Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Нємковича Владислава Володимировича*
(ПІБ)

академічної групи *121-18ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка середовища для створення програмного коду розгалуженого сюжету комп'ютерних ігор*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18ск-1* (група) *Нємковича Владислава Володимировича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка середовища для створення програмного коду розгалуженого сюжету комп'ютерних ігор*

затверджена наказом ректора НТУ «ДП» від «07» червня 2021 р. № 317-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021р.***

Завдання видав _____ *доц. Гуліна І.Г.*.
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Нємкович В.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *15.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 68 с., 20 рис., 3 дод., 28 джерел.

Об'єкт розробки: веб-редактор епізодів для програмування розгалуженого сужету ігор.

Мета кваліфікаційної роботи: є теоретичне і експериментальне обґрунтування вимог щодо функціональних можливостей візуального редактору сюжету та оцінка технологічності застосування бібліотеки на мові C# при розробці реальних ігор, та полегшення створення сюжетних ігор невеликими студіями та поодинокими розробниками.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування застосунку, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження сайту, описана робота інтернет-магазину.

В економічному розділі визначено трудомісткість розробленого програмного продукту, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні веб-додатку графічного редактору, який дозволяє створювати, зберігати, відкривати, редагувати та експортувати граф епізоду. Створено скрипт на мові JS, який на основі шаблону формує готовий для роботи зі створеною мною бібліотекою клас на мові C#. Було створено максимально просту бібліотеку на мові C# для управління епізодом, та обробкою різних варіантів дії з боку гравця.

Проект розроблено для полегшення створення сюжетних ігор невеликими студіями та поодинокими розробниками.

Актуальність програмного продукту визначається великим попитом на подібні інструменти.

Список ключових слів: ВЕБ-ДОДАТОК, ГРАФІЧНИЙ РЕДАКТОР.

ABSTRACT

Explanatory note: 68 p., 20 figs., 3 apps., 28 sources.

Object of development: web editor of episodes for programming of the branched subject of games.

The purpose of the qualification work: there is a theoretical and experimental justification of the requirements for the functionality of the visual plot editor and assessment of the manufacturability of the library in C # when developing real games, and facilitating the creation of story games by small studios and individual developers.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the application, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download site, describes the Internet -shop.

In the economic section, the complexity of the developed software product is determined, the cost of work on creating the application is calculated and the time for its creation is calculated.

The practical value is to create a web application graphic editor that allows you to create, save, open, edit and export an episode graph. A script in the JS language has been created, which, based on a template, forms a class in the C # language ready to work with the library I created. The simplest C # library has been created to manage the episode and handle different player options.

The project is designed to facilitate the creation of story games by small studios and single developers.

The relevance of the software product is determined by the high demand for such tools.

Keywords: WEB APP, GRAPHIC EDITOR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

ПК - персональний комп’ютер;

СУБД - система управління базами даних;

API - Application Programming Interface;

BE - BackEnd;

CMS - система управління контентом;

DOM - Document Object Model;

IT - інформаційні технології;

TS - Type Script;

ORM - Object-Relational Mapping;

HTML - Hypertext Markup Language.

ВСТУП

Тема розробки комп’ютерних ігор на сьогоднішній момент дуже актуальна. Розробники ігор нерідко заробляють значні гроші на одній грі. Почати створювати ігри може кожен, але у самостійній розробці ігор є проблема – усю роботу доводиться робити власноруч. Саме тому в інтернеті можна знайти багато дешевих або зовсім безкоштовних бібліотек для розробки. Але не дивлячись на це, такі речі, як розгалуженість сюжету доводиться створювати все ж таки власноруч.

Є два основні жанри ігор, які побудовані на розгалуженості сюжету, де на майбутні події може впливати гравець, обираючи один з запропонованих варіантів дії. Це інтерактивне кіно та візуальні новели. Загалом розробник, який бажає створити таку гру зазвичай натикається на дві проблеми – графічна частина гри (якщо візуальні новели є простими двовимірними іграми, то інтерактивне кіно – це зазвичай 3D-гра, і краса віртуального світу – це один з основних чинників, які впливають на враження від гри) і сюжет. Останню проблему саме намагатимемося вирішити у цій роботі, створивши зручний набір інструментів для розробників.

Для створення бібліотеки було обрано мову програмування C#, оскільки найпопулярніший серед поодиноких розробників ігор рушій «Unity» використовує саме цю мову для створення ігрової логіки.

Сутність проблеми, яка розглянута в кваліфікаційній роботі, полягає у вивченні практики застосування розгалуженого сюжету у відеоіграх та у аналізі різних підходів до вирішення подібної задачі.

Метою є теоретичне і експериментальне обґрунтування вимог щодо функціональних можливостей візуального редактору сюжету та оцінка технологічності застосування бібліотеки на мові C# при розробці реальних ігор.

Об’єктом розробки є інструментарій для побудови розгалуженого сюжету ігор.

При аналізу предметної галузі а саме, існуючих відеоігор та пошук схожих патернів сюжету, які у майбутньому можна було б реалізувати в редакторі та бібліотеці що розробляється в роботі.

Для досягнення цієї мети в роботі ставляться та вирішуються такі завдання:

1. Зробити теоретичне обґрунтування роботи.
2. Створити бібліотеку на мові C#, яка б дозволяла реалізовувати розгалужений сюжет у грі.
3. За допомогою створеної бібліотеки створити програму (подібну до комп’ютерної гри, у якій можна було б використати розгалужений сюжет), та продемонструвати наступне:

а) проходження лінійної частини сюжету (так званих чекпоінтів);

б) вибір одного з двох варіантів дій;

г) зміни у наступних чекпоінтах залежно від вибору гравця.

Практичним результатом даної кваліфікаційної роботи є створення гнучкої бібліотеки на мові C#, яка концептуально і функціонально адаптована до ігрового рушія “Unity” та в цілому розробки на мові C#. Як наслідок – продукт, що не має аналогів на сучасному ринку індустрії розробки комп’ютерних ігор, і має багато плюсів:

– онлайн-редактор є у кілька разів зручнішим за офлайн додаток, оскільки не потребує встановлення;

– створення сюжету у зручному редакторі полегшує роботу сценаристам, а його експорт у клас на мові C# допомагає вже розробникам;

– можливість портувати бібліотеку на будь-яку мову без значних зусиль.

Завдання кваліфікаційної роботи та об'єкт її діяльності безпосередньо пов'язані зі спеціальністю 121 «Інженерія програмного забезпечення» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених компетенцій.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

Ігри - це мільярдна галузь промисловості, яка фокусується на приведенні найкращих електронних ігор або відеоігор на комп'ютер або ігрові приставки.

Ігрове програмування є підмножиною розробки ігор і є процесом розробки програмного забезпечення відеоігор. Створення ігор передбачає багато областей знань, таких як: моделювання, комп'ютерна графіка, штучний інтелект, фізика, аудіопрограмування і введення.

З усіма інструментами, доступними на ринку сьогоднішньої ігрової індустрії, легше, ніж будь-коли, розпочати розробку ігор з або без будь-якої спеціалізованої освіти. Нелегко стати успішним розробником гри, або створити хорошу гру, яку люди захочуть грати, але це неможливо.

Перед тим, як розпочати розробку середовища програмування для створення сюжетів, ігор, які використовують розгалужений сюжет, слід переглянути програми розробки ігор та виявити їх функціональні можливості.

GameMaker 2 є переписаною версією Game Maker: Studio, яка вийшла в 1999 році. Сьогодні вона стала одним з найпопулярніших і найактивніших двигунів для розробки безкоштовних ігор, доступних на ринку. Компанія також регулярно випускає оновлення нових функцій.

GameMaker2 дозволяє створювати цілі ігри за допомогою інтерфейсу перетягування або кодування. Це програмне забезпечення дає вам багато енергії, використовуючи її гнучку мову Game Maker, яка схожа на C ++.

Ця програма підтримує безліч функцій якості життя, таких як можливість додавати покупки до вашої гри, аналітику в реальному часі, керування джерелами, багатокористувацьку мережу. GameMaker2 має вбудовані редактори для зображень, анімацій і відтінків. Також можна розширити можливості GameMaker2, використовуючи розширення третьої сторони.

Основні функції:

- Дії перетягування - перетягування, перемикання, структури даних, буфери, ini-файли тощо.
- Оптимізований графічний інтерфейс.
- Sprite Editor.
- Інструмент - інструмент Magic Wand, інструмент для дуги тощо.
- Система шарів.
- Редагування розділеного екрана.
- Перероблено робочі місця GMS2.

Unity почався як 3D-движок, але він також підтримує 2D. Як попередження, ви можете іноді зіткнутися з деякими незначними проблемами при створенні 2D ігор. Зрештою, 2D система Unity побудована на 3D-системі ядра. Це також означає, що Unity додає багато непотрібних наворотів до 2D-ігор, що може вплинути на продуктивність. Програма не має розробки компонент-суб'єкт, але відіграє важливу роль у її популяризації. Ця платформа надає можливість розглядати все в грі як об'єкт, що можна редагувати, будучи в змозі прикріплювати різні компоненти до кожного об'єкта. Ця функція дозволяє керувати поведінкою об'єкта та логікою для отримання оптимальних результатів.

Щоб отримати максимальну віддачу від Unity, потрібно використовувати мову програмування C #. Хороша новина полягає в тому, що ви можете швидко зачепити цей інструмент, навіть якщо ви його раніше не використовували. Є буквально тисячі навчальних посібників, доступних на YouTube і на офіційному сайті Unity.

З Unity дуже легко публікувати свої твори за допомогою одного з підтримуваних форматів експорту, сумісних з Windows, Mac, Linux, Android і т.д. За допомогою цієї програми можна створювати ігри для систем VR, як Oculus Rift і Steam VR, а також ігрові консолі.

Unity Asset Store дає вам можливість завантажувати та використовувати різні попередньо зроблені функції у вашій грі. Ви можете легко додавати тривимірні моделі, графіку HUD, екологічні текстури, діалогові системи тощо.

Godot є ще одним чудовим варіантом для ваших потреб у програмуванні ігор. Цей движок підтримує створення ігор 2D і 3D. На відміну від Unity, підтримка Godot для 2D обробки набагато краще, оскільки вона була ретельно розроблена з самого початку. Це забезпечує більшу продуктивність, менше помилок і більш чистий загальний досвід.

Спосіб наближення до ігрової архітектури Годота унікальний тим, що все поділяється на сцени. Сцена - це набір елементів, таких як спрайти, звуки та скрипти. Ви можете використовувати цю функцію, щоб об'єднати кілька сцен у велику сцену, і тоді ви можете об'єднати ці сцени в ще більшу. Цей ієрархічний дизайн дозволяє легко залишатися організованим і легко змінювати конкретні елементи.

Це програмне забезпечення використовує систему перетягування для зберігання вкладок на елементах, але кожен з цих елементів може бути розширений за допомогою вбудованої системи сценаріїв, яка використовує мову мови Python, звана GDScript.

Godot є відмінним варіантом для початківців в ігровому дизайні, тому що мова програмування проста у використанні навіть без будь-якого досвіду кодування.

Так само, як Unity, Godot Engine може розгортатися на декількох платформах, включаючи Windows, Mac, Linux, Android, HTML5 тощо.

Основні функції:

- Швидкі швидкості обробки.
- Нові оновлення щороку.
- Фізика.
- Подальша обробка.
- Різні вбудовані редактори.
- Налагодження в реальному часі.

– Керування джерелом.

Unreal Engine 4 розроблений майстрами ігрової індустрії, і це найпрофесійніший движок програмування з цього списку.

Ця платформа дуже ефективна, тому що вона була розроблена спеціально для того, щоб дозволити вам ітерацію та розвиток якнайшвидше. Ви можете використовувати його без будь-якого досвіду кодування, оскільки його система Blueprint дозволяє створювати логіку гри без кодування.

Ця функція може бути використана для створення цілих ігор, навіть складних, без відкриття редактора джерел, але якщо ви хочете пройти в глибину, ви можете легко кодувати власні креслення.

