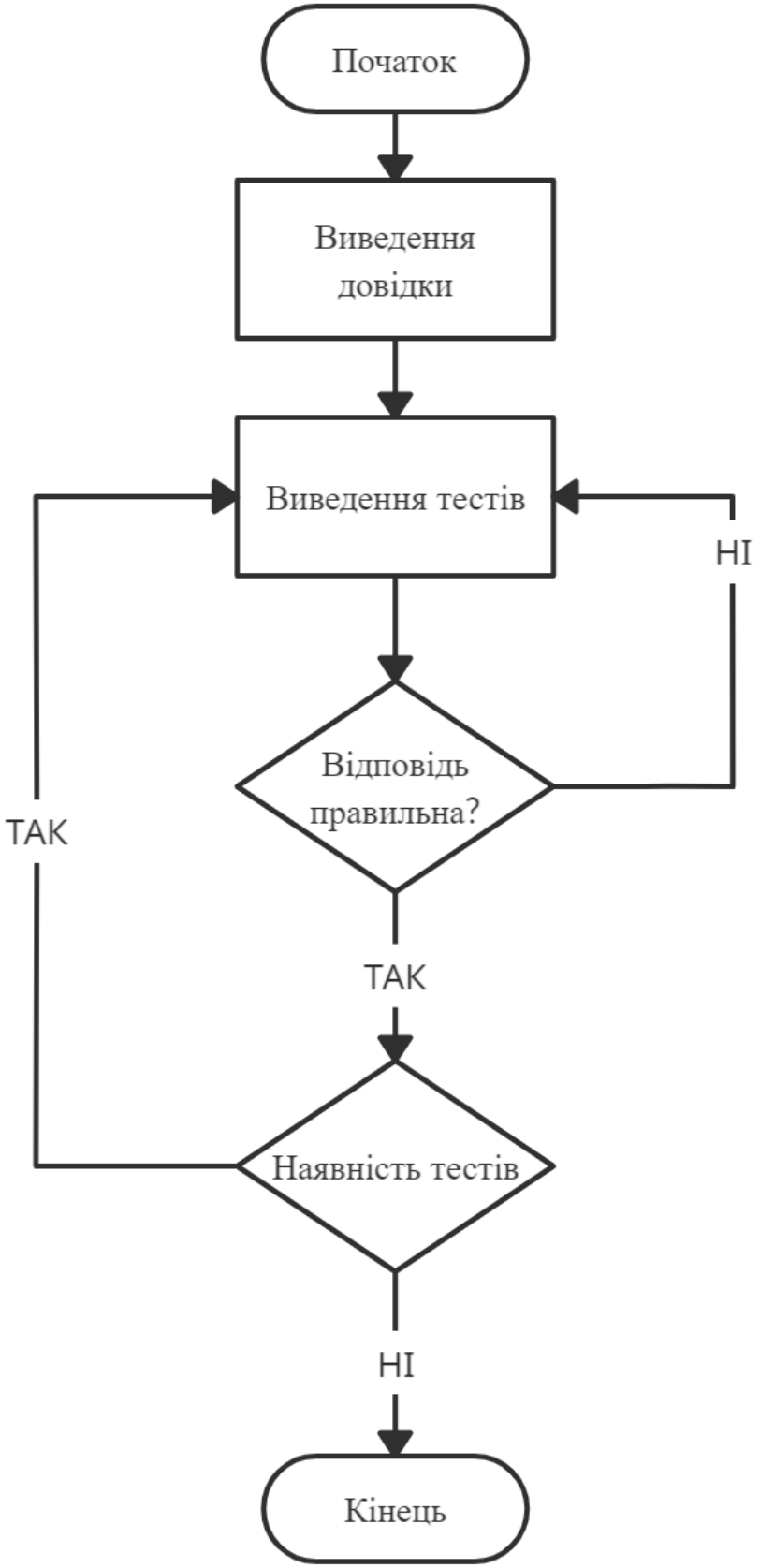


Рисунок 3.4 – Блок-схема програми-тренажера

4 Практична частина

4.1 Опис програмної реалізації

Програмна реалізація відбувалась за допомогою платформи Unity 2018 за допомогою мови програмування C#. Unity 2018 зручна та мультифункціональна платформа для розробки застосунків на будь-яку платформу, від IOS до Windows (див. Рисунок 4.1)

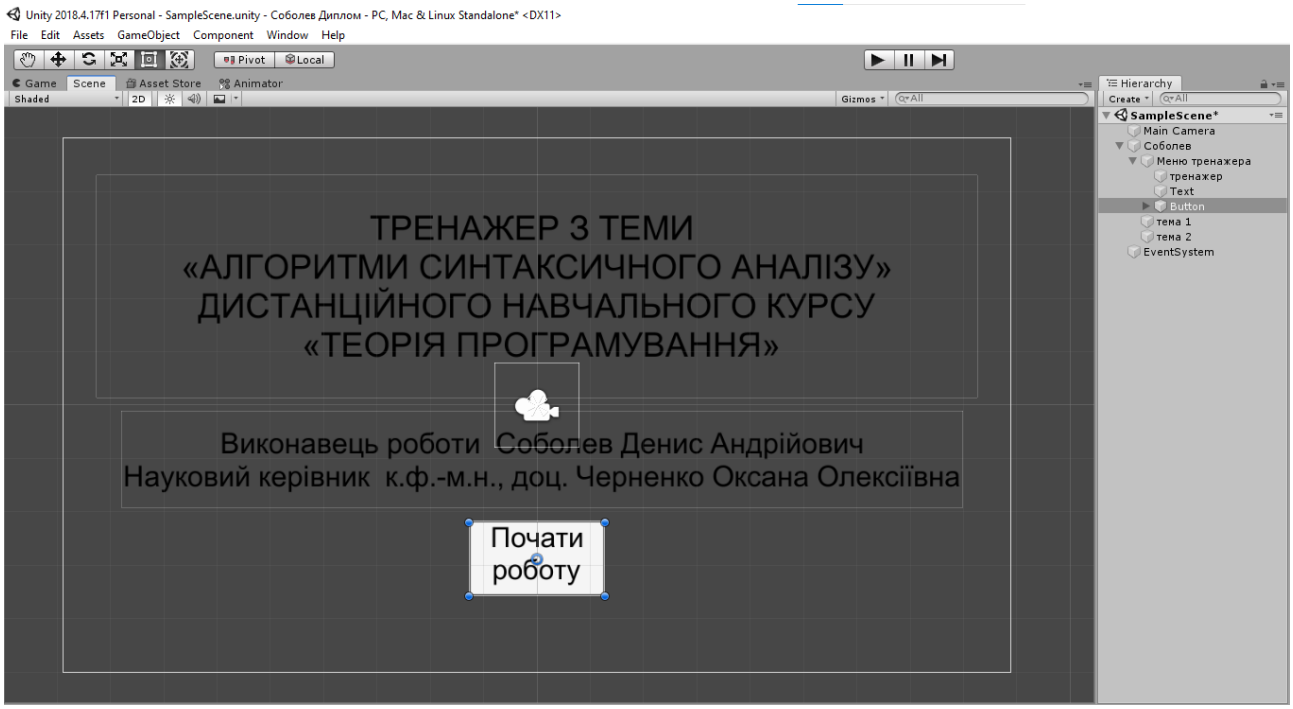


Рисунок 4.1 – Головне меню Unity 2018

Вся інформація програмного застосунку розміщується в Empty об’єктах, до яких прив’язано програмний код, всі вони розташовані в основній сцені Unity 2018 (див. Рисунок 4.2)

