Міністерство освіти і науки України Тернопільський національний педагогічний університет імені Володимира Гнатюка

Фізико-математичний факультет

Кафедра інформатики та методики її навчання

Кваліфікаційна робота

Розробка міжплатформної гри Candy Clicker

Спеціальність:

122 Комп'ютерні науки Освітня програма «Інженерія ігрових проєктів»

Здобувача вищої освіти освітньо-кваліфікаційного рівня «бакалавр»

Гаврилюка Андрія Васильовича

НАУКОВИЙ КЕРІВНИК:

кандидат педагогічних наук, доцент кафедри інформатики та методики її навчання

Габрусєв Валерій Юрійович

РЕЦЕНЗЕНТ:

Габрусєва Ірина Юріївна,

кандидат технічних наук,

доцент кафедри вищої математики Тернопільського національного університету імені Івана Пулюя.

Тернопіль – 2026

ЗМІСТ

ВСТУП

Актуальність проблеми дослідження. Сучасна ігрова індустрія є одним із найбільш динамічних та комерційно успішних сегментів глобального ринку цифрових технологій, що за обсягом доходів перевершує кіноіндустрію та музичну галузь разом узяті, а кількість активних гравців у світі перевищує три мільярди осіб. Серед різноманіття ігрових жанрів особливе місце посідають інкрементальні ігри (Idle/Clicker), які за останнє десятиліття перетворилися з нішевого експериментального формату на повноцінний комерційно успішний сегмент із глобальним ринком обсягом 2,5 мільярда доларів США у 2024 році та прогнозованим зростанням до 6,1 мільярда доларів до 2033 року. Низький поріг входу для розробників, мінімальні вимоги до апаратного забезпечення користувачів та ефективні механіки залучення гравців роблять цей жанр привабливою платформою для дослідження принципів ігрового проєктування та програмної інженерії. Водночас стрімкий розвиток вільних ігрових рушіїв, зокрема Godot Engine із відкритим вихідним кодом та ліцензією MIT, створює нові можливості для розробки повнофункціональних ігрових застосунків без фінансових зобов'язань, що є особливо актуальним для освітніх та дослідницьких проєктів. Актуальність даної роботи обумовлена необхідністю систематизації досвіду застосування рушія Godot Engine для розробки ігор жанру Idle/Clicker, дослідження архітектурних патернів, специфічних для інкрементальних ігор, та практичної реалізації збалансованої ігрової системи з урахуванням психологічних механізмів залучення гравців та сучасних вимог до кросплатформних мобільних застосунків.

Об'єкт дослідження - процес розробки двовимірних ігрових застосунків жанру Idle/Clicker із використанням сучасних технологій ігрової розробки, що включає аналіз архітектурних патернів, методів балансування ігрової економіки та підходів до організації взаємодії з користувачем у контексті специфічних вимог інкрементального жанру.

Предмет дослідження - методи та засоби реалізації ігрового застосунку «Candy Clicker» на базі рушія Godot Engine, зокрема архітектура програмного забезпечення на основі системи вузлів та сигналів, алгоритми розрахунку ігрової прогресії та пасивного накопичення ресурсів, механізми збереження та відновлення стану гри, а також принципи побудови адаптивного графічного інтерфейсу з використанням вбудованих засобів Godot.

Мета дослідження - розробка повнофункціонального ігрового застосунку «Candy Clicker» з використанням рушія Godot Engine версії 4.x та мови програмування GDScript, що реалізує класичні механіки інкрементального жанру в оригінальному тематичному оформленні кондитерської тематики та забезпечує збалансований ігровий досвід із можливістю збереження прогресу між сесіями.

Завдання дослідження:

– провести аналіз жанру Idle/Clicker ігор, їх ключових механік та психологічних принципів залучення гравців;

– здійснити порівняльний аналіз сучасних ігрових рушіїв для двовимірної розробки та обґрунтувати вибір Godot Engine як інструменту реалізації;

– дослідити архітектурні особливості рушія Godot, зокрема систему вузлів, сцен, сигналів та мову програмування GDScript;

– спроєктувати архітектуру програмного забезпечення на основі компонентного підходу із забезпеченням модульності та розширюваності;

– розробити математичну модель ігрової економіки з експоненційним зростанням вартості генераторів та лінійною продуктивністю;

– реалізувати систему персистентного збереження ігрового стану з використанням JSON-серіалізації та механізмом розрахунку офлайн-прогресу;

– розробити адаптивний графічний інтерфейс користувача, оптимізований для екранів різної роздільної здатності та сенсорного вводу;

– провести тестування та оптимізацію продуктивності застосунку на цільових платформах Windows, Linux, Android та HTML5.

Методологія дослідження. У кваліфікаційній роботі використано методи об'єктно-орієнтованого проєктування та програмування для структурування кодової бази у вигляді ієрархії класів із чітко визначеними відповідальностями; принципи компонентної архітектури, реалізовані через систему вузлів Godot Engine; методи математичного моделювання для формалізації ігрової економіки та визначення оптимальних параметрів балансу; методи системного аналізу для дослідження предметної області інкрементальних ігор; а також емпіричні методи тестування, що включають модульне, інтеграційне, системне та UX-тестування для верифікації коректності роботи та оцінки якості користувацького досвіду.

Наукова новизна. У даній роботі запропоновано комплексний підхід до проєктування Clicker-гри на базі рушія Godot Engine, що поєднує теоретичний аналіз психологічних механізмів залучення гравців із практичною реалізацією збалансованої ігрової системи. Адаптовано архітектурні патерни системи вузлів та сигналів Godot до специфічних вимог інкрементального жанру, зокрема розроблено ефективну модель управління ігровою економікою з експоненційною функцією вартості та лінійною продуктивністю генераторів, а також реалізовано механізм розрахунку офлайн-прогресу з обмеженням максимального періоду накопичення та коефіцієнтом зниження пасивного доходу для збереження цінності активної гри.

Практичне значення одержаних результатів полягає у створенні готового до розповсюдження ігрового продукту «Candy Clicker», який може бути опублікований на платформах цифрової дистрибуції для персональних комп'ютерів (Windows, Linux) та мобільних пристроїв (Android), а також запущений у веб-браузері через HTML5-збірку. Систематизований досвід використання Godot Engine для розробки ігор жанру Clicker, включаючи архітектурні рішення, методи балансування економіки та оптимізації продуктивності, може бути корисним для інших розробників, які планують створення аналогічних проєктів, а також використаний як навчальний матеріал у курсах ігрової розробки.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи доповідалися на:

Структура роботи. Кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 43 найменувань, 2 додатків, 9 таблиць та 4 ілюстрацій по тексту. Повний обсяг роботи становить 77 сторінок, з них 58 сторінок основного тексту. У першому розділі проведено аналіз жанру Idle/Clicker ігор, порівняльний огляд ігрових рушіїв та дослідження архітектури Godot Engine. У другому розділі здійснено проєктування концепції гри, архітектури програмного забезпечення, математичної моделі ігрової економіки та алгоритмів збереження прогресу. У третьому розділі описано практичну реалізацію ігрових модулів, графічного інтерфейсу користувача, а також наведено результати тестування та оптимізації продуктивності застосунку.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГРОВИХ ПРОЄКТІВ ЖАНРУ CLICKER

1.1. Аналіз жанру Idle/Clicker ігор та їх особливостей

Сучасна ігрова індустрія характеризується надзвичайним розмаїттям жанрів та механік, серед яких особливе місце посідають так звані Idle-ігри або Clicker-ігри, що за останнє десятиліття перетворилися з нішевого експериментального формату на повноцінний комерційно успішний сегмент ринку [1, с. 120]. Incremental game (інкрементальна гра) - це жанр відеоігор, основною механікою якого є виконання простих повторюваних дій, таких як натискання на екран або клавішу, з метою накопичення віртуальних ресурсів, які згодом інвестуються у покращення, що автоматизують або прискорюють подальший прогрес гравця. Визначальною рисою цього жанру є можливість продовження ігрового прогресу навіть за відсутності активної участі користувача, що й зумовило появу альтернативної назви «idle games» (від англійського «idle» - бездіяльний, пасивний). Ця унікальна особливість радикально відрізняє дані ігри від традиційних жанрів, де прогрес безпосередньо залежить від безперервної взаємодії гравця з ігровим світом, і водночас створює специфічну модель залучення аудиторії, засновану на періодичних коротких сесіях замість тривалих безперервних ігрових марафонів.

Історія виникнення жанру інкрементальних ігор бере свій початок у 2002 році, коли американський розробник Ерік Фредріксен створив гру Progress Quest як сатиричну пародію на рольові ігри того часу, зокрема на численні MMORPG, що потребували від гравців багатогодинного гринду для досягнення прогресу [2, с. 73–86]. У Progress Quest гравець лише створював персонажа, після чого гра повністю грала сама за себе: персонаж автоматично досліджував підземелля, бився з монстрами та підвищував рівень без жодного втручання з боку користувача. Попри те, що гра задумувалася як жарт над ігровою індустрією, вона несподівано знайшла свою аудиторію та заклала концептуальний

фундамент для майбутнього жанру, продемонструвавши, що сам факт спостереження за автоматичним прогресом може бути джерелом задоволення для певної категорії користувачів. Важливим етапом у формуванні жанру стала поява у 2010 році гри Cow Clicker, створеної геймдизайнером Яном Богостом як критичний коментар щодо механік соціальних ігор на платформі Facebook. Парадоксально, але гра, задумана як пародія та критика, здобула величезну популярність і вперше привернула увагу широкої громадськості до феномену ігор із мінімальною інтерактивністю [3].

Справжньою революцією в жанрі стала поява 8 серпня 2013 року браузерної гри Cookie Clicker, створеної французьким розробником Жюльєном Тьєнно. Примітно, що перша версія гри була написана автором протягом однієї ночі та опублікована на форумі 4chan, де вона протягом лічених годин привернула увагу понад 50 000 гравців, а через місяць після запуску щоденна аудиторія сягала 200 000 унікальних користувачів. Cookie Clicker пропонувала гравцям натискати на велике зображення печива для отримання віртуальних печив, які можна було витрачати на придбання різноманітних «будівель» - від курсорів до фабрик, кожна з яких автоматично генерувала печиво з певною швидкістю. Саме Cookie Clicker сформувала канонічну структуру жанру, яка включає систему активного кліку, пасивного доходу від придбаних генераторів, ієрархію покращень із прогресивно зростаючою вартістю та ефективністю, а також механіку «престижу» - добровільного скидання прогресу в обмін на постійні бонуси, що прискорюють проходження у наступних циклах гри.

Феноменальна популярність Clicker-ігор значною мірою пояснюється їх здатністю ефективно активувати системи винагороди в людському мозку, зокрема механізми, пов'язані з виробленням нейромедіатора дофаміну. З точки зору нейропсихології, кожне натискання на екран, що супроводжується візуальним зворотним зв'язком у вигляді зростання числових показників, викликає мікровикид дофаміну - нейромедіатора, відповідального за відчуття задоволення та мотивації. Цей ефект формує так званий «дофаміновий цикл» (dopamine loop) - замкнений ланцюг дії, очікування та винагороди, який спонукає

гравця повторювати певну поведінку знову і знову. Дослідники в галузі поведінкової психології проводять паралелі між механіками Clicker-ігор та класичними експериментами Берреса Скіннера з оперантного обумовлення. Ключова відмінність полягає в тому, що на відміну від біологічної потреби в їжі, цифрові числа в грі не мають жодної об'єктивної цінності, проте людський мозок реагує на їх зростання так, ніби отримує реальну матеріальну винагороду.

Окрім дофамінових механізмів миттєвої винагороди, Clicker-ігри майстерно експлуатують фундаментальну людську потребу у відчутті прогресу та досягнень. Ігри цього жанру пропонують безперервний потік цілей різного масштабу - від короткострокових (накопити достатньо ресурсів для наступного покращення) до довгострокових (розблокувати всі типи генераторів або досягти астрономічних показників виробництва), що створює багатошарову систему мотивації, здатну утримувати увагу гравця протягом тривалого часу. Особливу роль відіграє система досягнень (achievements), яка формалізує та візуалізує прогрес гравця, надаючи йому конкретні орієнтири для подальших зусиль. Психологи відзначають, що в умовах сучасного стресового життя Clicker-ігри пропонують доступний та безпечний спосіб отримати відчуття компетентності та успіху, хоча це ж явище становить предмет занепокоєння щодо потенційної залежності від такого типу розваг [4, с. 339–349].