Канал YouTube YouTube має широкий діапазон відео (довжиною від 20 до 60 хвилин), який переглядає всі функції цього двигуна, і навчить вас його використовувати.

Основні функції:

– Налагодження.
– Гаряче перевантаження.
– Оптимізований потік активів.
– Попередній перегляд миттєвої гри.
– Штучний інтелект.
– Кінематографічні інструменти.
– Ефекти після обробки.

Construct 3 це легкий варіант програмного забезпечення для програмування 2D ігор, який підтримує багато платформ, але орієнтований переважно на розробку HTML5.

Магазин Scirra, що входить до цього програмного забезпечення, надає доступ до різних типів елементів - музичних пакетів і навіть ігор, створених іншими членами спільноти.

Ключові риси:

– Multi платформа.
– Перегляд макета.

– Tilemaps.
– Інтегровані редактори даних.
– Миттєвий перегляд.
– Попередній перегляд у прямому ефірі.
– Хмарні технології зберігання.
– Шари і спецефекти.

Аби зробити зручне та практичне рішення для створення сюжетів, треба подивитись на ігри, які вже використовують розгалужений сюжет.

Detroit: Become Human. Дуже популярна гра у жанрі інтерактивного кіно від студії Quantic Dream підіймає питання, чи може штучний інтелект усвідомити себе та почати жити на рівних з людьми правах (рис. 1.1).

Загалом гру можна поділити на дві умовні частини, в першій нам розповідають історію у спокійному форматі, знайомлять зі світом, персонажами. У другій частині починаються різкі події, і гравцю потрібно робити важливі вибори, які напряму впливають на подальший сюжет.

Рис. 1.1. Вікно гри «Detroit: Become Human» з меню вибором дії гравця

В цій грі є наступні види подій, які впливають на подальший розвиток подій:

Прямий вибір одного з варіантів – гравцю пропонується обрати один з варіантів дії, натиснувши одну з кнопок на клавіатурі, або геймпаді. Часто гра дає обмежену кількість часу на обмірковування, якщо гравець не встигає вибрати варіант – автоматично активується останній пункт в списку.

QTE (Quick Time Event) – гравець повинен вчасно натискати клавіші на клавіатурі, або на геймпаді. Найчастіше QTE використовується у сценах бійки, де є кілька можливих варіантів результату, наприклад: якщо провалити більшу частину QTE, персонаж отримує серйозні травми, якщо половину – збігає з місця бійки, якщо ж майже все QTE виконано ідеально, персонаж отримує перемогу над супротивником.

Відношення з персонажами – деякі вибори в грі не впливають на сюжет напряму, вони змінюють лише діалоги, та показник відношення з різними персонажами. В кінці гри ці показники впливають на деякі сцени, або навіть на важливі сюжетні події.

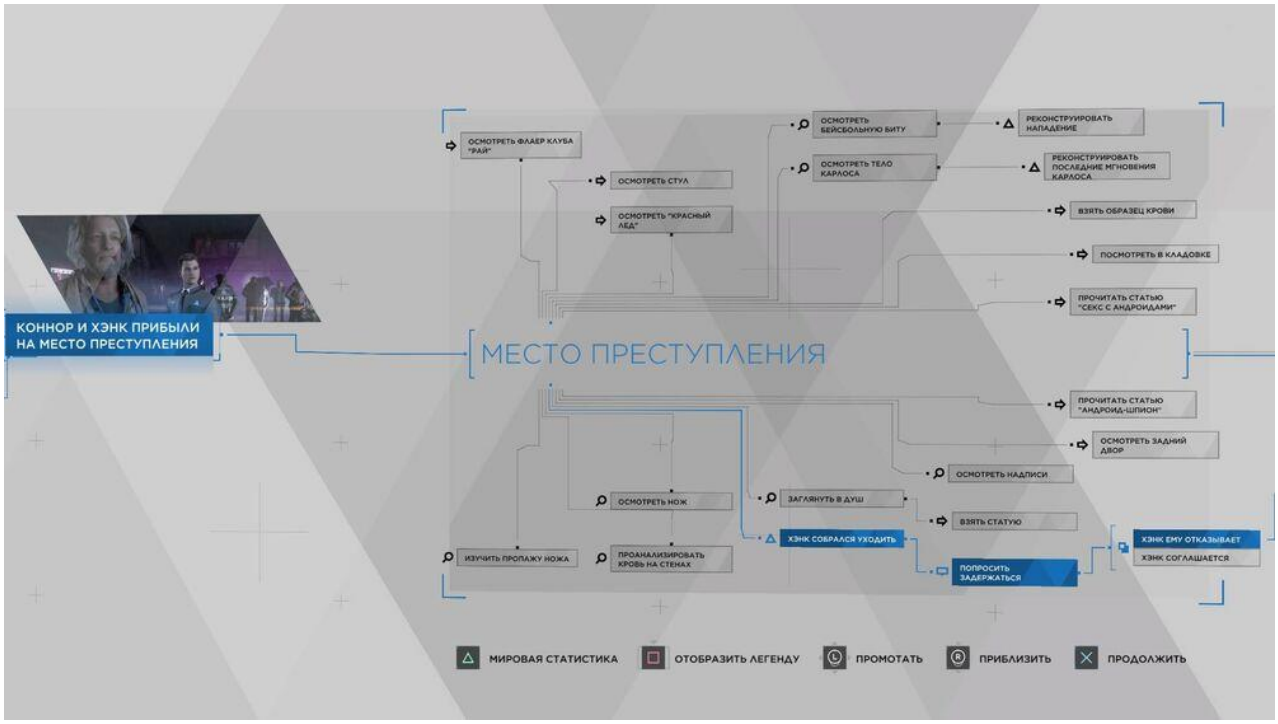

Рис. 1.2. «Карта сюжету» гри «Detroit: Become Human»

По проходженні кожного епізоду гри, нам демонструється «карта сюжету», де ми можемо побачити, які зробили вибори, на що вони вплинули та порівняти їх з виборами гравців зі всього світу завдяки вбудованій статистиці (рис. 1.2).

Everlasting Summer. Візуальна новела Everlasting Summer, створена зусиллями ентузіастів з усіх країн СНГ, розповідає в першу чергу про містичну історію потрапляння головного героя Семена у таємничий радянський літній табір «Совеня» і про його відношення з піонерами та піонерками (рис.1.3).

Рис.1.3. Візуальна новела Everlasting Summer

Гра має лише один вид дії гравця, який впливає на сюжет – простий вибір між кількома пунктами. В Everlasting Summer такі вибори зазвичай впливають тільки на наступну сцену, але також в деяких випадках впливають на відношення з персонажами.

Ближче до кінця гри показники відношення з персонажами починають значно впливати на сюжет, і саме від цих показників залежить кінцівка гри.

Until Dawn. Інтерактивне кіно у жанрі Survival Horror «Until Dawn» від студії Supermassive Games – ще одна гра, яка на відміну від попередніх вийшла

ексклюзивно для ігрової консолі PlayStation 4, і яку є сенс розібрати в якості прикладу (рис. 1.4).

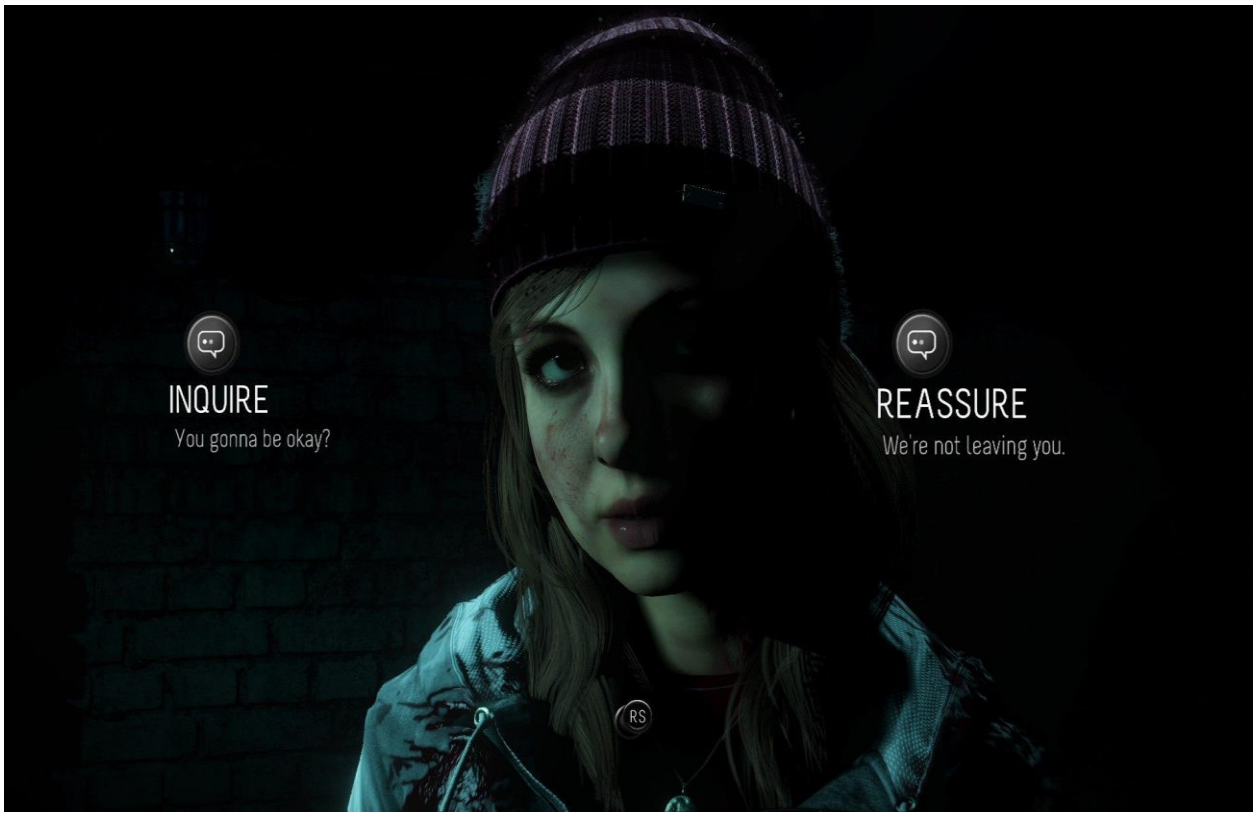

Рис. 1.4. Інтерактивне кіно у жанрі Survival Horror «Until Dawn» від студії Supermassive Games

На відміну від Detroit: Become Human ця гра завжди дає нам лише 2 варіанти вибору, а також ще більше покладається на показники відношення між персонажами.

Ще ця гра дає нам новий вид дії, що впливає на сюжет – сцени, в яких необхідно тримати геймпад нерухомо, відчуваючи всю напруженість ситуації, в яку потрапив персонаж.

Також гра демонструє ще одну цікаву фішку по роботі з сюжетом – тотеми, підібравши які персонаж бачить короткий відрізок можливого майбутнього, на яке, однак, гравець може вплинути зробивши правильний вибір.

1.2. Призначення розробки та область застосування

Веб-додаток графічного редактору дозволяє створювати, зберігати, відкривати, редагувати та експортувати граф епізоду. Було створено скрипт на мові JS, який на основі шаблону формує готовий для роботи зі створеною бібліотекою клас на мові C#. Було створено максимально просту бібліотеку на мові C# для управління епізодом, та обробкою різних варіантів дії з боку гравця.

Метою розробки є створення інструменту для полегшення процесу розробки сюжетних ігор невеликими студіями та поодинокими розробниками.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу.

Також підставою для розробки кваліфікаційної роботи на тему «Розробка середовища для створення програмного коду розгалуженого сюжету комп’ютерних ігор» є наказ по Національному технічному університету «Дніпровська політехніка» від 07.06. 2021р. № 317-с.

1.4. Постановка завдання

Об’єктом розробки є інструментарій для побудови розгалуженого сюжету ігор.