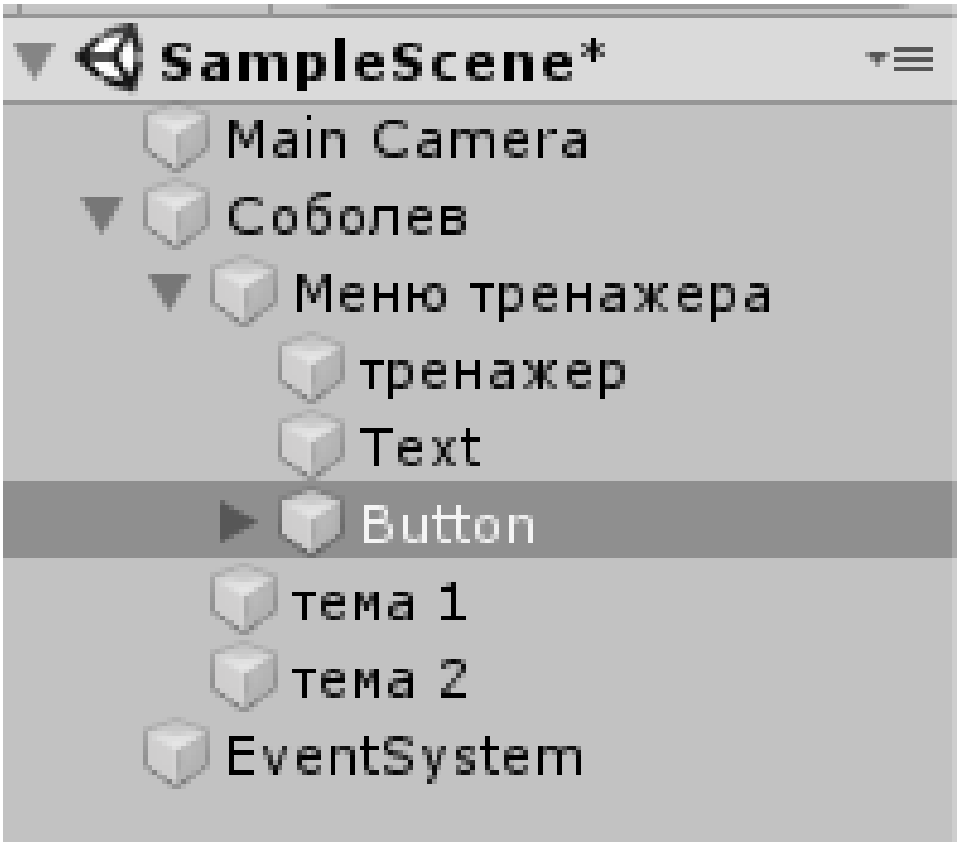


Рисунок 4.2 – Меню сцени в Unity 2018

Всі скріпти кнопок для навігації між елементами тренажеру створено за допомогою вбудованих функцій SetActive (bool) (див. Рисунок 4.3).

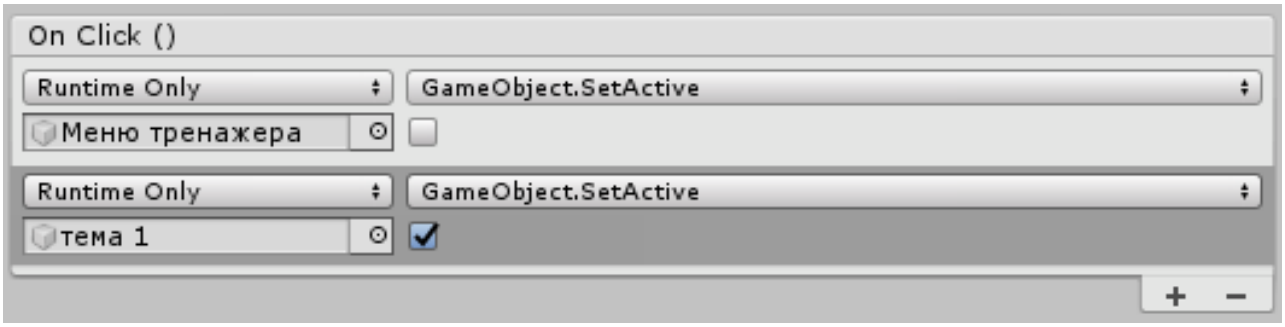


Рисунок 4.3 – Функції для роботи з кнопками

Таким чином кнопка «Почати роботу» закриває Empty об’єкт з головним меню тренажеру та відкриває Empty об’єкт з навчальним матеріалом до теми 1.

Створення об’єктів з довідковим матеріалом відбувалося за допомогою скріпту для тексту, який вставляє певну довідкову інформацію в елемент Text (див. Рисунок 4.4)