Аналіз структури типової Clicker-гри дозволяє виділити кілька ключових ігрових механік, що формують ядро геймплею та забезпечують залучення гравця. Першою та найбільш очевидною механікою є активний клік - безпосередня взаємодія гравця з ігровим елементом, кожне натискання на який генерує певну кількість ігрової валюти. Другою критично важливою механікою є система генераторів пасивного доходу - різноманітних об'єктів, які автоматично виробляють ресурси з плином часу без участі гравця. Третьою механікою є ієрархічна система покращень (upgrades), яка дозволяє підвищувати ефективність як активного кліку, так і пасивних генераторів. Четвертою механікою є система престижу (prestige system) - можливість добровільно скинути весь накопичений прогрес в обмін на постійні множники, що значно

прискорюють проходження гри при повторному старті. Нарешті, п'ятою важливою механікою є офлайн-прогрес - здатність гри продовжувати накопичувати ресурси навіть коли додаток закритий, що забезпечує позитивний досвід повернення до гри та мотивує регулярні короткі сесії.

Таблиця 1.1 - Порівняльна характеристика популярних представників жанру Idle/Clicker

Гра	Рік	Платформи	Ключові особливості
Cookie Clicker	2013	Web, Steam, Mobile	Golden Cookies, понад 15 типів будівель
AdVenture Capitalist	2014	Web, Mobile, Steam	Офлайн-прогрес, менеджери, три локації
Clicker Heroes	2014	Web, Mobile, Steam	RPG-елементи, система героїв, Ascension
Idle Miner Tycoon	2016	Mobile	Управління шахтою, Super Managers
Egg, Inc.	2016	Mobile	Кооперативні контракти, prestige eggs

Серед представників жанру особливої уваги заслуговує гра AdVenture Capitalist, випущена студією Hyper Hippo у 2014 році, яка стала першою інкрементальною грою, що успішно реалізувала повноцінну систему офлайн-прогресу та професійну модель монетизації для мобільних платформ. На відміну від попередніх браузерних Clicker-ігор, що працювали лише при відкритій вкладці браузера, AdVenture Capitalist відстежувала час, проведений гравцем поза грою, та нараховувала відповідну кількість ресурсів при наступному запуску, що кардинально змінило ритм взаємодії користувача з грою. Ще одним важливим етапом еволюції жанру стала поява гри Clicker Heroes від студії Playsaurus, яка успішно інтегрувала механіки інкрементальних ігор з елементами рольових ігор, створивши новий піджанр Idle RPG, що згодом став одним із найприбутковіших сегментів мобільного ігрового ринку.

З економічної точки зору, сегмент Idle-ігор демонструє вражаюче зростання та комерційний потенціал. За даними аналітичних агентств, глобальний ринок Idle-ігор у 2024 році досяг обсягу 2,5 мільярда доларів США, а прогнозований середньорічний темп зростання на період до 2033 року становить близько 10,8%, що дозволяє очікувати досягнення ринком позначки у 6,1 мільярда доларів. Регіональна структура ринку характеризується

домінуванням Азійсько-Тихоокеанського регіону з часткою понад 42% глобальних доходів. Модель монетизації Idle-ігор передбачає, що 60-70% виручки надходить від показу реклами у форматі rewarded video, тоді як 30-40% забезпечують внутрішньоігрові покупки. Така модель є значно м'якшою порівняно з агресивними практиками деяких free-to-play ігор та дозволяє користувачам повноцінно насолоджуватися грою без обов'язкових витрат.

Важливим аспектом проєктування Clicker-ігор є математичне балансування ігрової економіки, що потребує ретельного налаштування кривих зростання вартості та ефективності різних елементів гри. Базова формула вартості наступного рівня генератора зазвичай має експоненційний характер: $cost\_next = cost\_base * \left(rate_{growth}\right)^{owned}$, де rate_growth є коефіцієнтом зростання, що типово становить від 1,07 до 1,15. Таке співвідношення гарантує, що на початкових етапах гри виробництво значно перевищує вартість нових покупок, створюючи відчуття швидкого прогресу, проте з часом експоненційне зростання витрат формує так звані «стіни» - точки, де подальший прогрес суттєво сповільнюється. Майстерне налаштування цих параметрів є критично важливим для комерційного успіху гри: занадто швидкий прогрес призводить до швидкого вичерпання контенту, тоді як занадто повільний - до фрустрації та негативних відгуків [5, с. 38–51].

Підсумовуючи проведений аналіз, можна констатувати, що жанр Idle/Clicker ігор пройшов значний шлях розвитку від експериментальних сатиричних проєктів до повноцінного комерційно успішного сегменту ігрової індустрії з власною екосистемою піджанрів та усталеними механіками. Ключовими факторами успіху жанру є його здатність ефективно активувати психологічні механізми винагороди, низький поріг входу, що забезпечує доступність для широкої аудиторії, можливість інтеграції в повсякденне життя користувача, а також гнучкість формату, що дозволяє успішно поєднувати базові інкрементальні механіки з елементами інших популярних жанрів. Розробка гри «Candy Clicker», що є предметом даної кваліфікаційної роботи, базується на аналізі найкращих практик жанру та спрямована на створення збалансованого

ігрового досвіду, що поєднує задоволення від прогресії з повагою до часу та психологічного благополуччя користувача, використовуючи при цьому сучасні технології ігрової розробки на базі рушія Godot.

1.2. Огляд ігрових рушіїв для розробки 2D ігор

Ігровий рушій (game engine) являє собою комплексне програмне забезпечення, що надає розробникам інтегрований набір інструментів, бібліотек та середовищ виконання для створення відеоігор різних жанрів та рівнів складності, суттєво спрощуючи процес розробки за рахунок абстрагування низькорівневих технічних аспектів, таких як рендеринг графіки, обробка фізики, управління пам'яттю та взаємодія з апаратним забезпеченням різних платформ [6, с. 622–642]. Термін «game engine» набув широкого вжитку після випуску компанією id Software гри Doom у 1993 році, архітектура якої була спеціально спроєктована для ліцензування та використання іншими розробниками, що поклало початок практиці комерційного розповсюдження ігрових рушіїв як самостійних продуктів. Сучасні ігрові рушії включають кілька ключових функціональних підсистем: рендерер для візуалізації графічного контенту, фізичний двигун для симуляції поведінки об'єктів та звукову підсистему для відтворення аудіо.

Вибір ігрового рушія для конкретного проєкту є стратегічним рішенням, що суттєво впливає на весь подальший процес розробки, визначаючи доступний інструментарій, цільові платформи та модель ліцензування. Для розробки двовимірних ігор, до яких належить і проєкт «Candy Clicker», на ринку представлено широкий спектр рушіїв, серед яких виділяються Unity, Unreal Engine, Godot, GameMaker, Defold та Construct. Unity є одним із найпопулярніших рушіїв у світі, що використовує мову C# та пропонує модель freemium-ліцензування з безкоштовною персональною версією для проєктів із річним доходом до 100 000 доларів США. Unreal Engine традиційно асоціюється з тривимірними проєктами та використовує C++ разом із системою візуального

скриптування Blueprints, однак для типових 2D-проєктів жанру Idle/Clicker може виявитися надмірно складним та ресурсоємним.

GameMaker є спеціалізованим рушієм для двовимірної розробки, що використовувався для створення таких успішних проєктів як Undertale та Hotline Miami, однак передбачає платну підписку для повного доступу до функціональності. Defold, спочатку створений компанією King для серії Candy Crush, орієнтований на мобільні платформи та використовує мову Lua. Construct дозволяє створювати ігри без програмування через систему візуальних подій, що робить його ідеальним для прототипування, але обмежує можливості для складних проєктів.

Таблиця 1.2 - Порівняльна характеристика ігрових рушіїв для 2D-розробки

Характеристика	Unity	Unreal Engine	Godot	GameMaker
Ліцензія	Freemium	Роялті 5% після $1M	MIT (безкоштовно)	Підписка
Мова програмування	C#	C++, Blueprints	GDScript, C#	GML
Розмір базової збірки	~50-100 МБ	~100-300 МБ	~20-40 МБ	~15-50 МБ
Поріг входу	Середній	Високий	Низький	Низький

Аналіз даних таблиці дозволяє виділити ключові диференціюючі фактори між рушіями. Для проєктів жанру Idle/Clicker особливо важливими є розмір базової збірки та поріг входу [7, с. 86–89]. Godot демонструє оптимальний баланс між цими параметрами, пропонуючи найменший розмір збірки серед універсальних рушіїв та повністю безкоштовну ліцензію MIT без будь-яких роялті. Жанр Idle/Clicker характеризується простими вимогами до графіки, але потребує ефективної системи управління станом гри, збереження прогресу та оптимізації для тривалої роботи з обчисленням офлайн-прогресу.

Важливим фактором є також активність спільноти користувачів рушія. Unity та Unreal Engine мають найбільші спільноти з мільйонами користувачів. Godot демонструє динамічне зростання, що частково пояснюється розчаруванням розробників змінами в політиці ліцензування Unity у 2023 році та

загальним трендом на підтримку відкритого програмного забезпечення [8, с. 36]. Враховуючи всі фактори - технічні можливості, ліцензування, поріг входу, розмір збірки та вимоги жанру - для реалізації проєкту «Candy Clicker» було обрано Godot Engine, що поєднує низький поріг входу з достатньою потужністю, безкоштовну ліцензію, компактний розмір збірки та підтримку всіх необхідних платформ, включаючи Windows, Android та HTML5.

1.3. Характеристика рушія Godot та мови програмування GDScript

Godot Engine є вільним ігровим рушієм із відкритим вихідним кодом, що розповсюджується під ліцензією MIT і надає розробникам повноцінний інструментарій для створення двовимірних та тривимірних відеоігор без будь-яких фінансових зобов'язань, роялті чи обмежень на комерційне використання кінцевого продукту [9, с. 169]. Назва рушія походить від п'єси Семюела Беккета «Чекаючи на Годо» (Waiting for Godot), що символізує філософію постійного вдосконалення та очікування нових можливостей з кожним оновленням. Рушій був створений аргентинськими розробниками Хуаном Лініецьким та Аріелем Манзуром, які розпочали роботу над проєктом у 2007 році як внутрішній інструмент, а у 2014 році опублікували вихідний код на GitHub під вільною ліцензією, що дозволило глобальній спільноті долучитися до розвитку проєкту.

Архітектура Godot Engine базується на унікальній концепції вузлів (nodes) та сцен (scenes), яка суттєво відрізняється від підходів інших рушіїв і пропонує інтуїтивно зрозумілу модель організації проєкту, де кожен елемент гри представлений як вузол у деревовидній структурі, а групи пов'язаних вузлів формують сцени, що можуть бути повторно використані та вкладені одна в одну [10]. Вузол (Node) є базовою одиницею композиції ігрового світу, що може представляти широкий спектр функціональних елементів - від трансформацій положення до контролерів фізики, джерел звуку та елементів інтерфейсу, причому кожен тип вузла успадковує властивості від батьківських класів та додає власну специфічну функціональність. Сцена (Scene) являє собою дерево

вузлів, збережене як окремий файл із розширенням .tscn, що може функціонувати як самостійна ігрова локація або як повторно використовуваний компонент на кшталт персонажа чи інтерактивного об'єкта.

Для розробки двовимірних ігор Godot надає спеціалізований набір вузлів із префіксом «2D», оптимізованих для роботи в площині. Базовим вузлом є Node2D, який надає властивості позиції, повороту та масштабу у двовимірному просторі. Sprite2D відповідає за відображення растрових зображень на екрані та підтримує анімацію через атласи спрайтів, модуляцію кольору та прозорості. AnimatedSprite2D розширює можливості базового спрайта, надаючи вбудовану систему покадрової анімації з підтримкою множинних анімаційних послідовностей. Система фізичного моделювання реалізована через спеціалізовані вузли: StaticBody2D для нерухомих об'єктів, RigidBody2D для динамічних тіл із фізичною симуляцією та CharacterBody2D для керованих персонажів із обробкою зіткнень [11, с. 54–60].

Графічний інтерфейс користувача (GUI) в Godot реалізований через гілку вузлів, що починається з базового класу Control і включає широкий спектр готових віджетів для створення меню, діалогових вікон та індикаторів. Система GUI підтримує автоматичне масштабування та позиціонування елементів через механізм «якорів» (anchors), що забезпечує коректне відображення на пристроях із різною роздільною здатністю. Button є базовим інтерактивним елементом, що реагує на натискання та підтримує різні візуальні стани. Label забезпечує відображення тексту з підтримкою різних шрифтів та автоматичного переносу рядків. TextureRect та TextureButton дозволяють інтегрувати растрові зображення в інтерфейс як декоративні елементи або інтерактивні кнопки.

Таблиця 1.3 - Основні типи вузлів Godot для 2D-розробки

Категорія	Вузол	Функціональне призначення
Візуалізація	Sprite2D	Відображення статичних зображень
Візуалізація	AnimatedSprite2D	Покадрова анімація спрайтів
Фізика	StaticBody2D	Нерухомі фізичні об'єкти
Фізика	CharacterBody2D	Керовані персонажі з обробкою зіткнень

Інтерфейс	Button	Інтерактивна кнопка
Інтерфейс	Label	Текстова мітка
Таймери	Timer	Відлік часу та періодичні події

Представлена класифікація вузлів демонструє модульний підхід Godot, де кожен вузол виконує чітко визначену роль, а складна поведінка досягається композицією простих компонентів. Для проєкту «Candy Clicker» особливо релевантними є вузли категорій «Інтерфейс» та «Таймери», оскільки ігри жанру Clicker переважно складаються з елементів GUI та систем автоматичного накопичення ресурсів.