При аналізу предметної галузі а саме, існуючих відеоігор та пошук схожих патернів сюжету, які у майбутньому можна було б реалізувати в редакторі та бібліотеці що розробляється в роботі.

Для досягнення цієї мети в роботі ставляться та вирішуються такі завдання:

4. Зробити теоретичне обґрунтування роботи.
5. Створити бібліотеку на мові C#, яка б дозволяла реалізовувати розгалужений сюжет у грі.
6. За допомогою створеної бібліотеки створити програму (подібну до комп’ютерної гри, у якій можна було б використати розгалужений сюжет), та продемонструвати наступне:

а) проходження лінійної частини сюжету (так званих чекпоінтів);

б) вибір одного з двох варіантів дій;

г) зміни у наступних чекпоінтах залежно від вибору гравця.

1.5 Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Висунуто вимоги до самої програми:

- програма має бути простою і надійною у використанні;
- вона повинна мати можливість інтеграції в наявну систему транспорту;
- повинна мати зручний та інтуітивно зрозумілий інтерфейс;
- повинна бути невимогливою до програмно-апаратних засобів.

1.5.2. Вимоги до інформаційної безпеки

Основні вимоги до інформаційної безпеки:

– цілісність даних;

– захист від неавторизованого входу та редагування;

– конфіденційність інформації;

– доступність інформації для всіх авторизованих користувачів.

Для надійної роботи програмного забезпечення необхідно дотримуватися таких факторів:

− використання ліцензійного ПЗ;

− захист від зловмисних програм;

− шифрування даних на сервері;

− встановлення блоків безперебійного живлення.

Оскільки додаток не повинен довгостроково зберігати дані, а їх зміст не є конференційною інформацією, то специфічних вимог до інформаційної безпеки немає.

1.5.3. Вимоги до складу та параметрів технічних засобів

Додаток повинен виконуватись на мобільних пристроях під керівництвом різних операційних систем. Мінімальна діагональ екрану для пристрою повинна бути не менш ніж 5 дюйми з щільністю точок не менш ніж 480×800 dpi. для зручного використання.

Програма повинна бути адаптована для роботи й на планшетах, а також телефонах з великим розширенням і великою діагоналлю екрану. Зміст програми при цьому не повинен втратити своїх якостей читаності.

1.5.4. Вимоги до інформаційної та програмної сумісності

Додаток не є вимогливим до інформаційної та програмної сумісності та повивен підтримувати роботу на різних мобільних пристроях різних конфігурацій.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Веб-додаток графічного редактору дозволяє створювати, зберігати, відкривати, редагувати та експортувати граф епізоду. Створений скрипт на мові JS, на основі шаблону формує готовий для роботи зі створеною мною бібліотекою клас на мові C#. Було створено максимально просту бібліотеку на мові C# для управління епізодом, та обробкою різних варіантів дії з боку гравця.

Подібний проект необхідний для полегшення створення сюжетних ігор невеликими студіями та поодинокими розробниками. Більш того, сьогодні, коли доводиться сидіти вдома через складні епідеміологічні обставини, ігри є однією з розваг для частини населення. Тому під час карантину бізнес розробки комп’ютерних ігор активно розвивається і потребує нових рішень. Саме тому створений інструмент є особливо актуальним і важливим саме зараз, дозволяючи вирішити основні проблеми цього напряму в ігровій індустрії.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування математичних методів, при розробці редактору математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проектування

DOM (Document Object Model - «об'єктна модель документа») - це незалежний від платформи і мови програмний інтерфейс, що дозволяє програмам і скриптам отримати доступ до вмісту HTML-, XHTML- і XML-

документів, а також його анулювання, структуру і оформлення таких документів.

Модель DOM не накладає обмежень на структуру документа. Будь-який документ відомої структури за допомогою DOM може бути представлений у вигляді дерева вузлів, кожен вузол якого є елементом, атрибут, текстовий, графічний або будь-який інший об'єкт. Вузли зв'язані між собою відносинами «батьківський-дочірній».

Спочатку різні браузери мали власні моделі документів (DOM), несумісні з іншими. Для забезпечення взаємної і зворотної сумісності фахівці міжнародного консорціуму W3C класифікували цю модель за рівнями, для кожного з яких була створена своя специфікація. Всі ці специфікації об'єднані в загальну групу, що носить назву W3C DOM.

Ще один шаблон, що використовувався у розробленій системі – це шаблон Model-View-Controller (MVC). Він використовується для розділення обов’язків у програмі.

Model – представляє модуль, відповідальний за бізнес-логіку програми, тобто данні та їх обробку. У випадку розробленої системи можна вважати, що цей компонент шаблону представлений back-end сервером та базою даних.

View – представляє візуалізацію даних, може мати елементи для управління системою, тобто інтерфейс користувача. Він може перетворювати отриману від контролера інформацію у таку, що потім відображатиме користувачу. У розробленій системі цей компонент – front-end сервер, що містить додаток для користувачів. Коли користувач відкриває сторінку додатку, додаток отримає від back-end сервера дані, видозмінить їх та відобразить користувачу у браузері.

Controller – стоїть між відображення та моделлю. Він контролює потік даних від відображення у модель, та навпаки. Він отримує від представлення дані, що потрібно обробити, передає їх моделі для обробки, та повертає відповідну відповідь представленню.

REST (Representational State Transfer) – це архітектурний стиль, що описує шість обмежень для побудови API.

Уніфікований інтерфейс.

Описує інтерфейс взаємодії клієнта та сервера. Він спрощує та роз'єднує архітектуру, що дозволяє кожній частині бути незалежною одна від одної. Інтерфейс має слідувати наступним принципам:

Основа-ресурс: індивідуальні ресурси ідентифікуються за допомогою URI.

Маніпуляція за допомогою уявлень: якщо клієнт має уявлення про репрезентацію ресурса, цього достатньо щоб модифікувати або видалити ресурс з сервера, якщо у користувача є таке право.

Повідомлення з самоописанням: кожне повідомлення несе у собі достатньо інформації, щоб описати як повідомлення повинно бути опрацьовано.

Використання гіпермедії як двигуна стану застосунку: клієнти пересилають стан за допомогою вмісту тіла запита, параметрів запиту, заголовків та URI. Сервіси повертають стан клієнтам за допомогою тіла відповіді, HTTP статус-кодів та заголовків відповіді.

Відсутність стану.

Означає, що увесь необхідний стан, що потрібно обробити запитом, знаходиться у самому запиті.

Кешований.

Декларує, що клієнту повинні мати можливість кешувати запити. Таким чином, відповіді мають описувати, чи можуть вони бути кешовані, чи ні, щоб запобігти перевикористанню застарілих або неправильних даних у майбутніх запитах.

Клієнт-серверність.

Уніфікований інтерфейс розділяє клієнт та сервер. Таке розділення дозволяє клієнту не турбуватися, наприклад, про те, як інформація зберігається на сервері, та яким чином обробляється. Усе, що потрібно знати – це наданий сервером інтерфейс.

Багатошарова система.

Клієнт не може точно виявити, чи він взаємодіє з кінцевим сервером, чи посередником. Посередницькі сервери можуть допомогти масштабованості системи за допомогою включення балансування навантаження та надання загальних кешів.

Код на вимогу (необов'язковий).

Сервери можуть тимчасово розширити або налаштувати функціональність клієнта, передаючи йому логіку, яку він може виконувати. Приклади цього можуть включати складені компоненти, такі як аплети Java та сценарії на стороні клієнта, такі як JavaScript.

2.4. Опис використаних технологій та мов програмування

Для виконання роботи було використано мову програмування на мові JavaScript (Node.JS), а також C# для створення бібліотеки. У якості інструментів розробки був використаний збірник проектів WebPack а також фреймворк Vue. Для редагування коду було використане програмне забезпечення Visual Studio і Visual Studio Code.

Node.js - платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості:

- асинхронна модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8.

Для керування модулями використовується пакетний менеджер npm (node package manager).

C# - об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET.

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk - C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів.

2.5 Опис структури програми та алгоритмів її функціонування

Проаналізувавши велику кількість ігор, було зроблено висновок, що розгалужений сюжет в цілому можна розділити на, так звані, «модулі». Модулі – це окремі частини, з яких і побудован «епізод». Для даного проекту було виокремлено кілька таких модулів:

Чекпоінт – одиниця сюжету, демонструє якусь подію сюжету, видає логічне значення на виході (рис. 2.1).

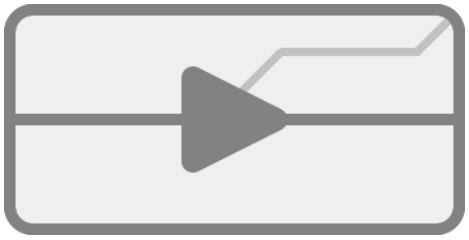

Рис. 2.1. Графічне позначення модуля «чекпоінт» в інтерфейсі редактора

Умова – логічна умова, яка зсилається на якусь подію у минулому або на логічне значення, завантажене за допомогою модуля «пошук логічного значення» (рис. 2.2).

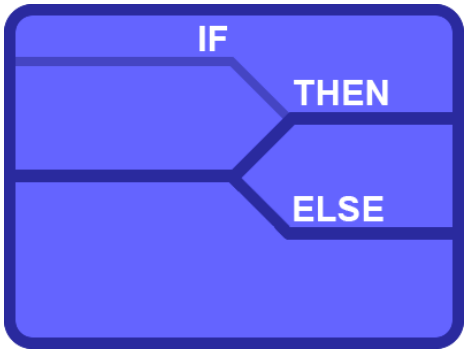

Рис. 2.2. Графічне позначення модуля «умова» в інтерфейсі редактора

Вибір – вибір гравцем між двох можливих варіантів (рис. 2.3).

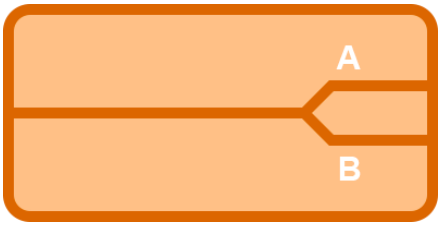

Рис. 2.3. Графічне позначення модуля «вибір» в інтерфейсі редактора

Пошук логічного значення – дістає логічне значення з коду гри, через виклик спеціальної функції (рис.2.4).

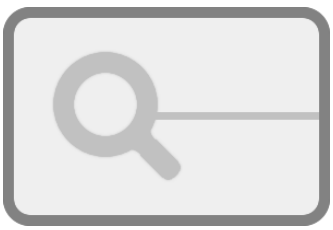

Рис. 2.4. Графічне позначення модуля «пошук логічного значення» в інтерфейсі редактора

Точка входу – базовий модуль, з якого починається сюжет всього епізоду.

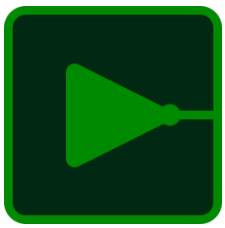

Рис. 2.5. Графічне позначення модуля «точка входу» в інтерфейсі редактора

Окрім модулів для роботи потрібні типи їх зв’язків, якими ці модулі будуть поєднуватись. На даний момент було реалізовано 2 типи зв’язків: сюжетну лінію та логічний зв’язок. Сюжетна лінія визначає послідовність подій у грі, в той час, як логічна дозволяє модулям звертатись до логічних станів одне одного.

Маючі окремі модулі і зв’язки між ними ми отримуємо ациклічний граф, тобто дерево, яке починається з модуля «Точка входу» і закінчується останнім чекпоінтом (рис. 2.6).