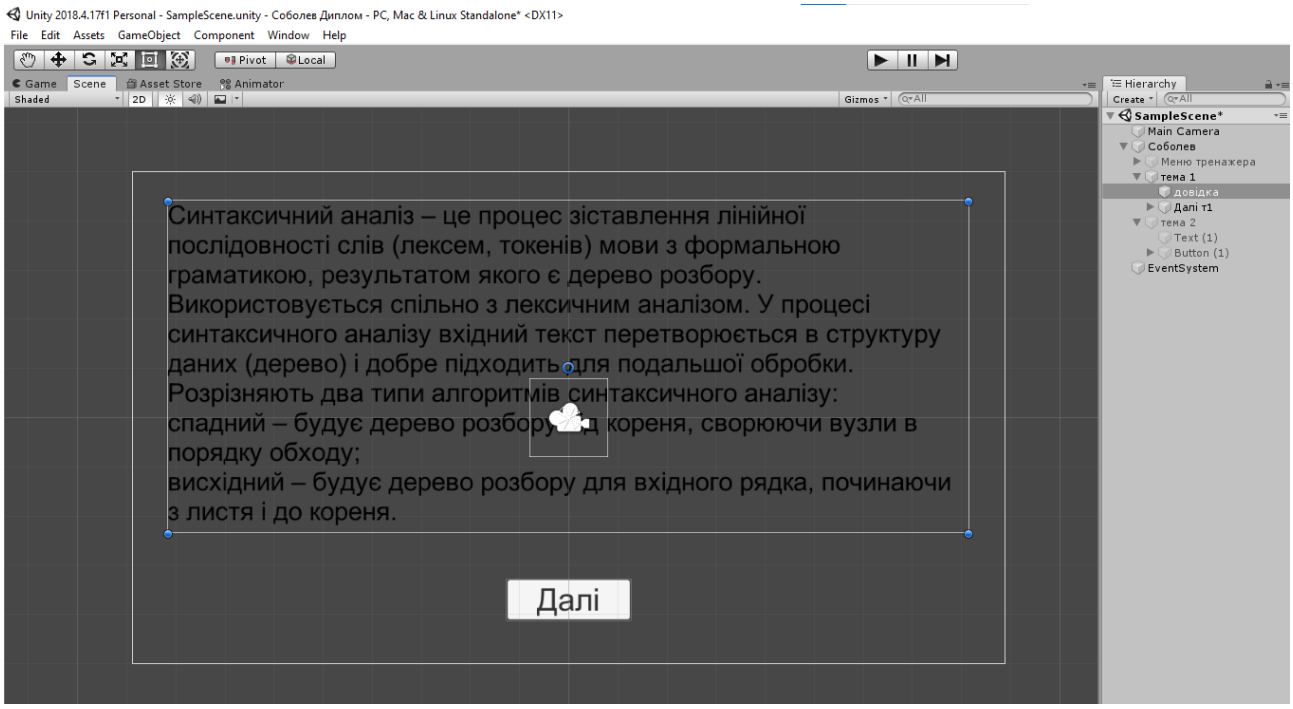


Рисунок 4.4 – Розміщення елементів в робочій зоні програми

Налаштування тексту для довідкового матеріалу (див. Рисунок 4.5)

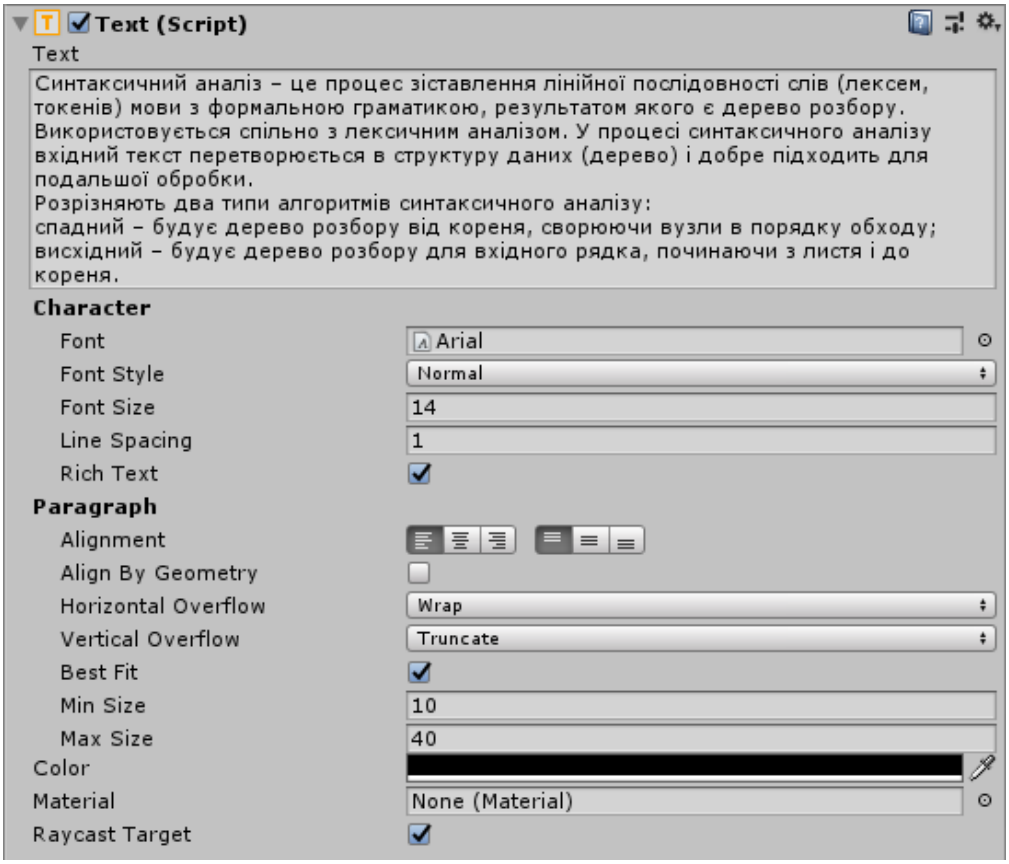


Рисунок 4.5 – Налаштування для тексту

Для створення об'єкту з практичним матеріалом використано наступне розміщення елементів (див. Рисунок 4.6)

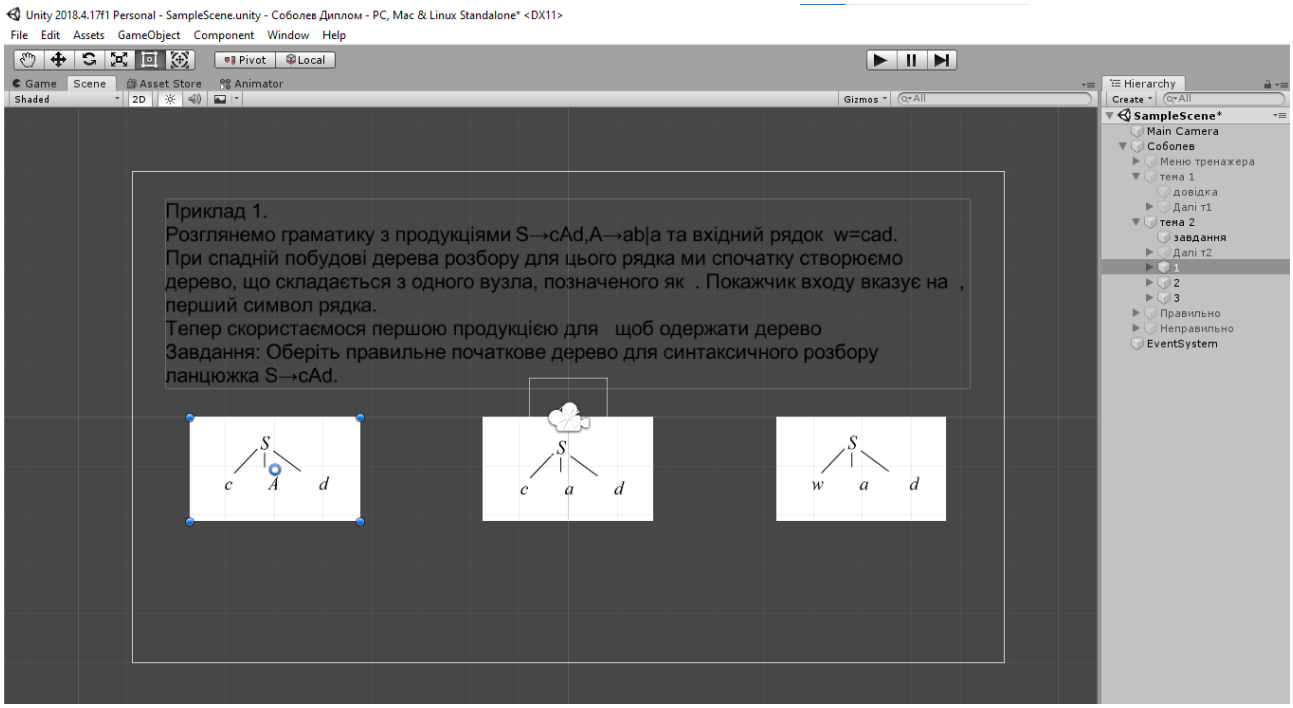


Рисунок 4.6 – Розміщення елементів для практичних завданнь

Робота з тестовим матеріалом відбувається шляхом натиснення на певне зображення варіанту відповіді. Скріпт для роботи кнопок з варіантами відповіді зображено на Рисунку 4.7.