GDScript є основною мовою програмування Godot Engine, спеціально розробленою для максимальної інтеграції з архітектурою рушія та оптимізованою для типових завдань ігрової розробки [12-13]. Синтаксис GDScript змодельований за зразком Python - мови з мінімалістичною пунктуацією, обов'язковими відступами для визначення блоків коду та зрозумілими ключовими словами, завдяки чому код читається майже як структурований текст і може бути зрозумілий навіть особам без глибокої технічної підготовки. Мова є динамічно типізованою, хоча починаючи з версії Godot 4.0 розробники можуть використовувати опціональні анотації типів для покращення продуктивності та раннього виявлення помилок.

Ключовою особливістю GDScript є глибока інтеграція з системою вузлів та сцен, що виражається у спеціальних синтаксичних конструкціях для роботи з ієрархією об'єктів. Конструкція extends вказує базовий клас, від якого успадковується поведінка. Анотація @onready вказує, що значення змінної має бути обчислене після повної побудови дерева вузлів сцени, що є критичним при отриманні посилань на інші вузли. Анотація @export робить змінну видимою та редагованою в інспекторі властивостей візуального редактора, дозволяючи налаштовувати параметри об'єкта без модифікації коду.

Система сигналів (signals) реалізує патерн «спостерігач» і є основним механізмом комунікації між вузлами, що дозволяє реалізувати слабко зв'язану

архітектуру, в якій об'єкти можуть реагувати на події інших об'єктів без прямих залежностей [14]. Сигнал оголошується за допомогою ключового слова signal, а емісія здійснюється викликом методу emit(). Підключення обробника може здійснюватися як програмно через метод connect(), так і візуально через інтерфейс редактора. У контексті Clicker-гри система сигналів ідеально підходить для оновлення інтерфейсу при зміні ігрових показників: основний скрипт емітує сигнали про зміну ресурсів, а елементи інтерфейсу автоматично оновлюють відображення.

Життєвий цикл вузла визначається набором віртуальних методів, що автоматично викликаються рушієм на різних етапах існування об'єкта. Метод _ready() викликається одноразово після ініціалізації вузла та його дочірніх елементів, що робить його ідеальним місцем для початкового налаштування. Метод _process(delta) викликається кожен кадр і приймає параметр delta - час, що минув з попереднього кадру, який необхідно використовувати для масштабування руху чи зміни стану. Метод _input(event) викликається при будь-якій події вводу та дозволяє реалізувати гнучку обробку користувацького вводу [15, с. 2996–3006].

Управління даними та їх персистентне збереження є критично важливим аспектом розробки Clicker-ігор. Godot надає клас FileAccess для роботи з файловою системою, що дозволяє створювати та читати файли у директорії user://, яка автоматично маппується на відповідну системну директорію на кожній платформі. Найпоширенішим підходом є серіалізація даних у формат JSON, що є людиночитаним і дозволяє легко налагоджувати збережені файли під час розробки.

Редактор Godot Engine являє собою повнофункціональне інтегроване середовище розробки (IDE), що включає візуальний редактор сцен, текстовий редактор з підсвічуванням синтаксису та автодоповненням, а також інтегрований налагоджувач. Центральним елементом є вікно перегляду сцени, де розробник може візуально розміщувати вузли та налаштовувати їх властивості через панель інспектора. Особливо корисною є можливість запустити проєкт безпосередньо з

редактора, причому при помилках редактор автоматично переходить до відповідного рядка коду [16].

Експорт готової гри здійснюється через систему шаблонів експорту для кожної цільової платформи. Для Android Godot генерує підписаний APK або AAB файл, готовий для Google Play Store. HTML5-експорт дозволяє запускати гру у веб-браузері через WebAssembly та WebGL. Версія Godot 4.0, випущена у березні 2023 року, принесла значні покращення, включаючи новий рендерер Vulkan та оновлений синтаксис GDScript із анотаціями @export та @onready. Для проєкту «Candy Clicker» використовується актуальна версія Godot 4.x.

Спільнота Godot є однією з найактивніших серед ігрових рушіїв. Офіційна документація є якісною та всеохоплюючою, включаючи детальний опис API та покрокові туторіали, причому доступна множиною мов, включаючи українську. Офіційний форум, Discord та Reddit забезпечують оперативну допомогу, а бібліотека Asset Library містить тисячі готових скриптів та плагінів для використання у власних проєктах.

Підсумовуючи, Godot Engine є оптимальним вибором для «Candy Clicker» завдяки низькому порогу входу, достатній функціональності, мінімальному розміру збірки та повній безкоштовності [17, с. 156]. Архітектура вузлів та сцен природно відповідає структурі Clicker-гри, де елементи інтерфейсу можуть бути реалізовані як незалежні сцени, об'єднані через систему сигналів. GDScript з Python-подібним синтаксисом дозволяє швидко реалізовувати ігрову логіку, а вбудовані засоби роботи з файлами забезпечують надійне збереження прогресу гравця.

1.4 Постановка задачі дослідження

На основі проведеного аналізу теоретичних засад жанру Idle/Clicker ігор, порівняльного огляду сучасних ігрових рушіїв для двовимірної розробки та детального вивчення архітектурних особливостей Godot Engine сформульовано основну мету даного дослідження, яка полягає у розробці повнофункціонального

ігрового застосунку «Candy Clicker» з використанням рушія Godot версії 4.x та мови програмування GDScript, що реалізує класичні механіки інкрементального жанру в оригінальному тематичному оформленні кондитерської тематики та забезпечує збалансований ігровий досвід із можливістю збереження прогресу між сесіями. Для досягнення поставленої мети необхідно вирішити комплекс взаємопов'язаних завдань, що охоплюють як теоретико-аналітичний, так і практично-реалізаційний аспекти розробки: провести формалізацію предметної області та визначити математичну модель ігрової економіки з урахуванням експоненційного зростання вартості покращень та лінійного збільшення продуктивності генераторів; спроєктувати архітектуру програмного забезпечення на основі системи вузлів та сцен Godot із забезпеченням модульності, розширюваності та простоти підтримки коду; реалізувати систему персистентного збереження ігрового стану з використанням серіалізації даних у форматі JSON для забезпечення надійного зберігання прогресу гравця; розробити графічний інтерфейс користувача, адаптований для різних роздільних здатностей екрана та оптимізований для сенсорного вводу на мобільних пристроях; провести тестування та оптимізацію продуктивності застосунку для забезпечення плавної роботи на цільових платформах.

Об'єктом дослідження є процес розробки двовимірних ігрових застосунків жанру Idle/Clicker із використанням сучасних технологій ігрової розробки, що включає аналіз архітектурних патернів, методів балансування ігрової економіки та підходів до організації взаємодії з користувачем у контексті специфічних вимог інкрементального жанру. Предметом дослідження виступають методи та засоби реалізації ігрового застосунку «Candy Clicker» на базі рушія Godot Engine, зокрема архітектура програмного забезпечення на основі системи вузлів та сигналів, алгоритми розрахунку ігрової прогресії та пасивного накопичення ресурсів, механізми збереження та відновлення стану гри, а також принципи побудови адаптивного графічного інтерфейсу з використанням вбудованих засобів Godot для роботи з елементами управління. Методологічну основу дослідження складають методи об'єктно-орієнтованого

проєктування та програмування, що забезпечують структурування коду у вигляді ієрархії класів із чітко визначеними відповідальностями; принципи компонентної архітектури, реалізовані через систему вузлів Godot, де складна поведінка досягається композицією простих елементів; методи математичного моделювання для формалізації ігрової економіки та визначення оптимальних параметрів балансу; а також емпіричні методи тестування, що включають як автоматизовану перевірку коректності роботи окремих модулів, так і ручне тестування користувацького досвіду для виявлення проблем юзабіліті та ергономіки інтерфейсу.

Практичне значення очікуваних результатів дослідження полягає у створенні готового до розповсюдження ігрового продукту, який може бути опублікований на платформах цифрової дистрибуції для персональних комп'ютерів та мобільних пристроїв, а також у систематизації досвіду використання Godot Engine для розробки ігор жанру Clicker, що може бути корисним для інших розробників, які планують створення аналогічних проєктів. Наукова новизна роботи визначається комплексним підходом до проєктування Clicker-гри, що поєднує теоретичний аналіз психологічних механізмів залучення гравців із практичною реалізацією збалансованої ігрової системи, а також адаптацією архітектурних патернів Godot Engine до специфічних вимог інкрементального жанру, зокрема розробкою ефективної системи управління великими числовими значеннями, що виникають при тривалій грі, та оптимізацією механізму розрахунку офлайн-прогресу для забезпечення чесного нарахування ресурсів за період відсутності гравця без створення можливостей для зловживань шляхом маніпуляції системним часом. Очікуваним результатом дослідження є функціонально завершений ігровий застосунок «Candy Clicker», що включає систему активного кліку для отримання базової ігрової валюти, ієрархію генераторів пасивного доходу з прогресивно зростаючою вартістю та ефективністю, механізм збереження та автоматичного завантаження прогресу при запуску гри, візуально привабливий інтерфейс у кондитерській тематиці з анімаціями зворотного зв'язку на дії користувача, а також підтримку експорту

для платформ Windows, Linux, Android та HTML5, що забезпечить максимальне охоплення потенційної аудиторії та можливість демонстрації результатів роботи у різних середовищах виконання.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено комплексне теоретичне дослідження предметної області розробки ігрових проєктів жанру Idle/Clicker, що створило необхідне підґрунтя для практичної реалізації ігрового застосунку «Candy Clicker» на базі рушія Godot Engine. Аналіз історії становлення та еволюції інкрементального жанру дозволив виявити ключові механіки, що забезпечують залучення гравців та комерційний успіх проєктів даного типу, зокрема систему активного кліку як базового способу отримання ресурсів, ієрархію генераторів пасивного доходу з експоненційним зростанням вартості, механіку престижу для забезпечення довгострокової мотивації та систему офлайн-прогресу, що дозволяє грі інтегруватися у повсякденне життя користувача без потреби безперервної активної участі. Дослідження психологічних аспектів жанру виявило, що популярність Clicker-ігор значною мірою пояснюється їх здатністю ефективно активувати дофамінові механізми винагороди в людському мозку через постійний потік мікродосягнень та візуальний зворотний зв'язок у вигляді зростаючих числових показників, що формує так званий дофаміновий цикл і спонукає гравця повертатися до гри знову і знову.

Порівняльний аналіз сучасних ігрових рушіїв для двовимірної розробки, що включав Unity, Unreal Engine, Godot, GameMaker, Defold та Construct, дозволив обґрунтувати вибір Godot Engine як оптимального інструменту для реалізації проєкту «Candy Clicker» на основі комплексу критеріїв, серед яких визначальними стали повністю безкоштовна ліцензія MIT без будь-яких роялті чи обмежень на комерційне використання, мінімальний розмір базової збірки серед універсальних рушіїв, низький поріг входу завдяки інтуїтивній мові

програмування GDScript із Python-подібним синтаксисом, а також активна спільнота з якісною документацією українською мовою. Детальне вивчення архітектури Godot Engine виявило, що його унікальна система вузлів та сцен природно відповідає структурі типової Clicker-гри, де окремі елементи інтерфейсу можуть бути реалізовані як незалежні компоненти з власною логікою, об'єднані через систему сигналів, що забезпечує слабку зв'язаність модулів та простоту підтримки коду в процесі розвитку проєкту.

На основі проведеного теоретичного аналізу сформульовано постановку задачі дослідження, що включає визначення мети, об'єкта, предмета та методологічної основи роботи, а також конкретизацію завдань, необхідних для досягнення поставленої мети. Встановлено, що розробка ігрового застосунку «Candy Clicker» потребує вирішення комплексу взаємопов'язаних завдань від формалізації математичної моделі ігрової економіки до реалізації системи персистентного збереження прогресу та оптимізації продуктивності для цільових платформ. Результати теоретичного дослідження, викладені у першому розділі, становлять методологічну базу для проєктування архітектури програмного забезпечення та практичної реалізації ігрового застосунку, що є предметом наступних розділів кваліфікаційної роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ «CANDY CLICKER»

2.1. Концепція та ігрова механіка гри

Концепція ігрового застосунку «Candy Clicker» базується на класичних принципах інкрементального жанру, адаптованих до оригінальної тематики кондитерських виробів, що створює візуально привабливий та емоційно позитивний ігровий досвід, орієнтований на широку аудиторію користувачів різного віку та рівня ігрового досвіду [18, с. 239]. Центральним елементом ігрового процесу є накопичення віртуальної валюти у вигляді цукерок (candies), які гравець отримує шляхом натискання на основний інтерактивний елемент у центрі екрана, а згодом - завдяки автоматичним генераторам пасивного доходу, що продовжують виробляти ресурси навіть без активної участі користувача. Вибір кондитерської тематики є свідомим дизайнерським рішенням, що ґрунтується на аналізі успішних представників жанру, зокрема культової гри Cookie Clicker, де використання візуально привабливих та асоціативно приємних образів їжі створює додатковий емоційний зв'язок із гравцем та підсилює задоволення від процесу накопичення ресурсів. Назва проєкту «Candy Clicker» є свідомою алюзією на канонічні зразки жанру та одразу комунікує потенційним користувачам жанрову приналежність гри, що є важливим фактором для органічного залучення цільової аудиторії в умовах насиченого ринку мобільних додатків, де користувачі приймають рішення про завантаження протягом кількох секунд на основі назви, іконки та перших скріншотів.