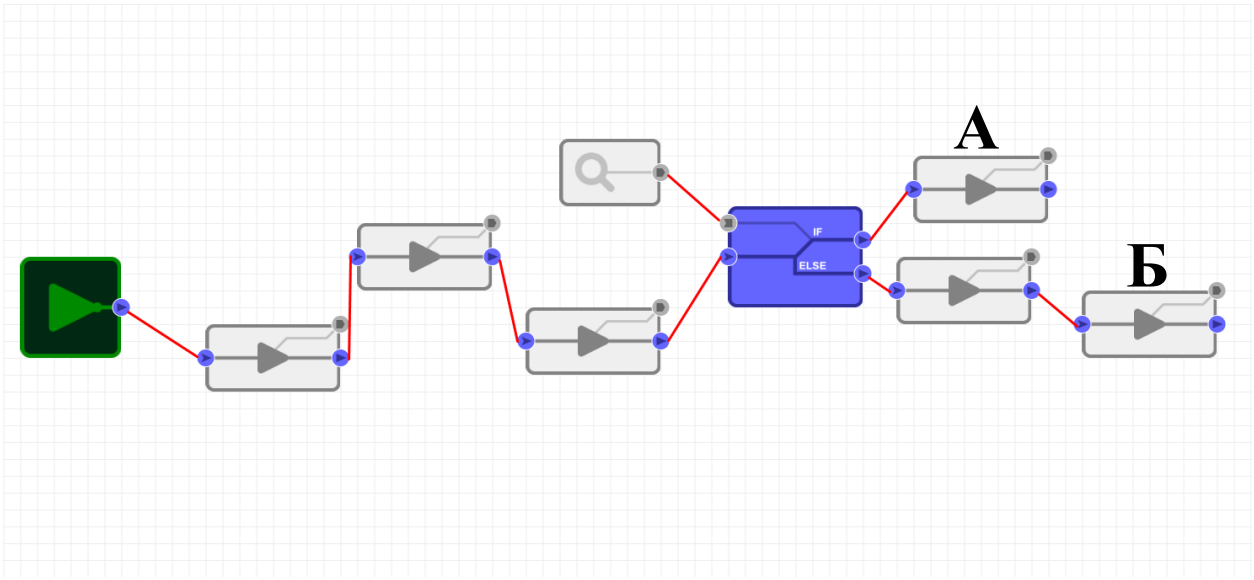

Рис. 2.6. Приклад сюжетного графу, який має два кінці «А» та «Б».

Тут в справу вступає один важливий нюанс. На зображенні видно, що на вхід модуля «умова» подаються логічні дані з модуля «пошук логічного значення», що є некоректним для деревоподібної структури. Насправді ж ніякі вхідні дані на модуль «умова» не надходять, а «пошук логічного значення» є дочірнім для модуля «умова».

На практиці для зберігання такої схеми легше за все використовувати XML або JSON розмітку, тому зображена вище схема може бути записана у наступному вигляді (приклад запису сюжетного графу у форматі JSON):

```
{
 “type”: “start”,
 “child”: {
  “type”: “checkpoint”,
  “child”: {
   “type”: “checkpoint”,
   “child”: {
    “type”: “checkpoint”,
    “child”: {
     “type”: “condition”,
     “logic”: {
      “type”: “logic_search”, “search”: “IS_SOMETHING_TRUE”
     },
     “results”:                                                    {
“true”: {
       “type”: “checkpoint”
      },
“false”: {
       “type”: “checkpoint”,
       “child”: {
        “type”: “checkpoint”
       }
      }
}
    }
   }
  }
 }
```

```
 }
}
```

Реалізація графічного редактора. Першим кроком у написанні графічного редактора стало вивчення роботи з елементом «canvas» у HTML.

Для початку нам потрібно ініціювати робочу поверхню канвасу, та створити функцію, яка буде автоматично підлаштовувати його під розмір контейнера, в якому цей канвас знаходиться.

```
const ctx = canvas.getContext('2d');
const updateSize = ()=>{
   canvas.height = cframe.offsetHeight;
   canvas.width = cframe.offsetWidth;
};
window.onresize = updateSize;
```

Далі необхідно створити так званий «цикл малювання», в якому буде знаходитись код, що малює актуальне зображення.

```
const loop = ()=>{
   fps = Math.round(1000/(performance.now() - lastFrameTime));
   lastFrameTime = performance.now();
   draw();
   window.requestAnimationFrame(loop);
};
window.requestAnimationFrame(loop);
```

Код з функції циклу (loop), вказаний вище, оновлює показник кількості кадрів, підмальованих за останню секунду, а також часу, за який був відмальований останній кадр, після чого викликає функцію малювання.

Сама функція малювання виглядає наступним чином:

```
const draw = ()=>{
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawGrid();
    drawBlocks();
};
```

Як можна бачити, вона викликає функцію drawGrid, що відповідає за малювання клітинчатого фону, а також drawBlocks, що відповідає за малювання самих модулів.

Реалізація роботи з модулями у редакторі:

```
const types = {
    "entry": {
        default: [0, 0],
        menu: {
            name: "Entry point",
            description: "Start of the episode"
        },
        width: 6,
        height: 6,
        texture: getImage("<base64>"),
        points:
            {
                type: "main-out",
                position: {
                    x: 6,
                    y: 3
                }
            }
        ]
```

```
},
"checkpoint":{
    menu: {
        icon: "fas fa-flag-checkered",
        name: "Checkpoint",
        description: "Linear checkpoint"
    },
    width: 8,
    height: 4,
    texture: getImage("<base64>"),
    points:
        {
            type: "main-out",
            position: {
                x: 8,
                y: 2
            }
        },
        {
            type: "main-in",
            position: {
                x: 0,
                y: 2
            }
        },
        {
            type: "logical-out",
            position: {
                x: 8,
                y: 0
```

```
            }
        }
    }
  },
};
```

Вище приведено частину коду, який описує та ініціює всі можливі типи модулів, а також описує точки з’єднання з іншими модулями.

Як можна побачити, кожен модуль має власну текстуру для виведення на екран, висоту та ширину у клітинках, та точки з’єднання. Модуль «entry» («точка входу») має додатковий параметр default, який робить одразу дві речі: вказує, що цей модуль є модулем за замовчанням (тобто модулем, який є в епізоді початково і який не можна додати до епізоду, видалити, або перемістити), а також вказує на його положення в системі координат у робочому просторі редактора.

Під час роботи з епізодом стани всіх об’єктів записуються в локальне сховище браузера для запобігання втрати при закритті вкладки з проектом.

Тепер, коли створене все для розробки сюжету, час подумати про розробку ігор. Я створив доволі простий алгоритм експорту.

Для початку відбувається ініціалізація основної структури класу:

```
using StoryFine;
namespace YourGame
{
   class YourEpisode : SFEpisode
   {
      }
}
```

Назву простору імен (namespace) та класу, користувач задає при активації функції експорту. Створений клас наслідує базовий клас «SFEpisode» (SF – скорочення від назви створюваного мною пакету інструментів – «StoryFine»).

Далі клас наповнюється полями, відповідними кожному модулю епізода.

2.6.Обґрунтування та організація вхідних та вихідних даних програми

Щоб полегшити процес інтеграції сюжету в саму гру, було вирішено створити бібліотеку на доволі популярній мові C#.

В першу чергу потрібно зрозуміти, які саме задачі повинна виконувати бібліотека. В першу чергу вона повинна бути базою для експортованого сюжету. Аби зробити процес розробки гри з використанням бібліотеки простішим, було вирішено реалізувати експорт сюжету у готовий клас на мові C# (детальніше читайте в наступному пункті теоретичної частини).

Для реалізації такого підходу, аби не заносити всю програмну логіку у наш генеруємий клас, було вирішено винести всю логіку в окрему бібліотеку. Тож бібліотека є в першу чергу основою і API для роботи з експортованим сюжетом.

Аби полегшити процес завантаження сюжету у проект гри, було вирішено експортувати його як повноцінний клас на мові C#. Це дозволить розробнику гри більше акцентуватись на роботі з кодом, а не на вивченні API, оскільки всі елементи сюжету будуть зберігатись як поля класу, а не елементи якого-небудь масиву.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Редактор розроблено на ноутбуці Lenovo IdeaPad 320-15IKB з такими характеристиками: екран 15.6" (1920x1080) Full HD, матовий / Intel Core i5-7200U (2.5 - 3.1 ГГц) / RAM 8 ГБ / HDD 1 ТБ + SSD 128 ГБ / nVidia GeForce GT

940MX, 2 ГБ / LAN / Wi-Fi / Bluetooth / веб-камера.

2.7.2. Використані програмні засоби

Для виконання роботи було використано мову програмування на мові JavaScript (Node.JS), а також C# для створення бібліотеки. У якості інструментів розробки був використаний збірник проектів WebPack а також фреймворк Vue. Для редагування коду було використане програмне забезпечення Visual Studio і Visual Studio Code.

2.7.3. Виклик та завантаження програми

Для завантаження сторінки необхідно лише веб-браузер.

Програма не є вимогливою до виклику та завантаженню і може завантажуватись на ПК, ноутбуках, різних типів та конфігурацій під управлінням різних ОС.

Для нормального функціонування данного застосунку необхідно, щоб персональний комп’ютер, на якому, буде функціонувати система відповідала наступним вимогам:

- процесор класу Intel Pantium з тактовою частотою не менш 2.4 ГГц та двума ядрами;
- доступ до мережі Internet;
- не менше 8 GB оперативної пам'яті;
- 500 GB вільного місця на жорсткому диску;
- клавіатура;
- маніпулятор "миша".

Наведені вище технічні характеристики ПК є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний

застосунок буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

2.7.4. Опис інтерфейсу користувача

Глолвна форма редактору сюжету наведено на рис. 2.7.

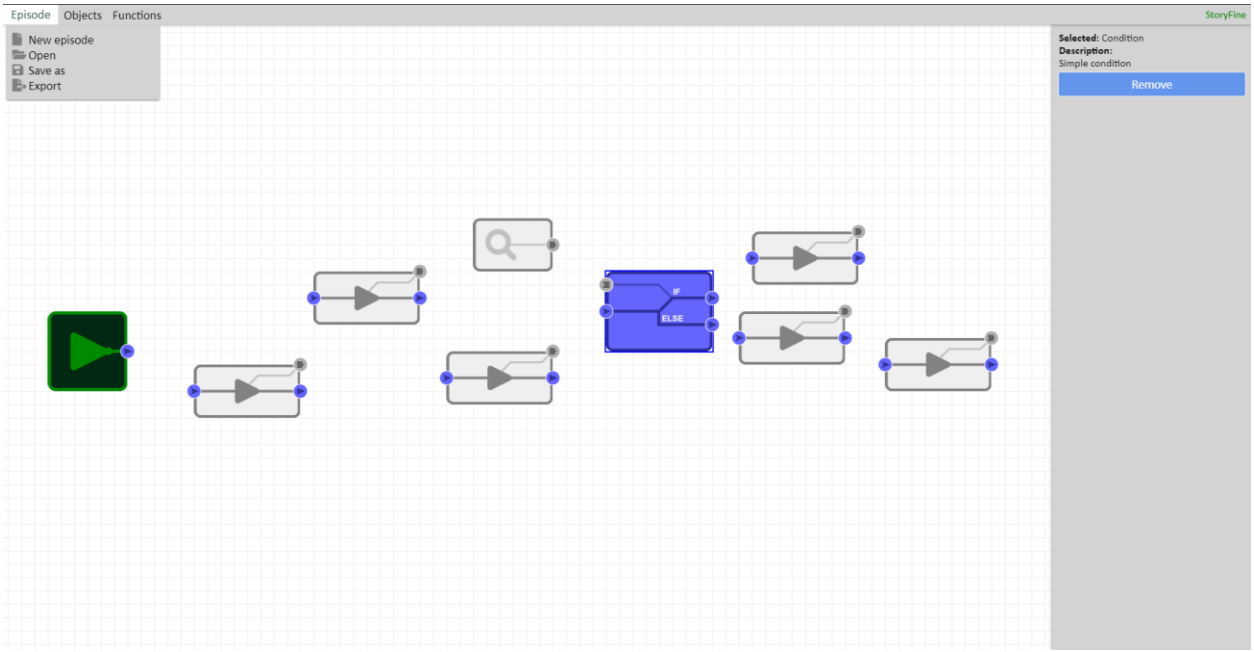

Рис. 2.7. Головна форма редактору «StoryFine Editor»

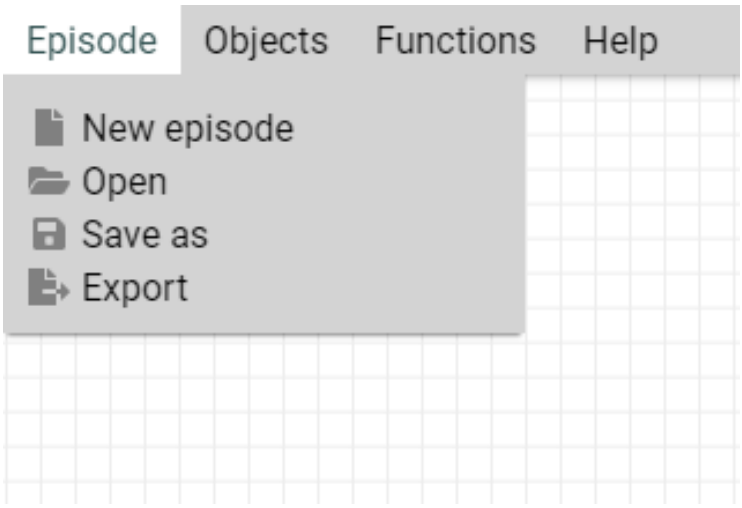

Рис. 2.8. Вкладинка меню роботи з епізодами

Меню епізодів дозволяє нам робити наступне:

– Створити новий епізод - повністю порожнє робоче простір, що містить тільки базовий модуль входу.