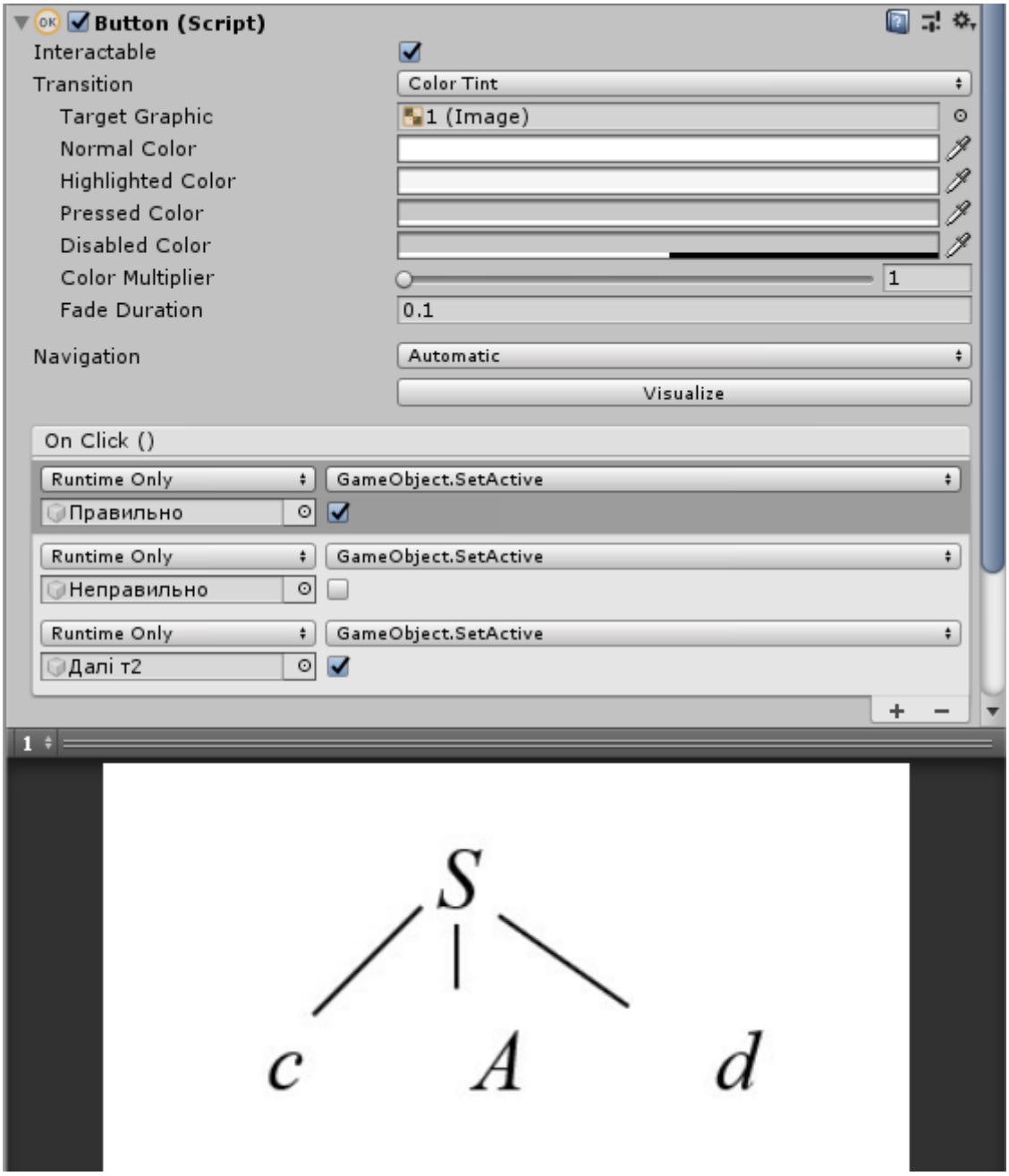


Рисунок 4.7 – Скріпт для роботи кнопок з варіантами відповіді

У випадку вибору правильного варіанту відповіді з'являється кнопка «Далі» для продовження роботи з навчальними матеріалами та повідомлення «Відповідь вірна». В іншому випадку з'являється повідомлення «Відповідь не вірна» та для продовження потрібно обрати правильну.

Типово було створено всі інші об'єкти програмного забезпечення.

4.2 Перевірка валідності програмного забезпечення

Після запуску застосунку користувач має можливість обрати будь-яке розширення екрану та якість зображення для комфортної роботи з програмою.

Вікно програми після запуску виконуваного файлу (див. Рисунок 4.8).

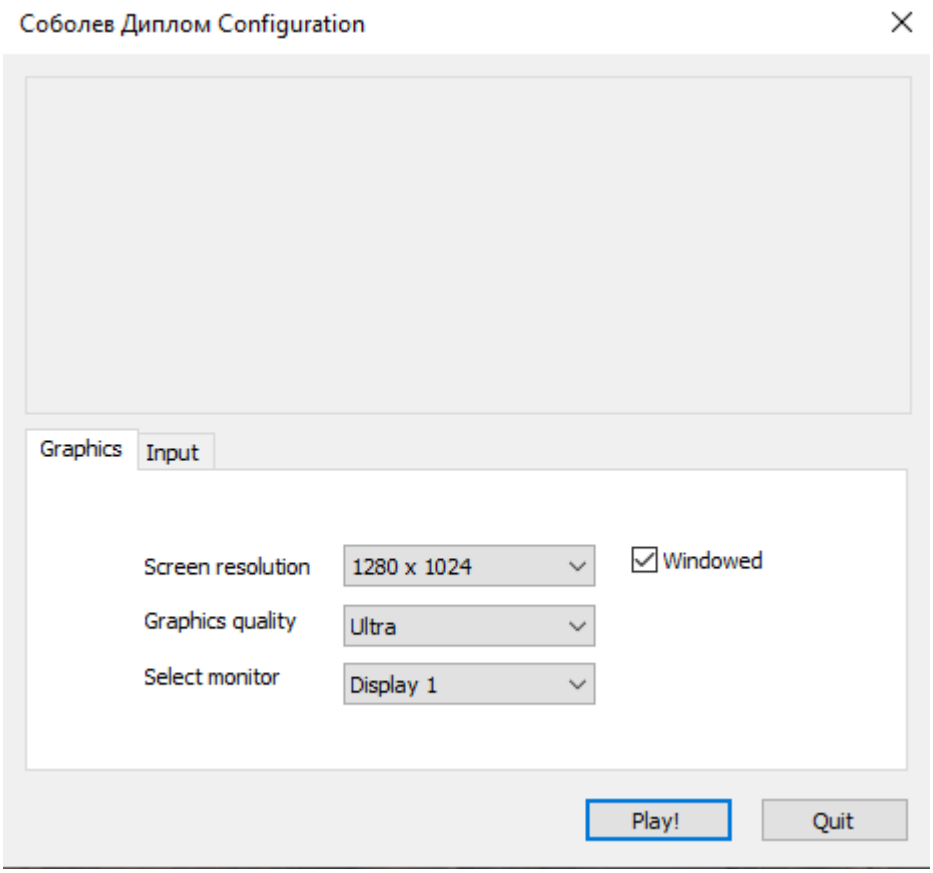


Рисунок 4.8 – Вікно з первинними налаштуваннями показу

Після вибору бажаних налаштувань та натиснення на кнопку «Play!» корисутвач переходить до головного меню тренажеру (див. Рисунок 4.9).