Ігрова механіка (game mechanics) у контексті геймдизайну визначається як формалізована система правил та процедур, що описують можливі дії гравця, їх наслідки та умови переходу між станами ігрової системи, і саме продумана архітектура механік визначає глибину, реіграбельність та здатність гри утримувати увагу користувача протягом тривалого часу [19]. Базовою механікою «Candy Clicker» є активний клік - натискання на центральний ігровий елемент,

кожне з яких генерує визначену кількість цукерок, що додаються до загального балансу гравця та відображаються у відповідному лічильнику інтерфейсу. Початкова ефективність кліку становить одну цукерку за натискання, проте цей показник може бути збільшений шляхом придбання спеціальних покращень, що реалізує принцип прогресії та надає гравцю відчуття зростання власної потужності у грі. Кожне натискання супроводжується візуальним та, за можливості, тактильним зворотним зв'язком у вигляді анімації масштабування елемента, появи числового індикатора отриманих ресурсів та короткої вібрації на мобільних пристроях, що підтримують цю функцію, оскільки дослідження в галузі UX-дизайну демонструють, що мультисенсорний зворотний зв'язок значно підвищує задоволення від взаємодії та формує більш стійкі поведінкові патерни у користувачів.

Система генераторів пасивного доходу є другою фундаментальною механікою, що трансформує «Candy Clicker» з простої гри на реакцію у повноцінний інкрементальний досвід із стратегічним виміром, де гравець має приймати рішення щодо оптимального розподілу накопичених ресурсів між різними типами інвестицій [20]. У грі реалізовано ієрархію генераторів, кожен з яких має унікальну візуальну репрезентацію у вигляді різних видів солодощів - від простих льодяників та карамелі на початкових рівнях до вишуканих тортів та кондитерських фабрик на пізніших етапах прогресії, що створює відчуття розвитку власної «кондитерської імперії» та надає довгострокову мету для утримання мотивації гравця. Кожен генератор характеризується двома ключовими параметрами: базовою вартістю придбання, вираженою у цукерках, та продуктивністю, що визначає кількість цукерок, які даний генератор автоматично виробляє за одиницю часу без участі гравця. Вартість кожної наступної одиниці того самого типу генератора зростає за експоненційним законом відповідно до формули:

$$Cost(n) = BaseCost \times GrowthRate^{n}, \qquad (2.1)$$

де n - кількість вже придбаних одиниць,

GrowthRate - є коефіцієнтом зростання, типово встановленим у діапазоні від 1,10 до 1,15 для забезпечення відчутного, але не надмірного збільшення вартості, що змушує гравця диверсифікувати свої інвестиції між різними типами генераторів замість концентрації на єдиному найефективнішому варіанті.

Таблиця 2.1 - Характеристики генераторів пасивного доходу у грі «Candy Clicker»

Генератор	Базова вартість	Продуктивність (цукерок/сек)	Коефіцієнт зростання
Льодяник	15	0,1	1,12
Карамель	100	0,5	1,12
Шоколадка	500	2,0	1,12
Тістечко	2 000	8,0	1,12
Торт	10 000	30,0	1,12
Кондитерська	50 000	100,0	1,12

Наведені у таблиці параметри генераторів є результатом ітеративного балансування, спрямованого на забезпечення плавної кривої прогресії, де кожен новий тип генератора стає доступним приблизно через рівні проміжки ігрового часу при активній грі, а співвідношення вартості до продуктивності залишається приблизно однаковим для всіх рівнів, що унеможливлює існування єдиної домінуючої стратегії та заохочує різноманітність підходів до розвитку. Базова вартість кожного наступного типу генератора приблизно у п'ять разів перевищує вартість попереднього, тоді як продуктивність зростає у чотири рази, що створює незначну перевагу для інвестицій у більш дорогі генератори при достатньому накопиченні капіталу, але не робить дешевші варіанти повністю неактуальними на жодному етапі гри [21-23]. Уніфікований коефіцієнт зростання вартості 1,12 для всіх типів генераторів спрощує ментальну модель гри для гравця та забезпечує передбачуваність економічних розрахунків, водночас гарантуючи, що вартість десятої одиниці будь-якого генератора становитиме приблизно втричі більше від базової вартості, а сотої - у понад тисячу разів, що природно

сповільнює прогресію на пізніх етапах і створює потребу у механіці престижу для довгострокового утримання.

Взаємодія між механіками активного кліку та пасивної генерації створює динамічний ігровий цикл, де на ранніх етапах гравець покладається на ручне натискання для накопичення початкового капіталу, а із розвитком ігрової імперії баланс поступово зміщується у бік пасивного доходу, що може в сотні разів перевищувати ефективність ручного кліку, трансформуючи роль гравця з активного «виробника» на «менеджера» власної кондитерської імперії. Інтерфейс відображає поточний стан обох джерел доходу: загальний баланс оновлюється у реальному часі, а окремий індикатор показує сукупну продуктивність генераторів у форматі «X цукерок за секунду».

Цілісність ігрового досвіду забезпечується послідовним застосуванням кондитерської тематики до всіх елементів - від фонових зображень та кольорової палітри у теплих пастельних тонах рожевого, бежевого та карамельного до назв генераторів та текстових повідомлень. Типографіка використовує заокруглені шрифти без зарубок, а великі числові показники відображаються у скорочених форматах (K, M, B) для збереження читабельності на пізніх етапах прогресії. Звуковий дизайн доповнює візуальну естетику мелодійними сигналами при успішних діях та ненав'язливою фоновою музикою з можливістю вимкнення через меню налаштувань.

2.2. Архітектура програмного забезпечення

Архітектура ігрового застосунку «Candy Clicker» спроєктована з урахуванням специфіки рушія Godot Engine та його системи вузлів і сцен, що природно реалізує компонентний підхід до організації коду. Ключовим принципом є розділення відповідальностей (separation of concerns), де ігрова логіка, управління даними, візуальне представлення та обробка вводу ізольовані у відокремлених модулях, які взаємодіють через систему сигналів Godot,

забезпечуючи слабку зв'язаність (loose coupling) між компонентами та можливість модифікації окремих частин без каскадних змін у суміжних модулях.

Структура проєкту організована за принципом логічного групування ресурсів: головна сцена main.tscn є точкою входу, каталог assets містить графічні ресурси (підкаталоги gui, candy, font), головний скрипт main.gd реалізує всі основні механіки гри, а splash.tscn із скриптом splash.gd забезпечують стартовий екран при запуску. Конфігураційний файл project.godot містить глобальні налаштування проєкту, а export_presets.cfg зберігає профілі експорту для Windows, Android та HTML5.

Таблиця 2.2 - Структура основних компонентів проєкту «Candy Clicker»

Компонент	Файл	Тип	Функціональне призначення
Головна сцена	main.tscn	Scene	Кореневий контейнер ігрового інтерфейсу
Ігрова логіка	main.gd	Script	Механіки гри, управління станом
Конфігурація	project.godot	Config	Глобальні налаштування проєкту
Налаштування експорту	export_presets.cfg	Config	Профілі збірки для платформ
Графіка інтерфейсу	assets/gui/*	Resources	Візуальні елементи UI
Спрайти генераторів	assets/candy/*	Resources	Зображення солодощів
Шрифти	assets/font/*	Resources	Типографічні ресурси

Архітектура головної сцени main.tscn побудована на ієрархії вузлів Control із адаптивним масштабуванням через систему якорів та контейнерів. Кореневий вузол містить фонове зображення TextureRect, верхню панель статистики з Label балансу та показника пасивної генерації, центральну кнопку TextureButton із зображенням цукерки та анімацією зворотного зв'язку, а також нижню панель генераторів - ScrollContainer із VBoxContainer, де кожен генератор представлений HBoxContainer із зображенням, назвою, вартістю та кнопкою придбання [24].

Система сигналів Godot реалізує патерн «видавець-підписник» для комунікації між компонентами. Головний скрипт оголошує кастомні сигнали candies_changed, generator_purchased та cps_updated, що дозволяє елементам інтерфейсу автоматично оновлюватися при зміні стану без явних викликів. Вбудовані сигнали pressed та timeout підключаються до обробників у функції _ready(), забезпечуючи централізоване налаштування взаємозв'язків між компонентами.

Управління глобальним станом реалізовано через централізований словник (Dictionary) GDScript у головному скрипті, що містить баланс цукерок, кількість генераторів, покращення та статистику. Доступ здійснюється через функції-аксесори з валідацією даних та автоматичною емісією сигналів - наприклад, add_candies() збільшує баланс, генерує сигнал candies_changed та перевіряє milestone-показники. Автозбереження реалізовано через Timer із періодичним викликом функції збереження, а ручне збереження виконується при кожній значущій дії гравця.

2.3. Проєктування системи прогресії та балансу

Система прогресії «Candy Clicker» забезпечує відчуття розвитку гравця через багатошарову структуру цілей - від короткострокових (накопичити ресурси для наступного покращення) до довгострокових (розблокувати всі типи генераторів). Проєктування базується на математичному моделюванні ігрової економіки з урахуванням теорії потоку Міхая Чіксентміхаї, де оптимальний досвід досягається при балансі між складністю завдань та можливостями гравця. Ключовим викликом є забезпечення плавної кривої прогресу, особливо важливої для мобільних ігор, де типова сесія триває лише кілька хвилин і гравець має відчувати прогрес навіть за такий короткий проміжок взаємодії [25]. Математична модель економіки «Candy Clicker» базується на класичній формулі експоненційного зростання вартості генераторів, що є галузевим стандартом для інкрементальних ігор і забезпечує природне сповільнення прогресу на пізніх

етапах без штучних обмежень або «стін», які б повністю блокували просування гравця. Вартість n-ої одиниці генератора розраховується за формулою:

$$C(n) = C_0 * r^n, \tag{2.2}$$

де C_0 - базова вартість першої одиниці генератора,

r - коефіцієнт зростання (у проєкті встановлено значення 1,12 для всіх типів генераторів),

n - кількість вже придбаних одиниць даного типу, причому така формула гарантує, що кожна наступна одиниця коштує на 12% дорожче за попередню, що створює передбачувану, але відчутну ескалацію витрат.

Продуктивність генераторів, на противагу вартості, зростає лінійно пропорційно до кількості придбаних одиниць за формулою:

$$P(n) = P_0 * n, \tag{2.3}$$

де P_0 - базова продуктивність одного генератора,

n - загальна кількість придбаних одиниць, що означає, що десять льодяників виробляють рівно в десять разів більше цукерок, ніж один, без diminishing returns або бонусів за масштаб.

Таке співвідношення експоненційної вартості та лінійної продуктивності створює характерну для жанру динаміку, де на ранніх етапах інвестиції швидко окупаються, а прогрес відчувається стрімким, тоді як на пізніх етапах кожне нове придбання потребує значно більшого часу накопичення, що природно підводить гравця до пошуку альтернативних стратегій оптимізації або використання механіки престижу.

Показник ефективності інвестиції, що дозволяє гравцю та розробнику оцінити оптимальність різних стратегій розвитку, визначається як

співвідношення приросту продуктивності до витраченої вартості, тобто $ROI = \frac{\Delta P}{C}$, де ΔP - збільшення пасивної генерації від придбання, а C - вартість цього придбання у цукерках. Для першої одиниці будь-якого генератора $ROI = \frac{P_0}{C_0}$, і порівняння цього показника між різними типами генераторів дозволяє визначити, який із них забезпечує найкращу віддачу на початковому етапі гри, причому у «Candy Clicker» цей показник свідомо вирівняний між усіма генераторами для уникнення домінуючої стратегії та заохочення різноманітності підходів. Концепція «часу окупності» (payback time) є ще одним корисним інструментом аналізу балансу, що визначає час, необхідний для того, щоб придбаний генератор виробив кількість ресурсів, еквівалентну його вартості, і розраховується за формулою:

$$T = \frac{C}{P}, \tag{2.4}$$

де T - час окупності у секундах,

C - вартість придбання,

P - продуктивність у цукерках за секунду.