– Відкрийте епізод StoryFine з файлу .sfep.

– Збережіть випуск StoryFine в файл .sfep.

– Експорт епізоду в клас на C #.

Меню об'єкта дозволяє додати в проект кілька модулів (рис. 2.8).

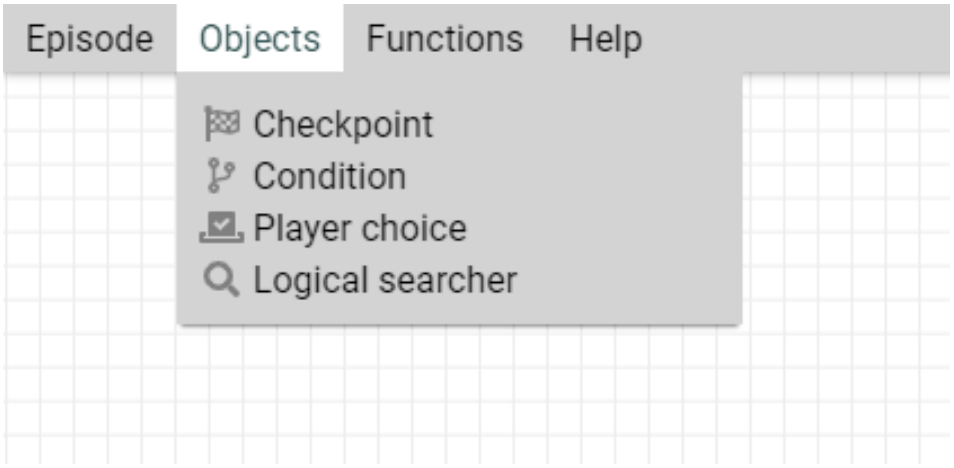

Рис. 2.8. Вкладинка меню роботи з об’єктами

Контрольна точка - це базовий модуль з основним входом, основним виходом і логічним виходом.

Умова являє собою модуль, що містить логіку і основні входи, а також дві основні виходи, що перемикаються в залежності від значення на логічний вхід.

Вибір гравця дозволяє вам надати гравцеві вибір з двох варіантів. Цей модуль складається з основного входу і двох основних виходів.

Логічний шукач дозволяє отримати логічні (булеві) значення з основного ігрового коду, викликавши зумовлений делегат. Цей модуль містить тільки один логічний вихід.

Є ще один модуль, якого немає в цьому списку - модуль Entry. Він містить тільки головний вихід і є відправною точкою сюжету (рис. 2.9).

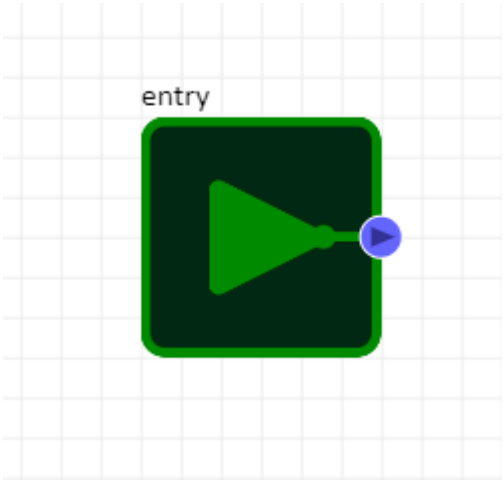

Рис. 2.9. Відправна точка сюжету

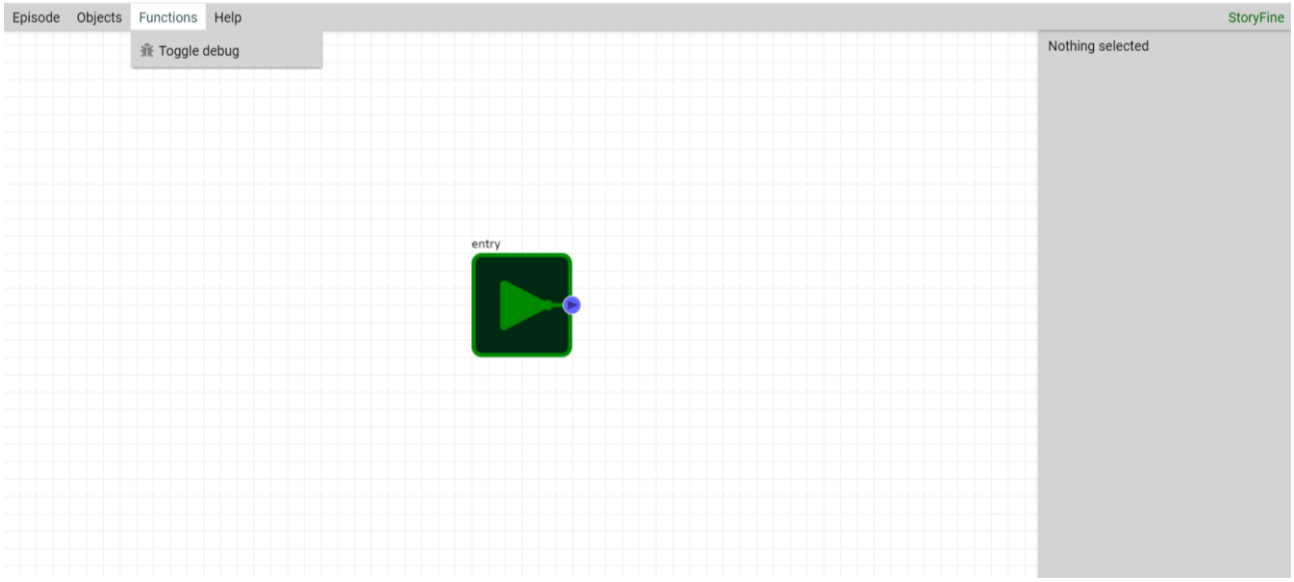

Рис. 2.10. Меню відлагодження

Завнтаження інструкції користувача може бути виконано через вибір відповідного пункту меню (рис.2.11).

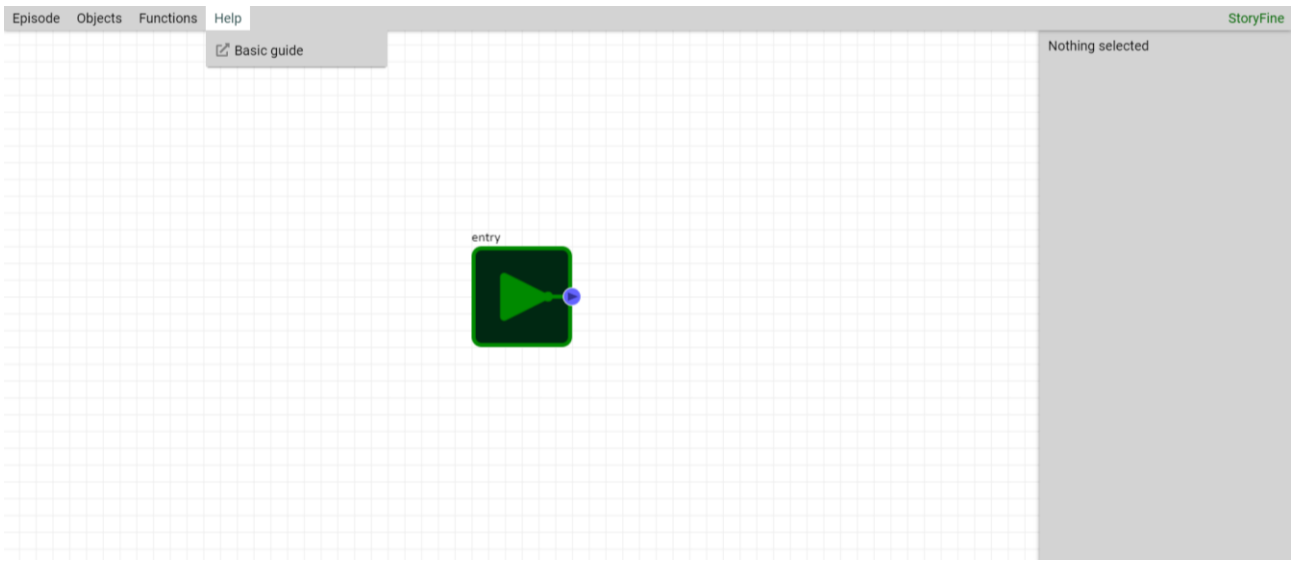

Рис. 2.11. Меню допомоги

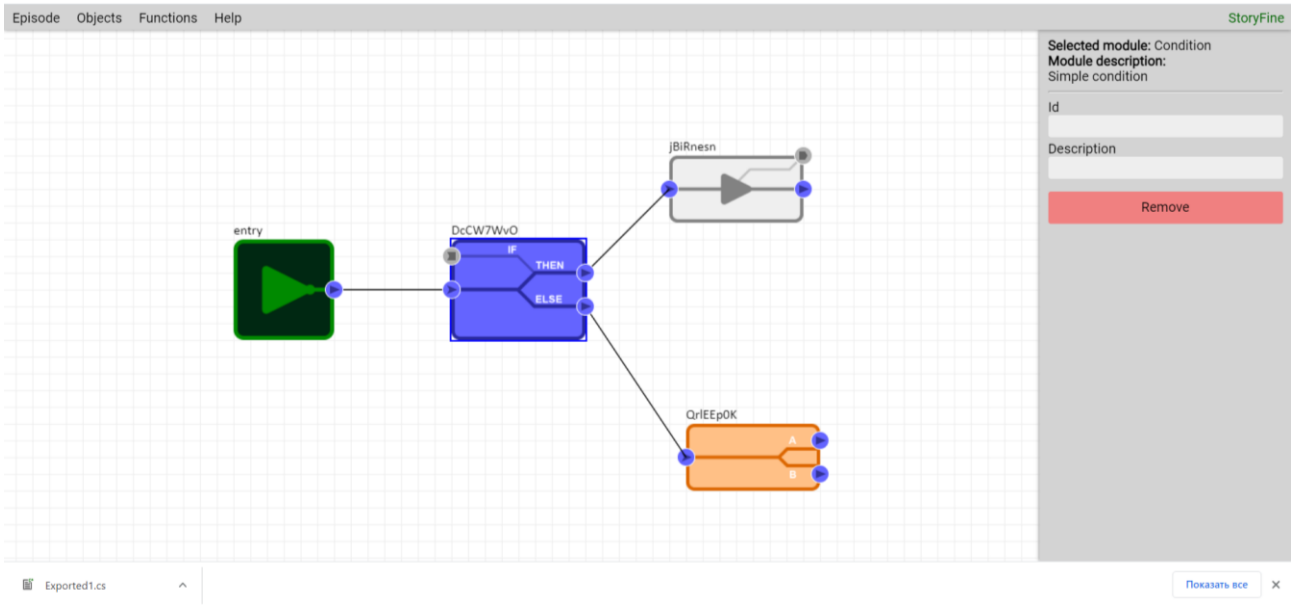

Рис. 2.12. Робоча область редактору

В робочій області редактора користувач за допомогою миші може перетягувати елементи, при цьому вся зв’язки зберігаються(рис.2.13). В правій частині редактора відображаються властивості та підказки щодо обраного елементу.