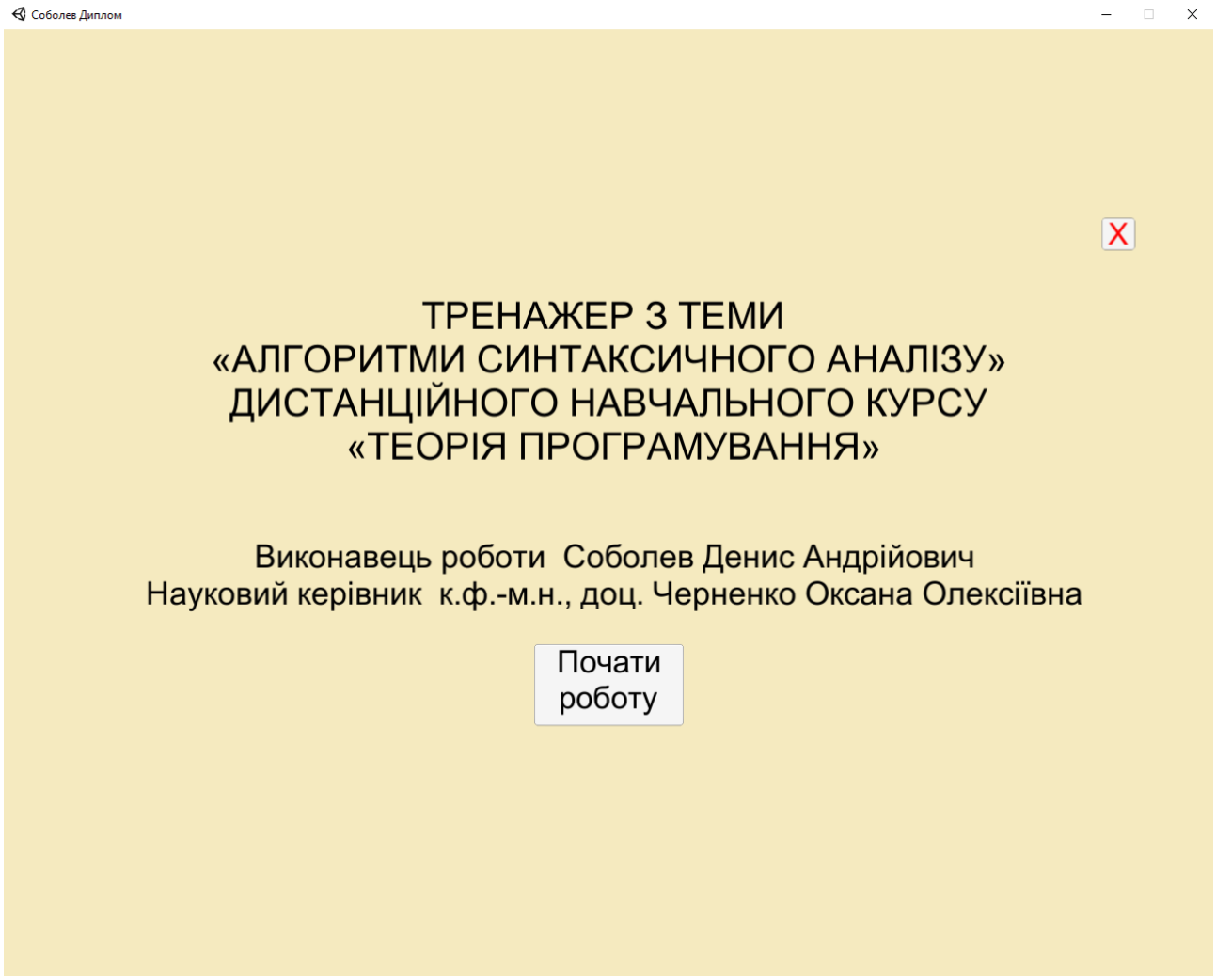


Рисунок 4.9 – Головне меню тренажеру

Після натиснення на кнопку «Почати роботу» користувач переходить до довідкового матеріалу (див. Рисунок 4.10).