Для забезпечення здорової економіки гри час окупності першої одиниці генератора має становити від 30 секунд до кількох хвилин залежно від етапу гри, а для пізніших одиниць того ж типу - поступово зростати, але залишатися в розумних межах, що не перевищують терпіння типового казуального гравця.

Таблиця 2.3 - Аналіз економічних показників генераторів при різній кількості придбаних одиниць

Генератор	Кількість	Вартість наступного	Сумарна продуктивність	Час окупності (сек)
Льодяник	1	17	0,1	170
Льодяник	10	47	1,0	47
Льодяник	25	253	2,5	101

Карамель	1	112	0,5	224
Карамель	10	311	5,0	62
Шоколадка	1	560	2,0	280
Шоколадка	10	1 555	20,0	78
Торт	1	11 200	30,0	373

Дана таблиця демонструє ключову особливість експоненційної моделі: хоча абсолютна вартість кожної наступної одиниці зростає, час окупності для генераторів із значною накопиченою базою може бути нижчим, ніж для перших одиниць дорожчих генераторів, що створює нетривіальний простір для оптимізаційних рішень гравця та уникає ситуації, де єдиною правильною стратегією є завжди купувати найдорожчий доступний варіант. Балансування цих показників здійснювалося ітеративно через серію тестових сесій, де відстежувалися реальні патерни прогресу та час, необхідний для досягнення ключових milestone-показників, з подальшим коригуванням базових параметрів для забезпечення цільової тривалості проходження основного контенту гри [26].

Система прогресії «Candy Clicker» включає кілька паралельних вимірів розвитку: кількісний - зростання балансу цукерок та пасивної генерації з подоланням milestone-порогів (перша тисяча, мільйон, мільярд); якісний - розблокування нових типів генераторів та покращень, що розширюють механічний репертуар гри; колекційний - система досягнень (achievements), що фіксують виконання специфічних умов від тривіальних до амбіційних. Офлайн-прогресія є критично важливим елементом, що дозволяє грі залишатися релевантною навіть під час відсутності користувача, створюючи позитивний емоційний досвід повернення та виявлення накопичених ресурсів. При кожному запуску гри система розраховує час, що минув з моменту останнього збереження, та нараховує відповідну кількість цукерок на основі зафіксованого показника пасивної генерації за формулою:

$$OfflineEarnings = CPS \times OfflineTime \times OfflineMultiplier, \quad (2.5)$$

де CPS - збережений показник цукерок за секунду,

$OfflineTime$ - час відсутності у секундах,

$OfflineMultiplier$ - коефіцієнт, що може бути менший за одиницю для обмеження офлайн-доходу та збереження цінності активної гри.

У «Candy Clicker» коефіцієнт офлайн-генерації встановлено на рівні 0.5 (50% від активної продуктивності), що є компромісним значенням, яке забезпечує відчутну винагороду за повернення до гри, але зберігає значну перевагу для активних гравців, які присвячують час безпосередній взаємодії із застосунком. Максимальний період офлайн-накопичення обмежений 24 годинами для запобігання ситуаціям, де гравець, який повертається після тривалої перерви, отримує настільки значну суму, що весь подальший прогрес втрачає сенс, а також для захисту від маніпуляцій із системним часом пристрою, хоча повноцінний захист від читерства виходить за межі даного проєкту і потребував би серверної валідації, недоцільної для офлайн-гри.

2.4. Розробка алгоритмів збереження та завантаження прогресу

Система збереження та завантаження прогресу у «Candy Clicker» спроєктована з урахуванням множини сценаріїв - від нормального завершення роботи до примусового закриття операційною системою та розрядження батареї [27]. Архітектура базується на принципі надмірності збереження, де критичні дані записуються значно частіше, ніж необхідно при ідеальних умовах. Формат збереження - JSON, обраний за людиночитабельність, стандартизованість та кросплатформну сумісність при незначному розмірі файлу (менше 10 кілобайт). Структура документа включає секції resources (баланс та статистика), generators (кількість придбаних одиниць), upgrades, achievements та meta (версія формату, мітка часу, загальний час гри).

Алгоритм збереження формує словник поточного стану, серіалізує його через JSON.stringify() та записує у директорію user:// через FileAccess із

створенням резервної копії .bak. Мітка часу у форматі Unix timestamp фіксується для розрахунку офлайн-прогресу при наступному завантаженні. Алгоритм завантаження включає парсинг JSON, валідацію даних із стратегією graceful degradation (некоректні значення замінюються на значення за замовчуванням), версіонування формату через поле save_version із ланцюжком міграційних функцій для зворотної сумісності.

Стратегія автозбереження поєднує періодичний підхід (Timer з інтервалом 30 секунд) та подієвий (при придбанні генераторів, активації покращень, отриманні досягнень), а також збереження при переході у фоновий режим через NOTIFICATION_APPLICATION_PAUSED та NOTIFICATION_WM_CLOSE_REQUEST. Обробка помилок реалізована за принципом захисного програмування: при неможливості прочитати файл система завантажує резервну копію .bak, а при її відсутності ініціалізує нову гру з повідомленням користувачу.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи здійснено комплексне проєктування ігрового застосунку «Candy Clicker», що охопило всі ключові аспекти розробки від концептуального дизайну ігрових механік до технічної архітектури програмного забезпечення та алгоритмів управління даними. Розроблена концепція гри базується на класичних принципах жанру Idle/Clicker з оригінальною кондитерською тематикою, де центральними механіками є активний клік для отримання базової валюти та система генераторів пасивного доходу з шістьма рівнями прогресії від найпростіших льодяників до кондитерських фабрик. Детально опрацьовані параметри кожного генератора, включаючи базову вартість, продуктивність та коефіцієнт зростання вартості, забезпечують збалансовану криву прогресу, де гравець постійно відчуває розвиток власної «кондитерської імперії» без надмірних періодів стагнації або нездоланних бар'єрів.

Архітектура програмного забезпечення спроєктована з використанням компонентного підходу на базі системи вузлів та сцен Godot Engine, що забезпечує модульність, розширюваність та простоту підтримки коду. Структура проєкту включає головну сцену main.tscn із централізованою ігровою логікою у скрипті main.gd, стартовий екран splash.tscn для професійної презентації застосунку та організовану ієрархію каталогів для графічних ресурсів, шрифтів та конфігураційних файлів. Система сигналів Godot використовується як основний механізм комунікації між компонентами, реалізуючи патерн «видавець-підписник» для забезпечення слабкої зв'язаності модулів та можливості незалежного розвитку окремих підсистем.

Математична модель ігрової економіки формалізована через експоненційну функцію зростання вартості генераторів із коефіцієнтом 1.12 та лінійну залежність продуктивності від кількості придбаних одиниць, що створює характерну для жанру динаміку з швидким початковим прогресом та поступовим сповільненням на пізніх етапах. Система персистентності даних реалізована на основі JSON-серіалізації з автоматичним та подієвим збереженням, валідацією завантажених даних, версіонуванням формату та механізмами відновлення при помилках, що гарантує надійне збереження прогресу гравця в усіх сценаріях використання. Розроблені проєктні рішення становлять повноцінну технічну специфікацію для практичної реалізації застосунку, що є предметом наступного розділу кваліфікаційної роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІГРОВОГО ПРОЄКТУ «CANDY CLICKER»

3.1. Вимоги до системи та вибір інструментальних засобів

Визначення вимог до «Candy Clicker» базується на аналізі специфіки жанру Idle/Clicker, можливостей Godot Engine та обмежень цільових платформ [28, с. 152–154]. Функціональні вимоги включають механіку активного кліку з візуальним зворотним зв'язком, систему генераторів пасивного доходу з шістьма рівнями прогресії, автоматичне збереження з періодичністю не рідше 30 секунд та при переході у фоновий режим, систему офлайн-прогресу з підсумковим діалогом, а також інтерфейс із відображенням балансу, показника CPS та панелі генераторів. Нефункціональні вимоги визначають частоту кадрів не нижче 30 FPS (60 FPS на флагманських пристроях), час запуску до 5 секунд, розмір пакету до 50 мегабайт та споживання RAM у межах 150–200 мегабайт. Сумісність охоплює Android 6.0+ (95% пристроїв), Windows та Linux з OpenGL 3.3+, HTML5 із WebGL 2.0 у Chrome, Firefox, Safari та Edge, а також адаптацію до роздільних здатностей від 480×800 до 2560×1440 пікселів.

Основним інструментом є Godot Engine 4.3 із вбудованим редактором коду, що забезпечує інтеграцію з системою вузлів, автодоповнення та індикацію помилок. Графічні ресурси підготовлені у форматі PNG, типографіка базується на шрифті Pusab із заокругленою естетикою кондитерської тематики, кольорова палітра - на пастельних тонах рожевого, бежевого та карамельного. Система контролю версій Git відстежує зміни у проєкті з виключенням директорії .godot через .gitignore, а .editorconfig забезпечує уніфікацію форматування коду [29-31].

Тестування включає ручну перевірку у редакторі Godot та тестування збірок на реальних пристроях: Android - через емулятор Android Studio та фізичні пристрої різних категорій, HTML5 - у різних браузерах для крос-браузерної сумісності, з особливою увагою до системи збереження у сценаріях примусового закриття та тривалих офлайн-періодів. Експорт налаштовано для кожної платформи: Android (версії SDK, дозволи, підпис APK, іконки), Windows (іконка,

метадані), HTML5 (оптимізація розміру, адаптивний canvas), причому всі шаблони дозволяють генерувати збірки безпосередньо з інтерфейсу Godot.

3.2. Реалізація основних ігрових модулів

Реалізація основних ігрових модулів застосунку «Candy Clicker» базується на головному скрипті main.gd, прикріпленому до кореневого вузла сцени main.tscn, який виконує роль координатора всіх ігрових підсистем - від ініціалізації змінних стану та обробки користувацького вводу до розрахунку пасивного доходу, управління транзакціями придбання генераторів та оновлення графічного інтерфейсу. Зосередження логіки в одному скрипті є прагматичним рішенням для проєкту даного масштабу, а модульність забезпечується внутрішньою структуризацією через функціональні блоки та систему сигналів Godot. Ініціалізація ігрового стану виконується у функції _ready(), де встановлюються посилання на дочірні вузли інтерфейсу через $NodePath, завантажується збережений прогрес через load_game() або ініціалізуються значення за замовчуванням, а також підключаються обробники сигналів - зокрема сигнал pressed кнопки кліку до функції _on_candy_clicked() та сигнал timeout таймера до функції _on_passive_timer_timeout().

Модуль обробки активного кліку реалізований у функції _on_candy_clicked(), яка при кожному натисканні на центральну кнопку із зображенням цукерки збільшує баланс на значення click_power, оновлює відповідний Label інтерфейсу та запускає візуальну анімацію масштабування кнопки через клас Tween - кнопка зменшується до 90% розміру і плавно повертається до нормального стану, створюючи відчутний тактильний відгук. Додатково при кожному кліку генерується плаваючий текстовий індикатор із кількістю отриманих цукерок, реалізований як динамічно створюваний вузол Label з анімацією руху вгору та поступового зникнення, який автоматично видаляється з дерева сцени після завершення анімації для запобігання витоку пам'яті.

Модуль пасивної генерації ресурсів забезпечує автоматичне накопичення цукерок відповідно до сумарної продуктивності всіх придбаних генераторів і є ключовим елементом, що трансформує «Candy Clicker» з простої гри на реакцію у повноцінний інкрементальний досвід. Технічна реалізація базується на вузлі Timer, налаштованому на інтервал спрацювання 0,1 секунди (100 мілісекунд), що забезпечує плавне візуальне оновлення лічильника балансу при високих показниках пасивної генерації замість ривкоподібного збільшення раз на секунду [32, с. 134]. При кожному спрацюванні таймера функція-обробник розраховує приріст ресурсів за формулою:

$$increment = total\_cps * delta\_time, \quad (3.1)$$

де total_cps є сумарною продуктивністю всіх генераторів у цукерках за секунду,

delta_time дорівнює інтервалу таймера, після чого цей приріст додається до поточного балансу та оновлюється відображення в інтерфейсі.

Важливим аспектом реалізації є використання числового типу float для зберігання балансу цукерок, що дозволяє коректно обробляти дробові значення приросту на ранніх етапах гри, коли продуктивність генераторів може становити частки одиниці за секунду, водночас відображення у інтерфейсі виконується з округленням до цілого числа для забезпечення читабельності та відповідності інтуїтивним очікуванням гравця щодо дискретної природи «цукерок» як ігрової валюти.