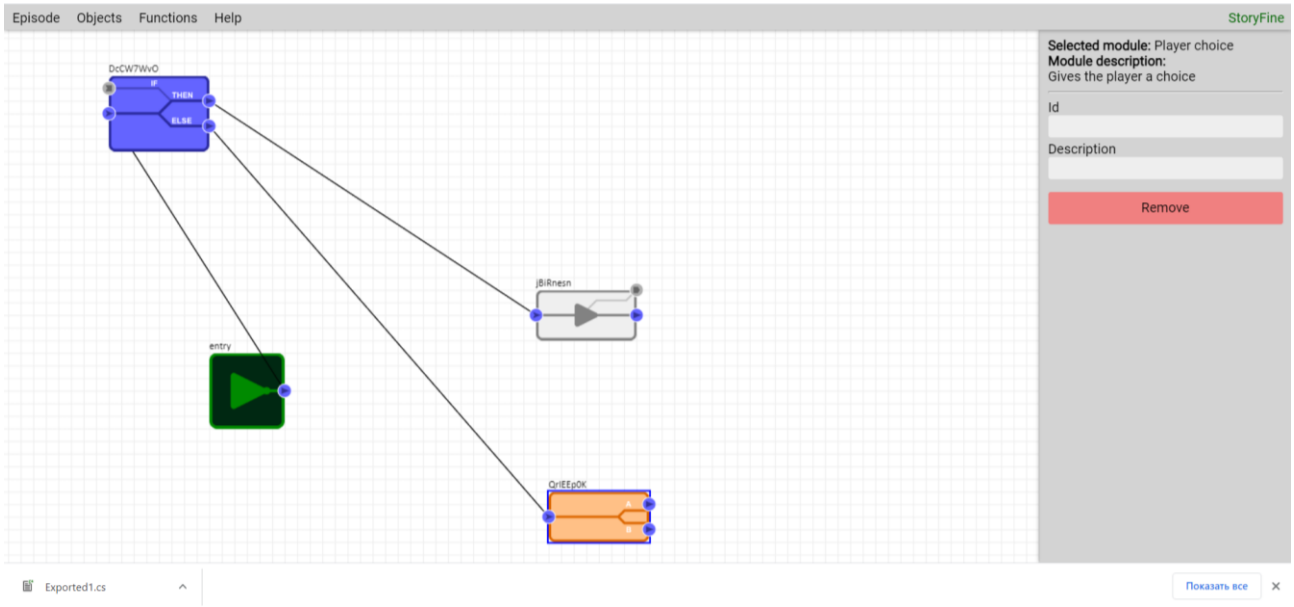

Рис. 2.13. Редагування сюжету

Після створення сюжету, розробник може експортувати свою розробку в файл .cs, тим самим автоматично створити відповідний клас.

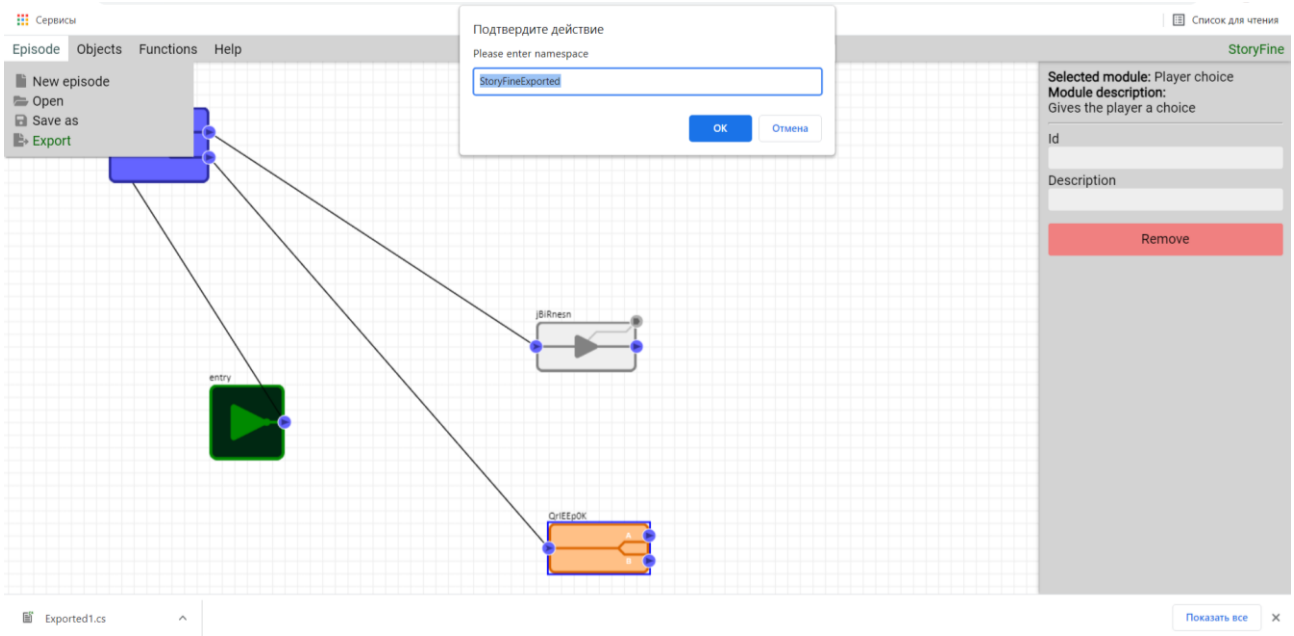

Рис.2.14. Експорт сюжету в клас C#

Експортувати можна навіть порожній сюжет (рис. 2.15).

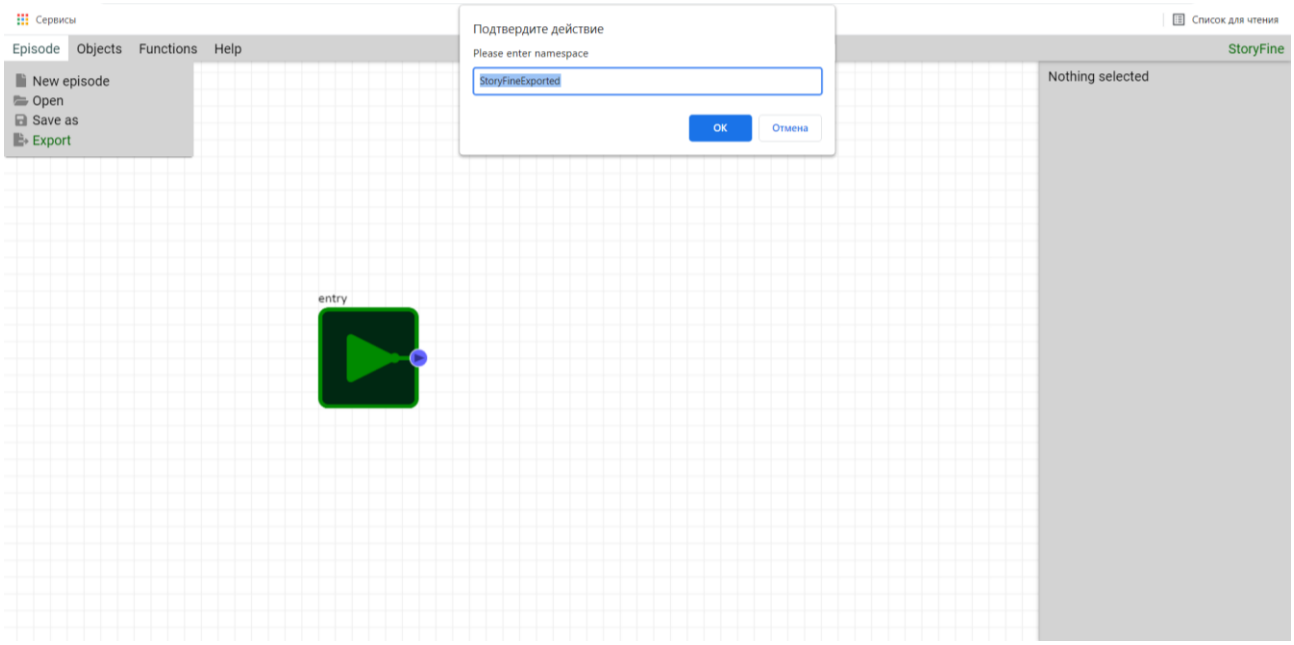

Рис. 2.15. Експорт порожнього сюжету в клас C#

Після створення та експорту сюжету в файл, його можна використати в подальшому програмуванні в інших редакторах (рис. 2.16).

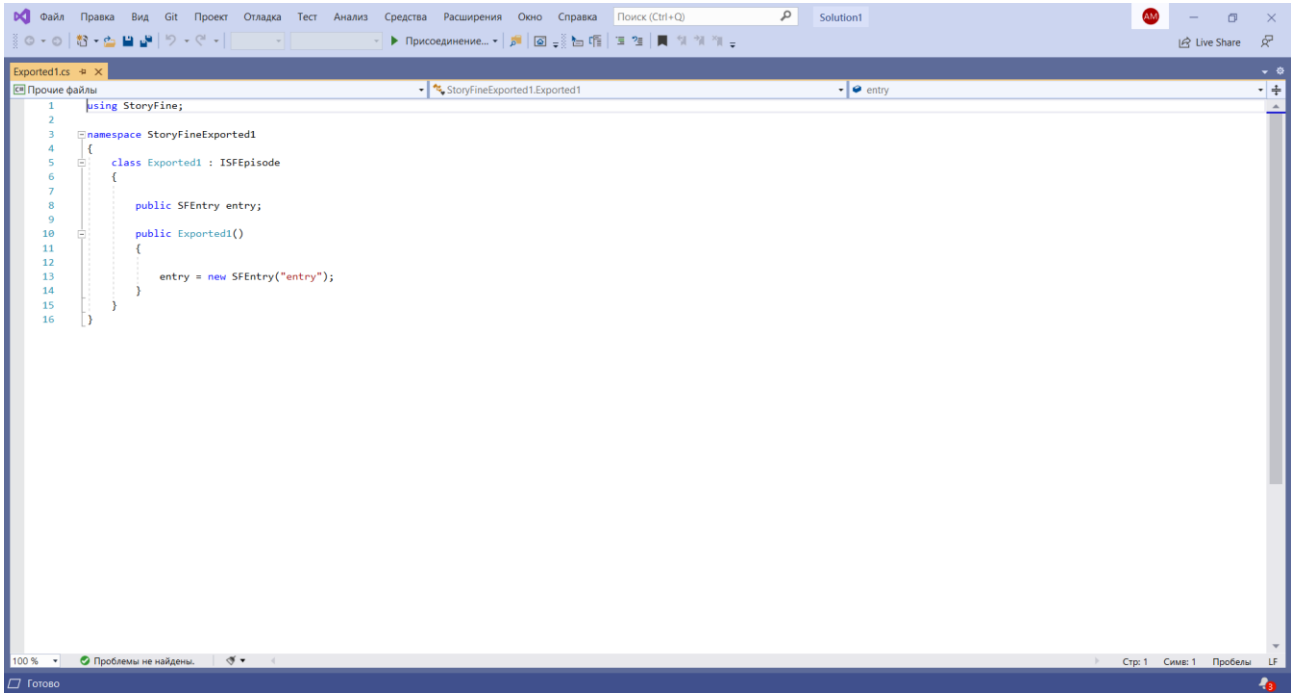

Рис. 2.16. Завантажений файл в середовище MS Visual Studio 2019

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

1) Передбачувана кількість операторів: 1200;
2) Коефіцієнт складності програми: 1,4;
3) Коефіцієнт корекції програми в ході розробки: 0,08;
4) Годинна заробітна плата програміста, грн./год.: 125 грн./год.;
5) Вартість машино-години, грн./год.: 15 грн./год.;

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д \text{, людино-годин} \quad (3.1)$$

де $t_о$- витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$- витрати праці на дослідження алгоритму рішення задачі;

$t_а$- витрати праці на розробку блок-схеми алгоритму;

$t_п$- витрати праці на програмування по готовій блок-схемі;

$t_{отл}$- витрати праці на налагодження програми на ЕОМ;

$t_д$- витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.2)$$

де q - передбачуване число операторів (1200);

C - коефіцієнт складності програми (1,4);

p - коефіцієнт кореляції програми в ході її розробки (0,07).