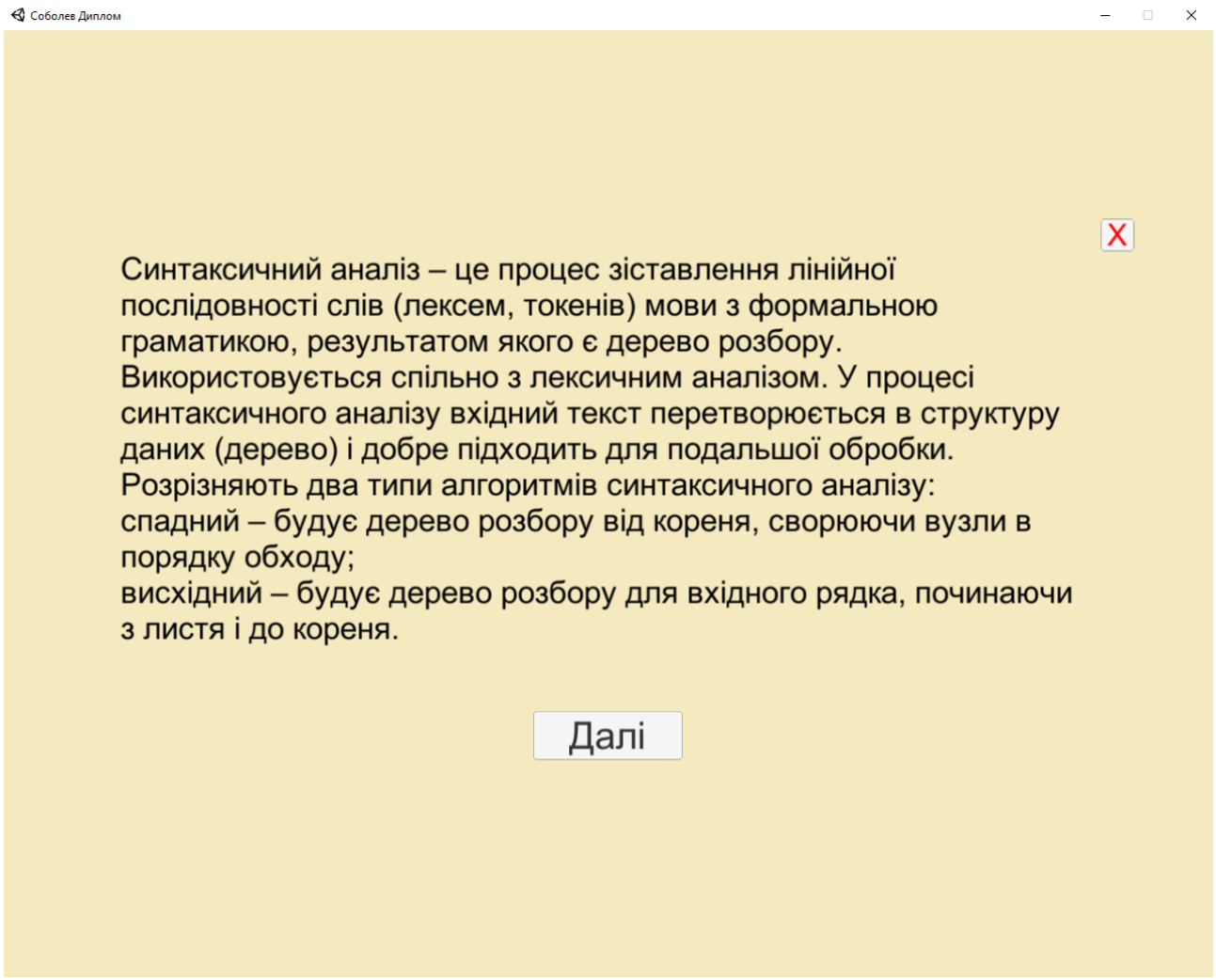


Рисунок 4.10 – Приклад оформлення довідкового матеріалу тренажеру

Після ознайомлення з довідковим матеріалом користувач переходить до тестових завданнь (див. Рисунок 4.11)

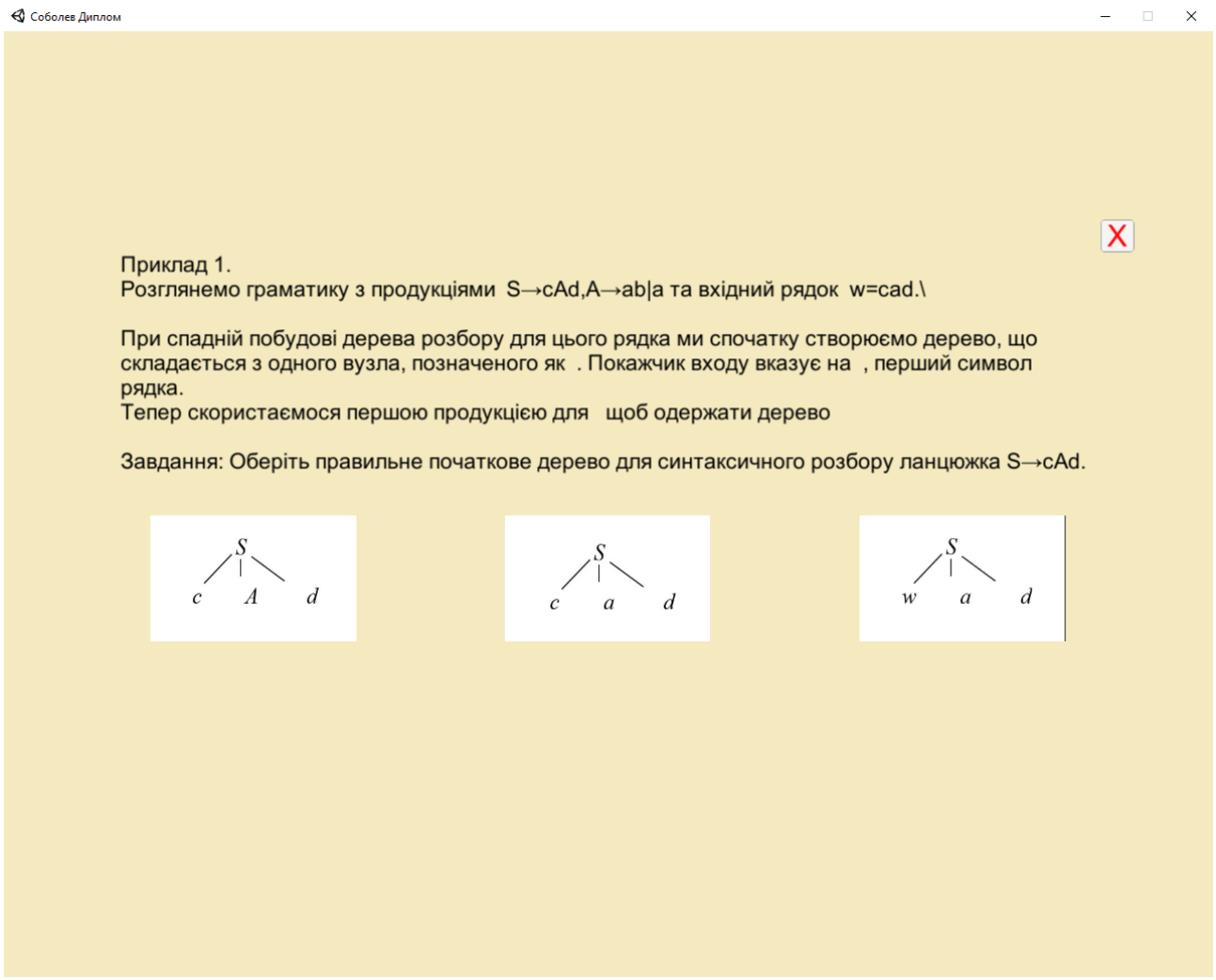


Рисунок 4.11 – Приклад оформлення тестового завдання

Для продовження роботи з тренажером необхідно натиснути на зображення з відповіддю. У разі вибору правильного варіанту з'явиться повідомлення «Відповіль вірна!» та кнопка для продовження роботи (див. Рисунок 4.12)