Модуль управління генераторами реалізує логіку придбання, обліку та розрахунку ефективності шести типів генераторів пасивного доходу, кожен з яких представлений записом у словнику GDScript із полями для базової вартості, продуктивності, коефіцієнта зростання та поточної кількості придбаних одиниць. Функція buy_generator(generator_id) виконує транзакцію придбання, яка включає перевірку достатності балансу для покупки за поточною вартістю,

віднімання вартості від балансу, збільшення лічильника придбаних одиниць, перерахунок вартості наступної одиниці за експоненційною формулою та оновлення сумарного показника пасивної генерації. Структура даних кожного генератора організована наступним чином, що наочно демонструє параметри ігрового балансу:

Таблиця 3.1 – Параметри реалізації генераторів у скрипті main.gd

Генератор	base_cost	base_cps	growth	Тип вузла UI
Льодяник	15	0,1	1,12	HBoxContainer
Карамель	100	0,5	1,12	HBoxContainer
Шоколадка	500	2,0	1,12	HBoxContainer
Тістечко	2 000	8,0	1,12	HBoxContainer
Торт	10 000	30,0	1,12	HBoxContainer
Кондитерська	50 000	100,0	1,12	HBoxContainer

Кожен рядок панелі генераторів у інтерфейсі формується динамічно при ініціалізації сцени через ітерацію по масиву конфігурацій генераторів, де для кожного елемента створюється HBoxContainer із дочірніми вузлами TextureRect для іконки солодощів, Label для назви та показників вартості й продуктивності, а також Button для здійснення покупки, причому стан кнопки придбання автоматично оновлюється при кожній зміні балансу через перевірку достатності коштів із відповідною зміною візуального стану від активного до неактивного, що надає гравцю миттєвий зворотний зв’язок про доступність тієї чи іншої покупки без необхідності натискати на кнопку та отримувати відмову (рис.3.1).

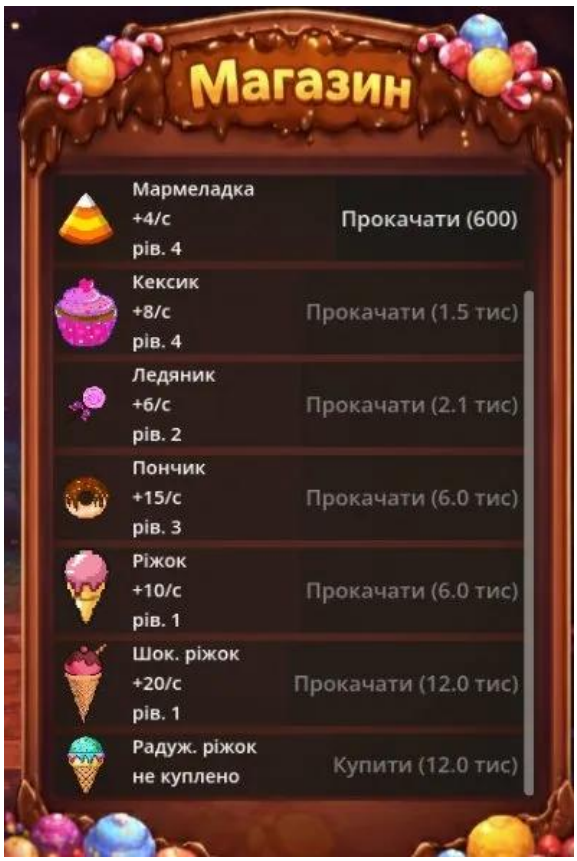

Рис. 3.1 – Система генераторів пасивного доходу на етапі активної прогресії гри

Модуль збереження та завантаження прогресу реалізує алгоритми персистентності даних, описані у підрозділі 2.4, через пару взаємодоповнюючих функцій save_game() та load_game(), що оперують файлами у директорії user:// з використанням класу FileAccess та нативної підтримки JSON у GDScript. Функція save_game() формує словник із поточним станом усіх ігрових змінних, включаючи баланс цукерок, масив кількостей придбаних генераторів, значення click_power, мітку часу збереження у форматі Unix timestamp через виклик Time.get_unix_time_from_system() та версію формату збереження, після чого серіалізує цей словник у JSON-рядок через JSON.stringify() та записує у файл savegame.json. Функція load_game() виконує зворотну операцію: відкриває файл для читання, парсить JSON-вміст через JSON.parse_string(), валідує наявність обов’язкових полів та відповідність типів даних, ініціалізує ігровий стан завантаженими значеннями та розраховує офлайн-прогрес на основі різниці між поточним часом і збереженою міткою часу. Особливу увагу приділено обробці

помилок: кожна операція файлового вводу-виводу перевіряється на успішність, при помилці парсингу JSON повертається порожній словник, а при відсутності окремих полів у старіших версіях збереження використовуються значення за замовчуванням, що забезпечує зворотну сумісність при оновленнях гри [33, с. 5470].

Модуль розрахунку офлайн-прогресу є специфічним компонентом, характерним для жанру Idle, що забезпечує нарахування ресурсів за період відсутності гравця та створює позитивний емоційний досвід повернення до застосунку. При завантаженні збереженого прогресу система витягує мітку часу останнього збереження та обчислює різницю з поточним системним часом у секундах, після чого застосовує формулу:

$$offline\_earnings = saved\_cps * elapsed\_seconds * offline\_multiplier, \quad (3.2)$$

де offline_multiplier встановлено на рівні 0,5 для збереження переваги активної гри над пасивним накопиченням.

Максимальний період офлайн-нарахування обмежений 86400 секундами (24 години) через конструкцію min(elapsed_seconds, 86400), що запобігає отриманню непропорційно великих сум при тривалих перервах та частково захищає від маніпуляцій із системним часом пристрою. Результат офлайн-нарахування відображається у спеціальному діалоговому вікні при запуску гри, що інформує гравця про кількість накопичених за час відсутності цукерок та тривалість офлайн-періоду, створюючи ефект приємного сюрпризу та мотивуючи регулярне повернення до застосунку.

Модуль управління анімаціями та візуальними ефектами базується на класі Tween, який у Godot 4.x забезпечує програмну інтерполяцію властивостей вузлів із підтримкою різних кривих прискорення, послідовного та паралельного виконання анімацій і автоматичного управління життєвим циклом. Анімація натискання на основну кнопку реалізована як послідовність двох твінів: перший

зменшує scale до Vector2(0.9, 0.9) протягом 0.05 секунди з кривою EASE_IN, другий повертає до Vector2(1.0, 1.0) протягом 0.1 секунди з кривою EASE_OUT, що створює ефект «пружинного» натискання. Плаваючі числові індикатори реалізовані через динамічне створення вузлів Label із одночасною анімацією зміщення по осі Y на 50 пікселів вгору та зменшення прозорості від 1.0 до 0.0 протягом 0.8 секунди, після чого вузол автоматично видаляється через виклик queue_free(). Модуль форматування великих чисел вирішує специфічну для інкрементальних ігор проблему відображення числових значень, що можуть сягати мільйонів, мільярдів та навіть більших порядків величини при тривалій грі, де стандартне числове представлення стає нечитабельним та виходить за межі відведеного для нього простору в інтерфейсі. Функція format_number(value) реалізує каскадну перевірку діапазону числа та повертає скорочений рядок із відповідним суфіксом: значення менше 1000 відображаються без змін, від 1000 до 999999 скорочуються з суфіксом «K» (кіло), від мільйона до мільярда із суфіксом «M» (мега), від мільярда до трильйона із суфіксом «B» (billion), а для ще більших значень використовуються суфікси «T», «Qa», «Qi» та інші відповідно до прийнятої у жанрі конвенції скорочення великих чисел (рис.3.2).

Рис. 3.2 – Відображення балансу цукерок із застосуванням скороченого формату великих чисел

Кожне скорочене значення відображається з одним десятковим знаком для збереження інформативності, наприклад 1 500 000 відображається як «1.5M», що забезпечує компактне, але достатньо точне уявлення про масштаб накопичених ресурсів та дозволяє гравцю відстежувати прогрес навіть при астрономічних значеннях балансу.

Модуль стартового екрану реалізований у окремій сцені splash.tscn із прикріпленим скриптом splash.gd та забезпечує професійну презентацію застосунку при його запуску, відображаючи логотип гри та назву з анімацією плавного з’явлення протягом перших секунд після старту програми, після чого автоматично здійснюється перехід до головної ігрової сцени через виклик get_tree().change_scene_to_file("res://main.tscn"). Стартовий екран виконує також функціональне призначення, маскуючи час, необхідний для ініціалізації ігрових ресурсів та завантаження збереженого прогресу на повільніших пристроях, що забезпечує сприйняття миттєвого запуску навіть при фактичній тривалості ініціалізації у кілька секунд [34, с. 9154]. Взаємодія між усіма описаними модулями організована за принципом однонаправленого потоку даних, де функції модифікації стану централізовано обробляються у головному скрипті, а оновлення інтерфейсу виконується як реакція на зміну стану через систему сигналів або безпосередні виклики функцій оновлення UI, що забезпечує передбачуваність поведінки системи та спрощує діагностику помилок при тестуванні.

3.3 Розробка графічного інтерфейсу користувача

Графічний інтерфейс користувача (GUI) ігрового застосунку «Candy Clicker» є основним каналом взаємодії гравця з ігровою системою, оскільки у жанрі Idle/Clicker, на відміну від ігор з тривимірним ігровим світом або складною фізичною симуляцією, весь ігровий досвід опосередковується виключно через елементи інтерфейсу, а тому якість їх проєктування та реалізації безпосередньо визначає рівень задоволення користувача та його бажання повертатися до гри (рис.3.3).

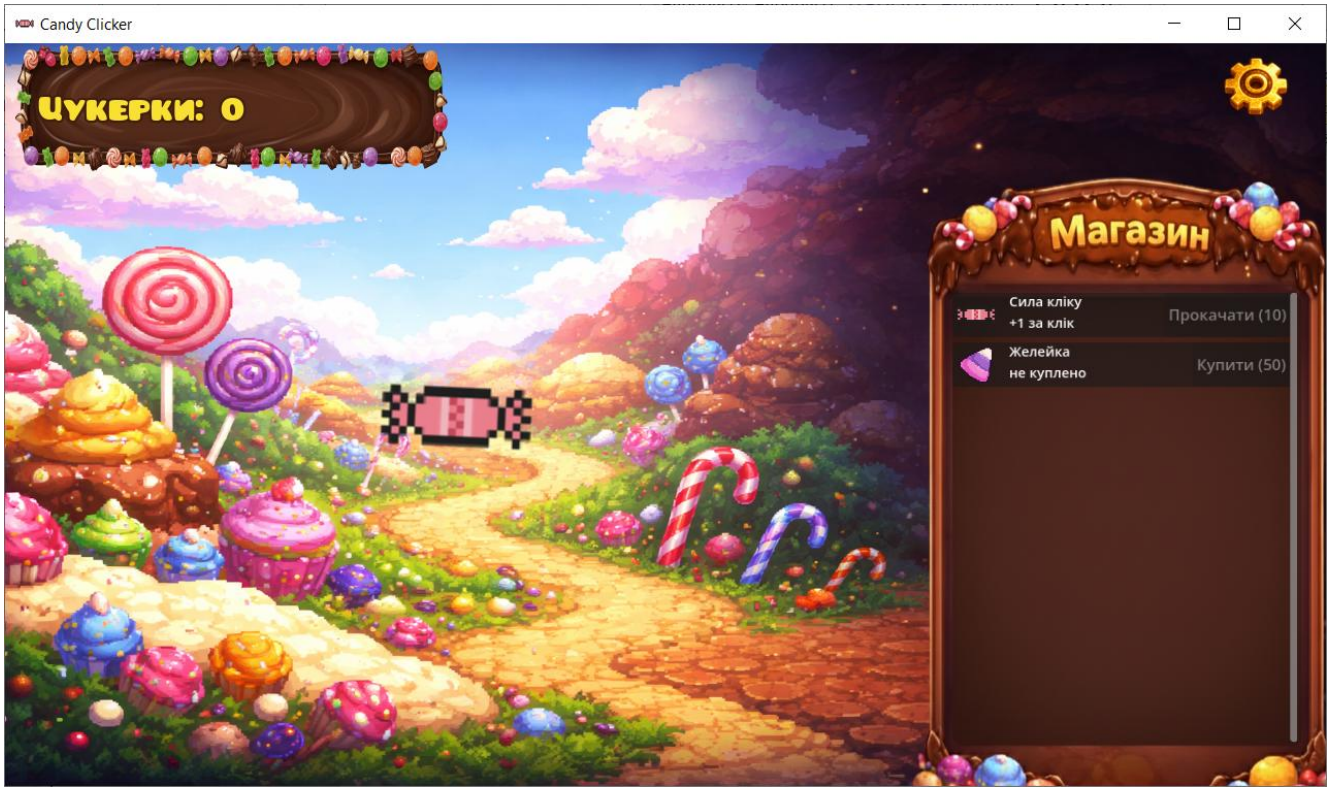

Рис. 3.3 - Загальний вигляд головного ігрового інтерфейсу застосунку «Candy Clicker» при першому запуску

Графічний інтерфейс користувача «Candy Clicker» побудований на ієрархії вузлів Control у Godot Engine, де кореневий вузол сцени main.tscn займає всю область вікна через налаштування якорів ANCHOR_BEGIN та ANCHOR_END, забезпечуючи автоматичне масштабування при зміні роздільної здатності. Базова роздільна здатність визначена як 720×1280 пікселів із режимом stretch_mode = canvas_items та stretch_aspect = keep_width, що гарантує збереження пропорцій на пристроях із різним співвідношенням сторін. Фонове оформлення реалізоване через TextureRect із налаштуванням STRETCH_KEEP_ASPECT_COVERED для повного заповнення екрана без спотворення, а поверх нього розміщено напівпрозорий ColorRect для забезпечення контрасту та читабельності текстових елементів.