Маємо:

$$Q = 1200 * 1,4 * (1 + 0,08) = 1814,4 \approx 1814 \text{ операторів}$$

Витрати праці на вивчення опису задачі $t_и$визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_и = \frac{Q * B}{(75..85) * k}, \text{ людино-годин} \quad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності (0,8).

Маємо:

$$t_и = \frac{1814 * 1,2}{75 * 0,8} = \frac{2177,28}{60} = 36,29 \text{ , людино-годин} \quad (3.4)$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_а = \frac{Q}{(20..25) * k}, \text{ людино-годин}$$

Маємо:

$$t_а = \frac{1814}{20 * 0,8} = 113,38 \text{ , людино-годин}$$

Витрати на складання програми по готовій блок-схемі:

$$t_п = \frac{Q}{(20..25) * k}, \text{ людино-годин} \quad (3.5)$$

Маємо:

$$t_п = \frac{1814}{20 * 0,8} = 113,38 \text{ , людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4..5) * k}, \text{ людино-годин} \quad (3.6)$$

Маємо:

$$t_{отл} = \frac{1814}{5 * 0,8} = 453,5 \text{ , людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин} \quad (3.7)$$

де $t_{др}$- трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) * k}, \text{ людино-годин} \quad (3.8)$$

$t_{до}$- трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 * t_{др}, \text{ людино-годин} \quad (3.9)$$

Маємо:

$$t_{др} = \frac{1814}{15 * 0,8} = 151,16 \text{ , людино-годин}$$

$$t_{до} = 0,75 * 151,16 = 113,375 \text{ , людино-годин}$$

$$t_д = 151,16 + 113,375 = 264,535 \text{ , людино-годин}$$

Маємо наступну трудомісткість розробки ПЗ:

$$t= 50+36,29+113,38+113,38+453,5+264,535 \approx 1\ 031 \text{ , людино-годин,}$$

3.2. Розрахунок на створення програмного забезпечення

Витрати на створення ПЗ $К_{по}$включають витрати на заробітну плату виконавця програми $З_{зп}$і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{по} = З_{зп} + З_{мв}, \text{ грн} \quad (3.10)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * С_{пр}, \text{ грн} \quad (3.11)$$

де: t- загальна трудомісткість, людино-годин (1 031);

$С_{пр}$- середня годинна заробітна плата програміста, грн/година (125)

Маємо:

$$З_{зп} = 1\ 031 * 125 = 128\ 875, \text{ грн}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * С_{мч}, грн \quad (3.12)$$

де $t_{отл}$- трудомісткість налагодження програми на ЕОМ, год (453,5).

$С_{мч}$- вартість машино-години ЕОМ, грн/год (15).

Маємо:

$З_{мв} = 453,5 * 15 = 6\ 802,5$, грн

Маємо наступні витрати на створення програмного продукту:

$К_{по} = 128\ 875 + 6\ 802,5 = 135\ 677,5$, грн

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_k * F_p}, міс \quad (3.13)$$

де B_k- число виконавців (1);

F_p- місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

$$Т = \frac{1031}{1*176} = 5,85 \approx 6, міс$$

Висновок: в кваліфікаційній роботі було розраховано трудомісткість розробки програмного забезпечення, очікуваний період створення програмного забезпечення та витрати на створення програмного забезпечення. Трудомісткість склала 1031 людино-годин, очікуваний період створення склав приблизно 6 місяців, витрати на створення програмного продукту склали 135 677,5 грн., з яких 128 875 грн. – витрати на заробітну плату виконавців, а 6 802,5 грн. – вартість машинного часу.

ВИСНОВКИ

Помітно, що запропонований у роботі підхід до створення сюжетів є досить практичним і може знайти реальне застосування в індустрії. Використання стандартизованого підходу до сюжетів спростить життя як сценаристам, так і розробникам.

Що було виконано:

- теоретичне обґрунтування необхідності такої бібліотеки;
- створення концепції модулів;
- створено редактор та бібліотеку на мові C#;
- створений демонстраційний додаток.

Отже, створена бібліотека може задовільнити частину потреб сучасної індустрії розробки ігор.

Для виконання роботи було використано мову програмування на мові JavaScript (Node.JS), а також C# для створення бібліотеки. У якості інструментів розробки був використаний збірник проектів WebPack а також фреймворк Vue. Для редагування коду було використане програмне забезпечення Visual Studio і Visual Studio Code.

Структура роботи:

1. Веб-редактор епізодів.
    a. Графічний редактор.
    b. Процесор графів.
    c. Скрипт формування C#-класу для роботи з бібліотекою на основі даних графа.
2. Бібліотека на мові C#.
    a. Класи епізоду та його компонентів.
    b. API для зручної роботи зі сформованим у C#-клас епізодом.

Було створено веб-додаток графічного редактору, який дозволяє створювати, зберігати, відкривати, редагувати та експортувати граф епізоду. Було створено скрипт на мові JS, який на основі шаблону формує готовий для

роботи зі створеною мною бібліотекою клас на мові C#. Було створено максимально просту бібліотеку на мові C# для управління епізодом, та обробкою різних варіантів дії з боку гравця.

Подібний проект був просто необхідним для полегшення створення сюжетних ігор невеликими студіями та поодинокими розробниками. Більш того, сьогодні, коли доводиться сидіти вдома через складні епідеміологічні обставини, ігри є основною розвагою для значної частини населення. Тому під час карантину бізнес розробки комп'ютерних ігор активно розвивається і потребує нових рішень. Саме тому створений інструмент є особливо актуальним і важливим саме зараз, дозволяючи вирішити основні проблеми цього напряму в ігровій індустрії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Альфред В. Ахо. Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 689 с.
2. Гарнаев Андрей. WEB-программирование на Java и JavaScript / Андрей Гарнаев, Сергей Гарнаев. - Москва: СПб. [и др.] : Питер, 2017. - 718 с.
3. Гонсалвес Энтони. Изучаем Java EE 7 / Энтони Гонсалвес. - М.: Питер, 2016. - 640 с.
4. Гупта Арун. Java EE 7. Основы / Арун Гупта. - М.: Вильямс, 2014. - 336 с.
5. Монахов В. Язык программирования Java и среда NetBeans В. Монахов. - М.: БХВ-Петербург, 2012. - 720 с.
6. Савитч Уолтер. Язык Java. Курс программирования / Уолтер Савитч. - М.: Вильямс, 2015. - 928 с.
7. Хабибуллин Ильдар. Самоучитель Java / Ильдар Хабибуллин. - М.: БХВ-Петербург, 2014. - 768 с.
8. Шилдт Герберт. Java 8. Руководство для начинающих / Герберт Шилдт. - М.: Вильямс, 2015. - 720 с.
9. Эккель Брюс. Философия Java / Брюс Эккель. - М.: Питер, 2016. - 809 с.
10. Joshua Bloch. Effective Java. Second Edition, 2008. - 309 p.
11. Yakov Fain. Java Programming 24-Hour Trainer, Second Edition, 2015. - 404 p.
12. John Carnell. Spring Microservices in Action, 2017. - 532 p.
13. Chris Richardson. Microservices Patterns: With Examples in Java, 2018. - 211 p.
14. Mark H. Massé. REST API Design Rulebook, 2011. - 435 p.
15. Dinesh Rajput. Spring 5 Design Patterns: Master Efficient Application Development with Patterns Such as Proxy, Singleton, the Template Method, and More, 2017. - 329 p.

16. Кей С. Хорстманн. Java. Бібліотека професіонала. Основи. 11-є видання, 2020. - 727 с.

17. Етан Браун. Вивчаємо JavaScript. Керівництво по створенню сучасних веб-сайтів. 3-є видання, 2020. - 435 с.

18. Агуров Павел. C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.

19. Альфред В. Ахо. Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.

20. Бишоп Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.

21. Вагнер Билл. C# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 с.

22. Зиборов Виктор. Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.

23. Ишкова Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.

24. Лотка Рокфорд. C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

25. Мак-Дональд Мэтью. Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

26. Подбельский В. В. Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

27. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 с.

28. Троелсен Эндрю. Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 с.

ДОДАТОК А

КОД ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SFDemo
{
   static class Program
   {
      /// <summary>
      /// Главная точка входа для приложения.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}



using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using StoryFine;

namespace SFDemo
{
    public partial class Form1 : Form
    {
        GameEpisode ge;
        ISFModule m;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ge = new GameEpisode(
                delegate (SFSearcher s, ISFEpisode ep) {
                    switch (s.Id)
                    {
                        case "IS_CURRENT_TIME_EVEN":
                            return DateTime.Now.Second % 2 == 0;
                    }
                    return false;
```

```
        }
    );
    m = ge.entry.Next;
    tabControl1.SelectedIndex = 1;
}

private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToString("hh:mm:ss");
}

private void button2_Click(object sender, EventArgs e)
{
    m = m.Next.Next;
    if(m.Id == "NOT_EVEN_FINAL") tabControl1.SelectedIndex = 2;
    else tabControl1.SelectedIndex = 3;
}

private void button3_Click(object sender, EventArgs e)
{
    ((SFChoice)m).Choose(SFChoice.ChoiceResult.A);
    Continue();
}

private void button4_Click(object sender, EventArgs e)
{
    ((SFChoice)m).Choose(SFChoice.ChoiceResult.B);
    Continue();
}
```

```
        private void Continue()
        {
            m = m.Next;
            if(m.Id == "EVEN_FINAL")
            {
                tabControl1.SelectedIndex = 4;
            }
            else
            {
                tabControl1.SelectedIndex = 5;
            }
        }

        private void button5_Click(object sender, EventArgs e)
        {
            m = m.Next;
            tabControl1.SelectedIndex = 6;
        }
    }
}



namespace SFDemo
{
    partial class Form1
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
```

```
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.tabControl1 = new System.Windows.Forms.TabControl();
            this.tabPage1 = new System.Windows.Forms.TabPage();
            this.button1 = new System.Windows.Forms.Button();
            this.tabPage2 = new System.Windows.Forms.TabPage();
            this.button2 = new System.Windows.Forms.Button();
```

```
this.label1 = new System.Windows.Forms.Label();
this.tabPage3 = new System.Windows.Forms.TabPage();
this.label2 = new System.Windows.Forms.Label();
this.tabPage4 = new System.Windows.Forms.TabPage();
this.tabPage5 = new System.Windows.Forms.TabPage();
this.tabPage6 = new System.Windows.Forms.TabPage();
this.tabPage7 = new System.Windows.Forms.TabPage();
this.timer1 = new System.Windows.Forms.Timer(this.components);
this.label3 = new System.Windows.Forms.Label();
this.button3 = new System.Windows.Forms.Button();
this.button4 = new System.Windows.Forms.Button();
this.label4 = new System.Windows.Forms.Label();
this.label5 = new System.Windows.Forms.Label();
this.button5 = new System.Windows.Forms.Button();
this.tabControl1.SuspendLayout();
this.tabPage1.SuspendLayout();
this.tabPage2.SuspendLayout();
this.tabPage3.SuspendLayout();
this.tabPage4.SuspendLayout();
this.tabPage5.SuspendLayout();
this.tabPage6.SuspendLayout();
this.tabPage7.SuspendLayout();
this.SuspendLayout();
//
// tabControl1
//
this.tabControl1.Controls.Add(this.tabPage1);
this.tabControl1.Controls.Add(this.tabPage2);
this.tabControl1.Controls.Add(this.tabPage3);
this.tabControl1.Controls.Add(this.tabPage4);
```