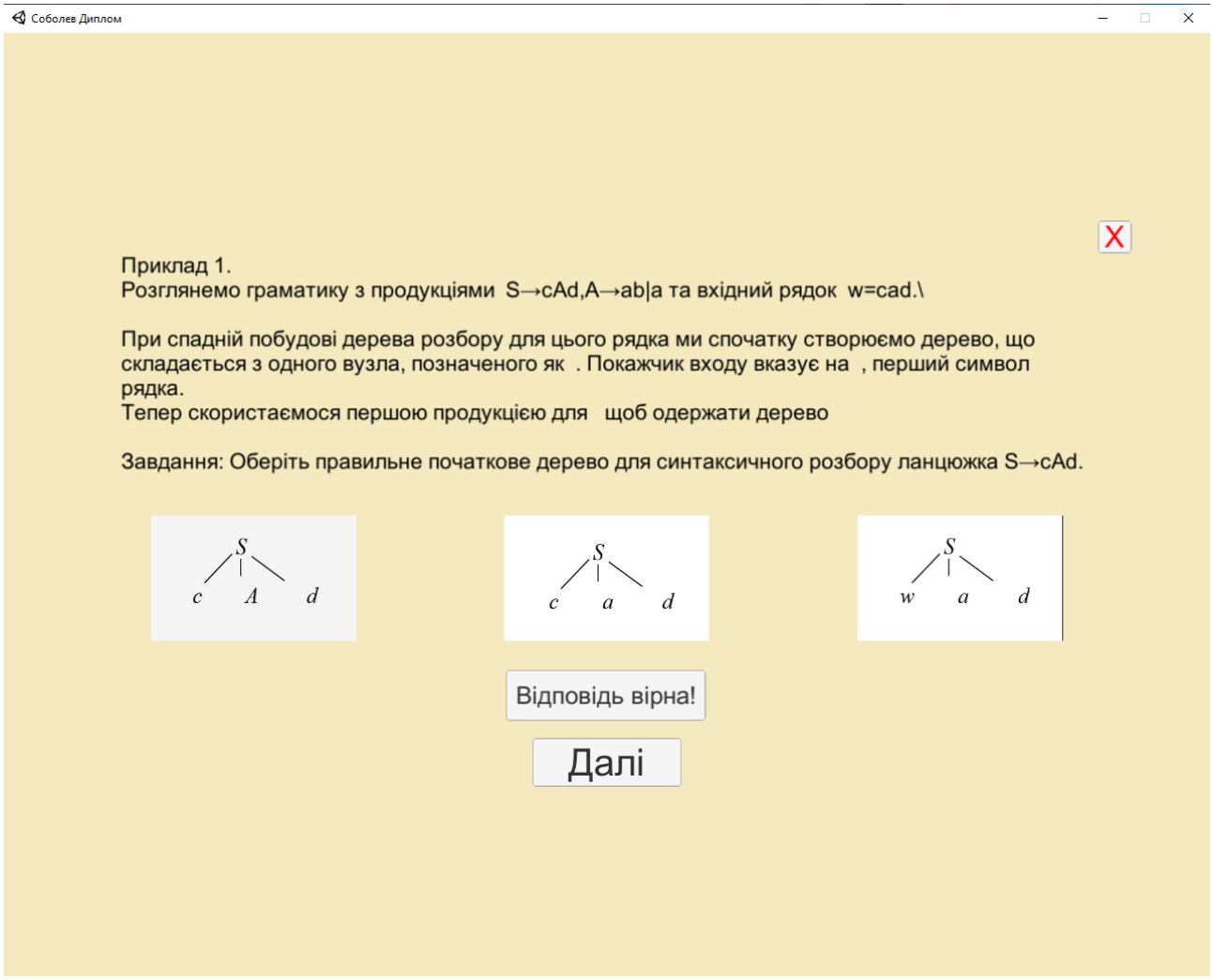


Рисунок 4.12 – Повідомлення про правильність відповіді

У разі вибору неправильного варіанту відповіді, користувач отримує повідомлення «Відповідь не вірна!» (див. Рисунок 4.13)

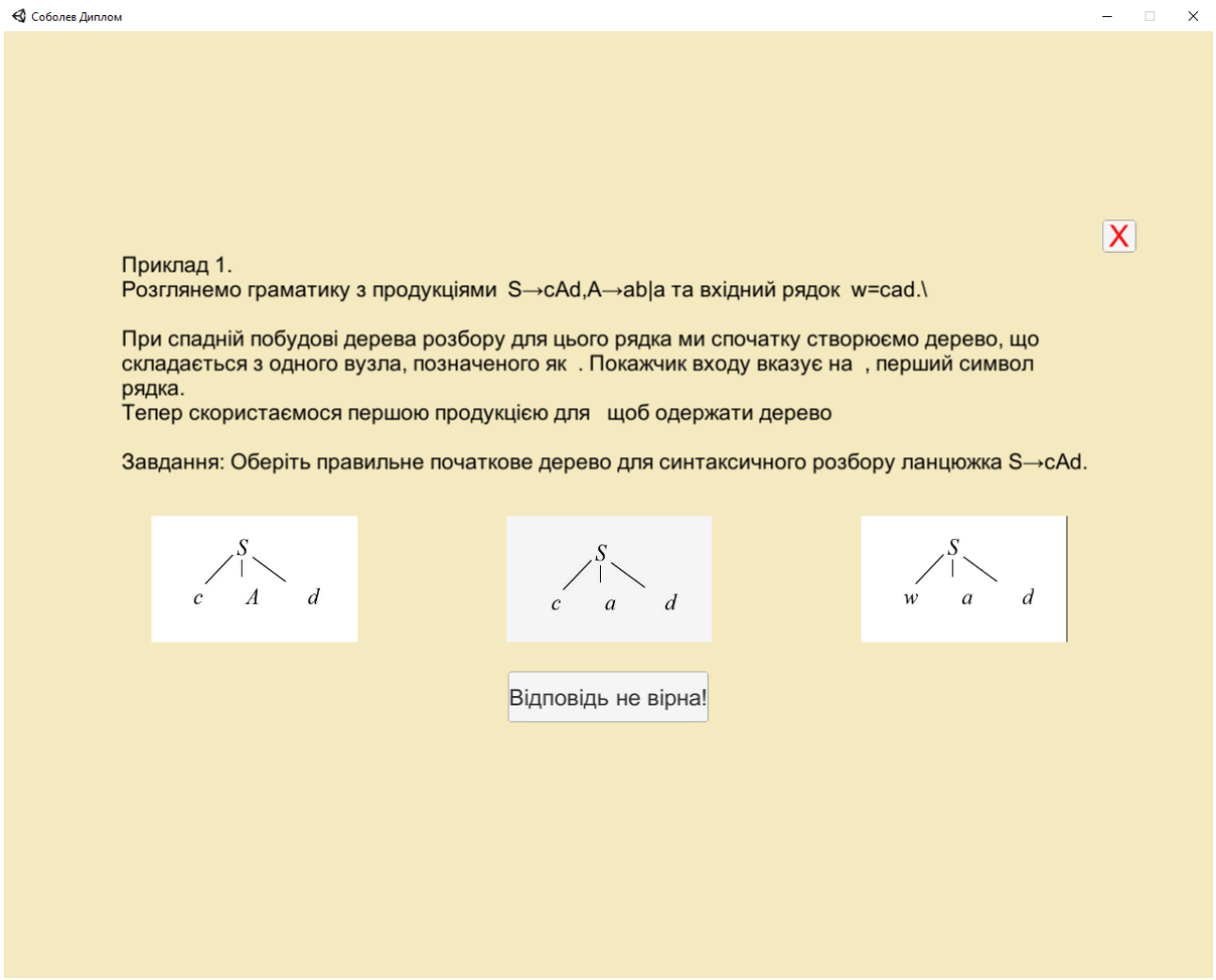


Рисунок 4.13 – Повідомлення про неправильність відповіді

Після закінчення роботи з навчальними матеріалами тренажеру на останньому елементі програми після повідомлення про завершення роботи замість кнопки «Далі» з'являється кнопка «Повтор», що дозволяє пройти тренажер без його перезапуску (див. Рисунок 4.14)