Верхня панель інтерфейсу (VBoxContainer) містить Label балансу цукерок, оформлений шрифтом Pusab із контурною обводкою через LabelSettings, та Label показника пасивної генерації у форматі «X цукерок/сек», причому обидва елементи використовують функцію format_number() для скороченого

відображення великих значень із суфіксами K, M, B. Центральна область відведена під кнопку кліку, реалізовану як TextureButton із зображенням стилізованої цукерки, де стани натискання обробляються програмно через анімацію масштабування замість зміни текстури для більш плавного візуального зворотного зв'язку. Позиціонування кнопки здійснюється через якорі на центр екрана, а властивість ignore_texture_size забезпечує коректне масштабування зображення відповідно до доступного простору.

Таблиця 3.2 – Ієрархія вузлів GUI головної сцени main.tscn

Вузол	Тип	Батьківський	Призначення
Root	Control	–	Кореневий контейнер сцени
Background	TextureRect	Root	Фонове зображення
TopPanel	VBoxContainer	Root	Баланс та показник CPS
CandyButton	TextureButton	Root	Кнопка активного кліку
ShopPanel	ScrollContainer	Root	Панель генераторів
GeneratorList	VBoxContainer	ShopPanel	Список елементів магазину
FloatingText	Label	Root	Плаваючі індикатори кліку

Панель магазину генераторів реалізована як ScrollContainer із вертикальною прокруткою, усередині якого розміщено VBoxContainer (GeneratorList) з дочірніми елементами для кожного типу генератора, сформованими динамічно у функції _ready(). Кожен генератор представлений HBoxContainer із іконкою солодощів (TextureRect 64×64), двома Label для назви та характеристик, лічильником придбаних одиниць та кнопкою покупки з динамічно оновлюваною вартістю. Адаптивне масштабування забезпечується комбінацією глобального масштабування (canvas_items, keep_width) та локального позиціонування через систему якорів: верхня панель закріплена до верхнього краю, панель магазину - до нижнього, кнопка кліку - відносно центру [35, с. 5809].

Кольорова палітра побудована на теплих пастельних тонах кондитерської тематики (#F5E6CC, #FADADD) з акцентами карамельного (#D4A574) та шоколадного (#7B4B2A) кольорів, а текстові елементи відображаються білим із

темною контурною обводкою для читабельності на будь-якому фоні. Кнопки генераторів змінюють колір залежно від доступності покупки - зелений (#4CAF50) при достатньому балансі та сірий (#9E9E9E) при недостатніх коштах через модифікацію властивості modulate. Типографіка базується на шрифті Pusab із диференційованим розміром: 48–64 пікселі для балансу, 24–32 для показника CPS, 18–24 для характеристик генераторів. Інтерактивні елементи спроєктовані з урахуванням мінімальної області дотику 48×48 пікселів згідно з рекомендаціями Material Design, а ScrollContainer забезпечує інерційну прокрутку жестом проведення на сенсорних екранах та колесом миші на десктопних платформах.

Реалізація візуальних переходів та станів інтерфейсу забезпечує плавну зміну відображення елементів при значущих ігрових подіях, що створює відчуття живості та інтерактивності застосунку замість механічного оновлення статичних числових показників (рис.3.4).

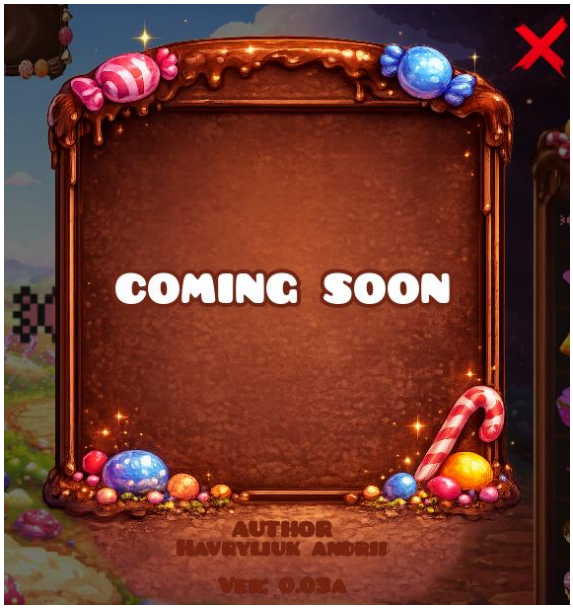

Рис. 3.4 – Модальне вікно налаштувань із зазначенням версії застосунку та автора

При придбанні нового генератора відповідний елемент у панелі магазину виконує коротку анімацію підсвічування через зміну кольору фонової панелі від базового до яскравішого відтінку з подальшим плавним поверненням,

реалізовану через клас Tween із інтерполяцією властивості modulate вузла PanelContainer. Оновлення лічильника кількості придбаних генераторів супроводжується анімацією масштабування відповідного Label від нормального розміру до збільшеного та назад, що акцентує увагу гравця на зміні та підтверджує успішність транзакції. Діалог відображення офлайн-прогресу при запуску гри реалізований як окремий вузол PanelContainer із центрованим розташуванням на екрані, напівпрозорим затемненням фону та послідовною анімацією появи тексту з інформацією про тривалість офлайн-періоду та кількість нарахованих цукерок, що створює ефект приємного сюрпризу та мотивує регулярне повернення до застосунку [36, с. 3403]. Перехід між стартовим екраном splash.tscn та головною ігровою сценою main.tscn реалізований через плавне затухання зображення стартового екрана з подальшим з’явленням ігрового інтерфейсу, що маскує момент завантаження ресурсів та ініціалізації ігрового стану і створює професійне враження від продукту.

3.4 Тестування та оптимізація продуктивності

Тестування «Candy Clicker» організовано за багаторівневою методологією, що охоплює модульне, інтеграційне, системне та UX-тестування. Модульне тестування зосереджене на верифікації математичних розрахунків ігрової економіки: функція calculate_cost() перевіряється порівнянням із еталонними значеннями для граничних випадків (нульова кількість одиниць, великі значення лічильника з контролем переповнення float), функція total_cps - на точність при малих значеннях продуктивності (0.1 цукерки/сек), а format_number() - на повному діапазоні від одиниць до трильйонів, включаючи граничні переходи між діапазонами [37-39].

Тестування системи збереження включає перевірку save_game() на створення файлу у user:// з валідною JSON-структурою та коректною міткою Unix timestamp, а також load_game() у множині сценаріїв: завантаження коректного файлу, відсутність файлу (ініціалізація за замовчуванням),

пошкоджений JSON (обробка без аварійного завершення) та старіша версія формату (міграція даних). Офлайн-прогрес тестується через штучну модифікацію мітки часу з перевіркою формули offline_earnings = saved_cps × elapsed_seconds × 0.5 та обмеження 86400 секундами.

Інтеграційне тестування перевіряє повний цикл придбання генератора - від натискання кнопки до зменшення балансу, збільшення лічильника, перерахунку вартості за експоненційною формулою, оновлення CPS та відображення змін в інтерфейсі протягом одного кадру. Тестується відмова транзакції при недостатньому балансі без змін стану, а також коректність ланцюга подій при зміні балансу від пасивного таймера - оновлення відображення, перевірка доступності кнопок та ініціювання автозбереження.

Таблиця 3.3 – Результати тестування на цільових платформах

Платформа	FPS	RAM, МБ	Запуск, с	Збірка, МБ	Статус
Windows 10/11	60	85–110	1,2–1,8	28	Пройдено
Linux (Ubuntu 22.04)	60	80–105	1,0–1,5	30	Пройдено
Android 10+ (середній)	55–60	120–160	2,5–3,5	22	Пройдено
Android 8 (бюджетний)	30–45	140–180	3,5–5,0	22	Пройдено
HTML5 (Chrome)	58–60	–	2,0–3,0	15	Пройдено

Системне тестування на цільових платформах підтвердило відповідність застосунку визначеним нефункціональним вимогам: частота кадрів стабільно утримується на рівні 60 FPS на настільних системах та 55–60 FPS на мобільних пристроях середнього сегмента, споживання пам'яті залишається в межах 85–180 мегабайт, час запуску не перевищує 5 секунд на жодній конфігурації, а розмір збірок не перевищує 30 мегабайт для настільних платформ та 22 мегабайти для Android. UX-тестування, проведене шляхом спостереження за групою користувачів без попередніх інструкцій, засвідчило інтуїтивну зрозумілість базової механіки кліку, хоча частина тестувальників потребувала додаткового

часу для усвідомлення системи генераторів. На основі зворотного зв'язку було збільшено розмір та контрастність кнопок придбання, додано текстову підказку при першому досягненні достатнього балансу та посилено візуальну відмінність між активними та неактивними станами елементів магазину [40, с. 575–585].

Оптимізація продуктивності виконувалася за допомогою вбудованого профайлера Godot (Debugger → Monitors), який виявив, що основним джерелом навантаження є частота оновлення текстових елементів Label, а не ігрова логіка. Для мінімізації застосовано стратегію дискретизації оновлень - текст балансу оновлюється лише при зміні відображуваного значення після format_number(). Управління пам'яттю забезпечується видаленням плаваючих індикаторів через queue_free() у callback завершення Tween та додатковим примусовим очищенням при перевищенні порогу у 50 одночасних вузлів. Для елементів інтерфейсу вимкнено генерацію міп-рівнів та застосовано Lossless-стиснення текстур.

Оптимізація енергоспоживання реалізована через налаштування Engine.max_fps на 30 FPS для мобільних платформ та 60 FPS для настільних, зниження Physics Ticks Per Second до мінімуму (фізичний двигун не використовується) та автоматичне призупинення таймерів при переході у фоновий режим [41-43]. Підсумовуючи, застосунок відповідає всім визначеним вимогам: модульне тестування підтвердило коректність математичних розрахунків економіки, тестування персистентності верифікувало надійність збереження в усіх сценаріях, а комплекс оптимізаційних рішень забезпечив ефективне використання ресурсів та мінімальне енергоспоживання на мобільних пристроях.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи здійснено практичну реалізацію ігрового застосунку «Candy Clicker» на базі рушія Godot Engine версії 4.3 з використанням мови програмування GDScript, що охопила повний цикл розробки від визначення системних вимог та налаштування інструментального середовища до тестування готових збірок на всіх цільових платформах та

оптимізації продуктивності для забезпечення комфортного користувацького досвіду.

Визначено функціональні та нефункціональні вимоги до застосунку, що включають реалізацію механіки активного кліку, системи шести типів генераторів пасивного доходу, автоматичного збереження прогресу з періодичністю не рідше 30 секунд, розрахунку офлайн-накопичень та адаптивного графічного інтерфейсу. Нефункціональні вимоги визначають стабільну частоту кадрів не нижче 30 FPS, споживання оперативної пам’яті до 200 мегабайт, час запуску до 5 секунд та розмір інсталяційного пакету до 50 мегабайт. Обґрунтовано вибір інструментальних засобів розробки, зокрема вбудованого редактора Godot як основного середовища написання скриптів, формату PNG для растрових ресурсів, шрифту Pusab для типографічного оформлення та системи контролю версій Git для відстеження змін у проєкті.

Реалізовано основні ігрові модулі застосунку: модуль ініціалізації стану у функції _ready() із завантаженням збереженого прогресу та підключенням обробників сигналів; модуль обробки активного кліку з анімацією масштабування кнопки через клас Tween та генерацією плаваючих текстових індикаторів; модуль пасивної генерації ресурсів на основі вузла Timer з інтервалом 0,1 секунди для плавного оновлення балансу; модуль управління генераторами з реалізацією транзакцій придбання та експоненційним перерахунком вартості; модуль збереження та завантаження прогресу на основі JSON-серіалізації з валідацією даних, обробкою помилок та створенням резервних копій; модуль розрахунку офлайн-прогресу з обмеженням максимального періоду 24 годинами та коефіцієнтом 0,5 від активної продуктивності; а також модуль форматування великих чисел із каскадною системою суфіксів K, M, B, T для компактного відображення значень будь-якого порядку.