```
this.tabControl1.Controls.Add(this.tabPage5);
this.tabControl1.Controls.Add(this.tabPage6);
this.tabControl1.Controls.Add(this.tabPage7);
this.tabControl1.Location = new System.Drawing.Point(-4, -22);
this.tabControl1.Name = "tabControl1";
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(808, 476);
this.tabControl1.TabIndex = 0;
//
// tabPage1
//
this.tabPage1.Controls.Add(this.button1);
this.tabPage1.Location = new System.Drawing.Point(4, 22);
this.tabPage1.Name = "tabPage1";
this.tabPage1.Padding = new System.Windows.Forms.Padding(3);
this.tabPage1.Size = new System.Drawing.Size(800, 450);
this.tabPage1.TabIndex = 0;
this.tabPage1.Text = "tabPage1";
this.tabPage1.UseVisualStyleBackColor = true;
//
// button1
//
this.button1.Location = new System.Drawing.Point(6, 200);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(788, 42);
this.button1.TabIndex = 0;
this.button1.Text = "Розпочати";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
```

```
            // tabPage2
            //
            this.tabPage2.Controls.Add(this.button2);
            this.tabPage2.Controls.Add(this.label1);
            this.tabPage2.Location = new System.Drawing.Point(4, 22);
            this.tabPage2.Name = "tabPage2";
            this.tabPage2.Padding = new System.Windows.Forms.Padding(3);
            this.tabPage2.Size = new System.Drawing.Size(800, 450);
            this.tabPage2.TabIndex = 1;
            this.tabPage2.Text = "tabPage2";
            this.tabPage2.UseVisualStyleBackColor = true;
            //
            // button2
            //
            this.button2.Location = new System.Drawing.Point(6, 203);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(788, 42);
            this.button2.TabIndex = 1;
            this.button2.Text = "Перший крок";
            this.button2.UseVisualStyleBackColor = true;
            this.button2.Click += new System.EventHandler(this.button2_Click);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 24F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label1.Location = new System.Drawing.Point(6, 9);
            this.label1.Name = "label1";
```

```
            this.label1.Size = new System.Drawing.Size(106, 37);
            this.label1.TabIndex = 0;
            this.label1.Text = "label1";
            //
            // tabPage3
            //
            this.tabPage3.Controls.Add(this.label2);
            this.tabPage3.Location = new System.Drawing.Point(4, 22);
            this.tabPage3.Name = "tabPage3";
            this.tabPage3.Size = new System.Drawing.Size(800, 450);
            this.tabPage3.TabIndex = 2;
            this.tabPage3.Text = "tabPage3";
            this.tabPage3.UseVisualStyleBackColor = true;
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 21.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label2.Location = new System.Drawing.Point(3, 169);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(794, 91);
            this.label2.TabIndex = 0;
            this.label2.Text = "Кількість секунд непарна. Досягнено фінальний
модуль";
            this.label2.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
            //
            // tabPage4
            //
            this.tabPage4.Controls.Add(this.button4);
```

```
this.tabPage4.Controls.Add(this.button3);
this.tabPage4.Controls.Add(this.label3);
this.tabPage4.Location = new System.Drawing.Point(4, 22);
this.tabPage4.Name = "tabPage4";
this.tabPage4.Size = new System.Drawing.Size(800, 450);
this.tabPage4.TabIndex = 3;
this.tabPage4.Text = "tabPage4";
this.tabPage4.UseVisualStyleBackColor = true;
//
// tabPage5
//
this.tabPage5.Controls.Add(this.label4);
this.tabPage5.Location = new System.Drawing.Point(4, 22);
this.tabPage5.Name = "tabPage5";
this.tabPage5.Size = new System.Drawing.Size(800, 450);
this.tabPage5.TabIndex = 4;
this.tabPage5.Text = "tabPage5";
this.tabPage5.UseVisualStyleBackColor = true;
//
// tabPage6
//
this.tabPage6.Controls.Add(this.button5);
this.tabPage6.Location = new System.Drawing.Point(4, 22);
this.tabPage6.Name = "tabPage6";
this.tabPage6.Size = new System.Drawing.Size(800, 450);
this.tabPage6.TabIndex = 5;
this.tabPage6.Text = "tabPage6";
this.tabPage6.UseVisualStyleBackColor = true;
//
// tabPage7
```

```
            //
            this.tabPage7.Controls.Add(this.label5);
            this.tabPage7.Location = new System.Drawing.Point(4, 22);
            this.tabPage7.Name = "tabPage7";
            this.tabPage7.Size = new System.Drawing.Size(800, 450);
            this.tabPage7.TabIndex = 6;
            this.tabPage7.Text = "tabPage7";
            this.tabPage7.UseVisualStyleBackColor = true;
            //
            // timer1
            //
            this.timer1.Enabled = true;
            this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Microsoft Sans Serif", 21.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label3.Location = new System.Drawing.Point(3, 141);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(794, 54);
            this.label3.TabIndex = 1;
            this.label3.Text = "Зробіть вибір";
            this.label3.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
            //
            // button3
            //
```

```
            this.button3.Font = new System.Drawing.Font("Microsoft Sans Serif",
20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.button3.Location = new System.Drawing.Point(12, 198);
            this.button3.Name = "button3";
            this.button3.Size = new System.Drawing.Size(387, 55);
            this.button3.TabIndex = 2;
            this.button3.Text = "Варіант 1";
            this.button3.UseVisualStyleBackColor = true;
            this.button3.Click += new System.EventHandler(this.button3_Click);
            //
            // button4
            //
            this.button4.Font = new System.Drawing.Font("Microsoft Sans Serif",
20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.button4.Location = new System.Drawing.Point(401, 198);
            this.button4.Name = "button4";
            this.button4.Size = new System.Drawing.Size(387, 55);
            this.button4.TabIndex = 2;
            this.button4.Text = "Варіант 2";
            this.button4.UseVisualStyleBackColor = true;
            this.button4.Click += new System.EventHandler(this.button4_Click);
            //
            // label4
            //
            this.label4.Font = new System.Drawing.Font("Microsoft Sans Serif", 21.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label4.Location = new System.Drawing.Point(3, 179);
```

```
        this.label4.Name = "label4";
        this.label4.Size = new System.Drawing.Size(794, 91);
        this.label4.TabIndex = 1;
        this.label4.Text = "Ви обрали 2 варіант. Досягнено фінальний модуль";
        this.label4.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
        //
        // label5
        //
        this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif", 21.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
        this.label5.Location = new System.Drawing.Point(3, 180);
        this.label5.Name = "label5";
        this.label5.Size = new System.Drawing.Size(794, 91);
        this.label5.TabIndex = 2;
        this.label5.Text = "Ви обрали 1 варіант. Досягнено фінальний модуль";
        this.label5.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
        //
        // button5
        //
        this.button5.Location = new System.Drawing.Point(3, 203);
        this.button5.Name = "button5";
        this.button5.Size = new System.Drawing.Size(794, 42);
        this.button5.TabIndex = 2;
        this.button5.Text = "Наступний крок";
        this.button5.UseVisualStyleBackColor = true;
        this.button5.Click += new System.EventHandler(this.button5_Click);
        //
        // Form1
        //
```

```
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.Black;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.tabControl1);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;
            this.Name = "Form1";
            this.ShowIcon = false;
            this.Text = "Demo";
            this.tabControl1.ResumeLayout(false);
            this.tabPage1.ResumeLayout(false);
            this.tabPage2.ResumeLayout(false);
            this.tabPage2.PerformLayout();
            this.tabPage3.ResumeLayout(false);
            this.tabPage4.ResumeLayout(false);
            this.tabPage5.ResumeLayout(false);
            this.tabPage6.ResumeLayout(false);
            this.tabPage7.ResumeLayout(false);
            this.ResumeLayout(false);
        }

        #endregion

        private System.Windows.Forms.TabControl tabControl1;
        private System.Windows.Forms.TabPage tabPage1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TabPage tabPage2;
        private System.Windows.Forms.TabPage tabPage3;
```

```
        private System.Windows.Forms.TabPage tabPage4;
        private System.Windows.Forms.TabPage tabPage5;
        private System.Windows.Forms.TabPage tabPage6;
        private System.Windows.Forms.TabPage tabPage7;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Timer timer1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Label label5;
    }
}


using StoryFine;

namespace SFDemo
{
    class GameEpisode : ISFEpisode
    {
        public SFSearcher IS_CURRENT_TIME_EVEN;

        public SFEntry entry;
        public SFCheckpoint PLAYER_ENTER_FIRST;
        public SFCondition EVEN_CONDITION;
        public SFChoice IF_EVEN_CHOICE;
```

```
        public SFCheckpoint EVEN_STEP_1;
        public SFCheckpoint EVEN_STEP_2_FINAL;
        public SFCheckpoint EVEN_FINAL;
        public SFCheckpoint NOT_EVEN_FINAL;

        public GameEpisode(SFSearcher.LogicSearcher _ls)
        {
            IS_CURRENT_TIME_EVEN = new
SFSearcher("IS_CURRENT_TIME_EVEN", _ls);

            entry = new SFEntry("entry");
            PLAYER_ENTER_FIRST = new
SFCheckpoint("PLAYER_ENTER_FIRST",entry,false);
            EVEN_CONDITION = new
SFCondition("EVEN_CONDITION",PLAYER_ENTER_FIRST,IS_CURRENT_TI
ME_EVEN);
            IF_EVEN_CHOICE = new
SFChoice("IF_EVEN_CHOICE",EVEN_CONDITION);
            EVEN_STEP_1 = new
SFCheckpoint("EVEN_STEP_1",IF_EVEN_CHOICE,false);
            EVEN_STEP_2_FINAL = new
SFCheckpoint("EVEN_STEP_2_FINAL",EVEN_STEP_1,true);
            EVEN_STEP_1.Next = EVEN_STEP_2_FINAL;
            IF_EVEN_CHOICE.A = EVEN_STEP_1;
            EVEN_FINAL = new
SFCheckpoint("EVEN_FINAL",IF_EVEN_CHOICE,true);
            IF_EVEN_CHOICE.B = EVEN_FINAL;
            EVEN_CONDITION.Then = IF_EVEN_CHOICE;
            NOT_EVEN_FINAL = new
SFCheckpoint("NOT_EVEN_FINAL",EVEN_CONDITION,true);
```

```
            EVEN_CONDITION.Else = NOT_EVEN_FINAL;
            PLAYER_ENTER_FIRST.Next = EVEN_CONDITION;
            entry.Next = PLAYER_ENTER_FIRST;
        }
    }
}
```

```
<!doctype html>
<html lang="en">
<head><meta charset="UTF-8">
<meta name="viewport" content="width=device-width,user-scalable=no,initial-scale=1,maximum-scale=1,minimum-scale=1">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<link rel="icon" href="favicon.ico" type="image/ico">
<title>Storyfine Editor</title>
<link href="main.a8d0feb2a0d25f089c2d.css" rel="stylesheet">
</head><body>
<div id="app">
</div><script src="681.a8d0feb2a0d25f089c2d.js">
</script>
<script src="main.a8d0feb2a0d25f089c2d.js">
</script>
</body></html>
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛВ НА ДИСКУ

Ім'я файла	**Опис**
Пояснювальні документи	
Поясн записка Нємкович .doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Нємкович.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Program Нємкович.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Нємкович .pptx	Презентація кваліфікаційної роботи.