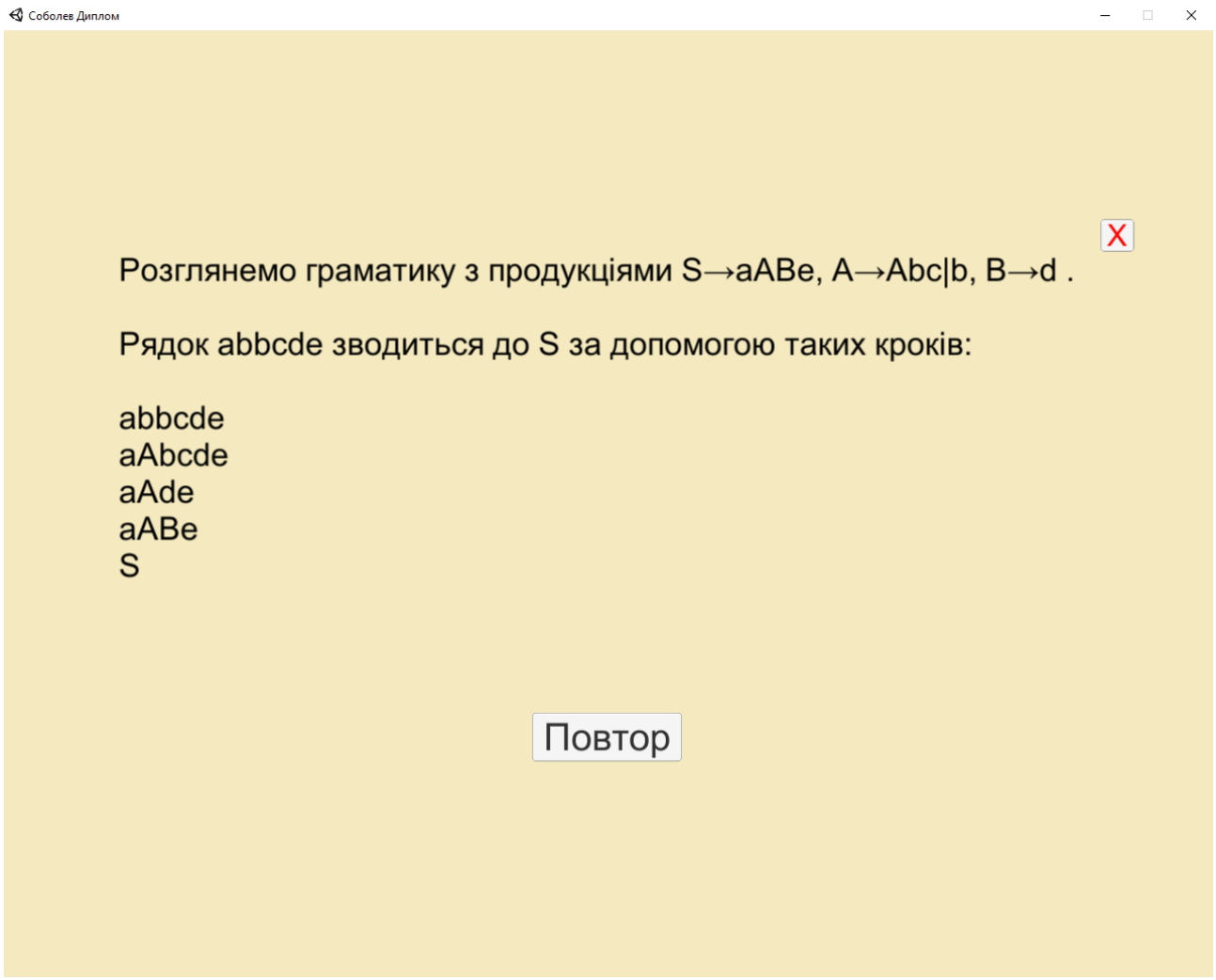


Рисунок 4.14 – Останній елемент програми та кнопка «Повтор»

ВИСНОВКИ

Під час роботи над бакалаврською роботою було розглянуто два тренажери зі схожою темою, досліджено їх плюси та недоліки, розроблено алгоритм роботи тренажеру та блок-схему алгоритму. В алгоритмі наведено довідковий матеріал та тестові завдання з теми «Алгоритми синтаксичного аналізу». Тему розглянуто за типовими прикладами.

Програмно реалізовано програму-тренажер на платформі Unity 2018 за допомогою MS Visual Studio та мови програмування C#.

Мету роботи виконано, правила оформлення дотримано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Величко, Артур Олександрович, ПОЯСНЮВАЛЬНА ЗАПИСКА ДО БАКАЛАВРСЬКОЇ РОБОТИ на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТРЕНАЖЕРА З ТЕМИ «СПОСОБИ ЗАДАННЯ МОВ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ» [Електронний ресурс]
   Режим доступу до ресурсу: http://dspace.puet.edu.ua/handle/123456789/9031
2. Белінський, Олексій Борисович, ПОЯСНЮВАЛЬНА ЗАПИСКА ДО БАКАЛАВСЬКОЇ РОБОТИ на тему ТРЕНАЖЕР З ТЕМИ «ПОБУДОВА ПРЕДИКАТИВНИХ СИНТАКСИЧНИХ АНАЛІЗАТОРІВ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ» ТА РОЗРОБКА ЙОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ [Електронний ресурс]
   Режим доступу до ресурсу: http://dspace.puet.edu.ua/handle/123456789/8999
3. el.puet.edu.ua, Черненко О.О. Теорія програмування, Синтаксичний аналіз [Електронний ресурс]

Режим доступу до ресурсу:
http://www2.el.puet.edu.ua/st/mod/page/view.php?id=105325

4. wikipedia.org , Синтаксичний аналіз [Електронний ресурс]

Режим доступу до ресурсу:
https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7

5. elartu.tntu.edu.ua, Синтаксичний аналіз [Електронний ресурс]

Режим доступу до ресурсу: http://elartu.tntu.edu.ua/bitstream/123456789/16531/2/VseukrStud_2016v1_Vanat_V-Concept_parsing_33.pdf

6. el.puet.edu.ua, Черненко О.О. Теорія програмування, Синтаксичний аналіз [Електронний ресурс]

Режим доступу до ресурсу: http://www2.el.puet.edu.ua/st/mod/page/view.php?id=105326

7. unity.com, Платформа розробки програмного забезпечення [Електронний ресурс]
   Режим доступу до ресурсу: https://unity.com/ru
8. learn.unity.com, Засіб вивчення платформи [Електронний ресурс]

Режим доступу до ресурсу: https://learn.unity.com/

9. docs.microsoft.com, Засіб вивчення мови програмування [Електронний ресурс]

Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/?view=vs-2019

10. _Unity Manual, Засіб роботи з платформою [Електронний ресурс]

Режим доступу до ресурсу: http://39.104.106.62/Manual/OfflineDocumentation.html