Розроблено графічний інтерфейс користувача на основі ієрархії вузлів Control із використанням системи якорів та контейнерів Godot для забезпечення адаптивного масштабування на екранах різної роздільної здатності від 480×800

до 2560×1440 пікселів. Інтерфейс включає верхню панель статистики з відображенням балансу та показника пасивної генерації, центральну кнопку кліку на основі TextureButton із анімованим зворотним зв’язком, та панель магазину генераторів із вертикальною прокруткою через ScrollContainer. Кольорова палітра побудована на теплих пастельних тонах кондитерської тематики, а інтерактивні елементи спроєктовані з урахуванням мінімальної області дотику 48×48 пікселів для комфортного використання на сенсорних екранах мобільних пристроїв.

Проведено комплексне тестування застосунку, що охопило модульне тестування математичних функцій ігрової економіки, тестування системи збереження та завантаження прогресу у множині сценаріїв включаючи пошкоджені файли та старіші версії формату, інтеграційне тестування взаємодії між модулями, системне тестування на платформах Windows, Linux, Android та HTML5, а також UX-тестування для оцінки інтуїтивності інтерфейсу. Результати підтвердили відповідність застосунку всім визначеним вимогам: частота кадрів становить 60 FPS на настільних системах та 30–60 FPS на мобільних пристроях, споживання пам’яті не перевищує 180 мегабайт, час запуску складає від 1,0 до 5,0 секунд залежно від платформи, а розмір збірки не перевищує 30 мегабайт.

Застосовано комплекс оптимізаційних рішень для забезпечення ефективного використання системних ресурсів: дискретизацію оновлень текстових елементів інтерфейсу для зменшення навантаження на рендерер, управління динамічними вузлами плаваючих індикаторів із примусовим очищенням при перевищенні порогового значення, оптимізацію імпорту текстур із вимкненням непотрібних міп-рівнів, налаштування цільової частоти кадрів 30 FPS для мобільних платформ та зниження частоти оновлення фізичного двигуна. Сукупність цих заходів забезпечила мінімальне енергоспоживання на мобільних пристроях, що є важливим фактором для позитивного сприйняття застосунку користувачами та його конкурентоспроможності на платформах цифрової дистрибуції.

ВИСНОВКИ

У кваліфікаційній роботі здійснено повний цикл розробки ігрового застосунку «Candy Clicker» жанру Idle/Clicker на базі рушія Godot Engine версії 4.3 з використанням мови програмування GDScript, що охопив теоретичне дослідження предметної області, проєктування архітектури та ігрових механік, практичну реалізацію програмного коду, розробку графічного інтерфейсу користувача, а також комплексне тестування та оптимізацію продуктивності на цільових платформах Windows, Linux, Android та HTML5.

У першому розділі проведено аналіз жанру Idle/Clicker ігор, що дозволив виявити ключові механіки залучення гравців - активний клік, пасивну генерацію ресурсів, систему покращень та офлайн-прогресію, а також визначити психологічні принципи, що забезпечують тривале утримання аудиторії, зокрема теорію потоку Чіксентміхаї та механізми варіативного підкріплення. Порівняльний аналіз сучасних ігрових рушіїв для двовимірної розробки - Unity, Unreal Engine, Godot Engine, GameMaker, Defold та Construct - дозволив обґрунтувати вибір Godot як оптимального інструменту для реалізації проєкту завдяки відкритому вихідному коду з ліцензією MIT, відсутності роялті та фінансових зобов'язань, компактному розміру дистрибутива, вбудованій мові GDScript із Python-подібним синтаксисом, потужній системі вузлів та сцен для компонентної архітектури, а також нативній підтримці експорту на всі цільові платформи. Досліджено архітектурні особливості Godot Engine, зокрема ієрархічну систему вузлів, механізм сигналів для слабкозв'язаної комунікації між компонентами, систему ресурсів та інструменти побудови графічних інтерфейсів на основі класу Control.

У другому розділі спроєктовано концепцію гри в кондитерській тематиці з оригінальною візуальною стилістикою та шістьма типами генераторів пасивного доходу - Желейка, Мармеладка, Кексик, Ледяник, Пончик та Ріжок - кожен з яких має унікальну базову вартість, продуктивність та візуальну репрезентацію. Розроблено архітектуру програмного забезпечення на основі компонентного

підходу із централізованим головним скриптом main.gd, системою сигналів для комунікації між модулями та словником Dictionary для управління глобальним станом. Формалізовано математичну модель ігрової економіки з експоненційною функцією вартості генераторів $C(n) = C_0 \times r^n$ із коефіцієнтом зростання $r = 1.12$ та лінійною продуктивністю $P(n) = P_0 \times n$, що забезпечує збалансований темп прогресії із поступовим уповільненням на пізніх етапах гри. Спроєктовано систему персистентного збереження на основі JSON-серіалізації з записом у директорію user://, механізмом резервного копіювання, валідацією даних за принципом graceful degradation, версіонуванням формату та стратегією автозбереження, що поєднує періодичний (кожні 30 секунд) та подієвий підходи. Розроблено алгоритм розрахунку офлайн-прогресу з формулою offline_earnings = saved_cps × elapsed_seconds × 0.5 та обмеженням максимального періоду 86400 секундами (24 години).

У третьому розділі здійснено практичну реалізацію всіх спроєктованих систем. Визначено функціональні та нефункціональні вимоги до застосунку, обґрунтовано вибір інструментальних засобів та налаштовано середовище розробки з системою контролю версій Git. Реалізовано основні ігрові модулі: ініціалізація стану у _ready() із завантаженням збереженого прогресу; обробка активного кліку з анімацією масштабування через Tween та генерацією плаваючих індикаторів; пасивна генерація на основі Timer з інтервалом 0.1 секунди; управління генераторами з транзакціями придбання та експоненційним перерахунком вартості; збереження та завантаження через JSON.stringify()/JSON.parse_string() із FileAccess; форматування великих чисел із каскадною системою суфіксів K, M, B, T. Розроблено адаптивний графічний інтерфейс на ієрархії вузлів Control із системою якорів та контейнерів для масштабування від 480×800 до 2560×1440 пікселів, що включає верхню панель статистики, центральну кнопку TextureButton, панель магазину генераторів на ScrollContainer та модальні діалоги. Кольорова палітра побудована на теплих пастельних тонах кондитерської тематики, типографіка базується на шрифті

Pusab, а інтерактивні елементи спроєктовані з мінімальною областю дотику 48×48 пікселів згідно з рекомендаціями Material Design.

Комплексне тестування підтвердило відповідність застосунку всім визначеним вимогам: частота кадрів становить 60 FPS на настільних системах та 30–60 FPS на мобільних пристроях, споживання пам'яті не перевищує 180 мегабайт, час запуску - від 1.0 до 5.0 секунд, розмір збірки - до 30 мегабайт. Модульне тестування верифікувало коректність математичних розрахунків економіки, тестування персистентності підтвердило надійність збереження в усіх сценаріях, а UX-тестування дозволило покращити інтуїтивність інтерфейсу. Застосовано комплекс оптимізаційних рішень: дискретизація оновлень текстових елементів, управління динамічними вузлами з примусовим очищенням, налаштування Engine.max_fps на 30 FPS для мобільних платформ, вимкнення міп-рівнів та оптимізація фізичного двигуна.

Таким чином, усі поставлені завдання кваліфікаційної роботи виконано в повному обсязі. Результатом є готовий до розповсюдження ігровий продукт «Candy Clicker», який може бути опублікований на платформах цифрової дистрибуції для Windows, Linux, Android та у веб-браузерах через HTML5-збірку. Систематизований досвід застосування Godot Engine для розробки ігор жанру Idle/Clicker, включаючи архітектурні рішення, методи балансування економіки та оптимізації продуктивності, може бути використаний як практичний посібник для розробників аналогічних проєктів та як навчальний матеріал у курсах ігрової розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Buergi J. Idle Yet Engaged: How Idle Games Satisfy Our Needs for Competence and Autonomy : diss. Rensselaer Polytechnic Institute, 2025. 120 p. URL: https://www.proquest.com/openview/daa4dd324eab2af807e492f8c945122a/1?pq-origsite=gscholar&cbl=18750&diss=y

2. Buergi J. Idle Games: A Cozy Genre Turned Exploitative. Replay. The Polish Journal of Game Studies. 2024. Vol. 12, No. 1. P. 73–86. URL: https://www.ceeol.com/search/article-detail?id=1298539

3. Howard J. Quests: Design, Theory, and History in Games and Narratives. AK Peters/CRC Press, 2022. 312 p.

4. Fortugno N. Designing for "Everyone": The Triumph of Casual Mechanics. Game Usability. CRC Press, 2022. P. 339–349. URL: https://www.taylorfrancis.com/chapters/edit/10.1201/9781003109389-36/designing-everyone-triumph-casual-mechanics-nicholas-fortugno

5. Heidari A. Gaming with Emojis: A Look at Different Strategies of Emoji Inclusion in the Design of Digital Games. Acta Ludologica. 2024. Vol. 7, No. 2. P. 38–51. URL: https://www.ceeol.com/search/article-detail?id=1290939

6. DeJong S., Blamey C. Top Shelf Drinks, Bottom Line Play: Examining Representations of Class in Bartending and Mixology Games. Games and Culture. 2023. Vol. 18, No. 5. P. 622–642. URL: https://journals.sagepub.com/doi/epub/10.1177/15554120221119962

7. Яценко О. І., Нечипорук В. О. Огляд переваг та недоліків ігрових рушіїв. 2024. С. 86–89. URL: https://eprints.zu.edu.ua/40326/1/4444.pdf

8. Майданюк В. П., Складанюк О. О. Графічний редактор для редагування десктопних відеоігор. Вісник Вінницького політехнічного інституту. 2025. № 3. С. 36. URL: https://www.tech.vernadskyjournals.in.ua/journals/2025/3_2025/part_2/36.pdf

9. Мельник П. П., Василенко Я. П. Технологічні аспекти використання рушія Unity для розробки гри-квесту. Збірник наукових праць. Тернопіль, 2024.

С. 169. URL: https://lib.iitta.gov.ua/id/eprint/740555/1/Збірник_Тернопіль_2024.pdf#page=169

10. Gazis A., Katsiri E. Serious Games in Digital Gaming: A Comprehensive Review of Applications, Game Engines and Advancements. arXiv preprint arXiv:2311.03384. 2023. URL: https://arxiv.org/abs/2311.03384

11. Mohd T. K., et al. Analyzing Strengths and Weaknesses of Modern Game Engines. International Journal of Computer Theory and Engineering. 2023. Vol. 15, No. 1. P. 54–60. URL: https://www.researchgate.net/profile/Tauheed-Khan-Mohd/publication/368590412_Analyzing_Strengths_and_Weaknesses_of_Modern_Game_Engines/links/656a0c70ce88b8703127f517/Analyzing-Strengths-and-Weaknesses-of-Modern-Game-Engines.pdf

12. Sharma V., et al. Game Engines: A Survey. AIP Conference Proceedings. 2024. Vol. 3214, No. 1. P. 020060. URL: https://pubs.aip.org/aip/acp/article-abstract/3214/1/020060/3318725/Game-engines-A-survey

13. González Ramírez L. Development of a 2D Engine : thesis. Universitat Politècnica de Catalunya, 2025. URL: https://upcommons.upc.edu/entities/publication/c01aacac-b13d-45ae-87c2-cc61f24fb039

14. Sobota B., Pietriková E. The Role of Game Engines in Game Development and Teaching. Computer Science for Game Development and Game Development for Computer Science. IntechOpen, 2023. URL: https://www.intechopen.com/chapters/1162246

15. Adeniyi A. E., et al. Development of Two Dimension (2D) Game Engine with Finite State Machine (FSM) Based Artificial Intelligence (AI) Subsystem. Procedia Computer Science. 2024. Vol. 235. P. 2996–3006. URL: https://www.sciencedirect.com/science/article/pii/S1877050924009621

16. Mustonen M. Web Based Game Engine Design : thesis. Lappeenranta–Lahti University of Technology LUT, 2023. URL: https://lutpub.lut.fi/bitstream/handle/10024/166230/Web%20Based%20Game%20Engine%20Design.pdf?sequence=1&isAllowed=y

17. Мельник П. П., Василенко Я. П. Особливості використання рушія Godot та C# для розробки ігрових застосунків. Збірник наукових праць. Тернопіль, 2025. С. 156. URL: https://lib.iitta.gov.ua/id/eprint/745238/1/Збірник_Тернопіль_2025.pdf#page=156

18. Гнатюк С. Технології та засоби розробки відеоігор. Збірник наукових праць Житомирського державного університету імені Івана Франка. 2022. С. 239. URL: https://lib.iitta.gov.ua/id/eprint/730060/1/Житомир_збірник_2022.pdf#page=240

19. Salmela T. Game Development Using the Open-Source Godot Game Engine : thesis. 2022. URL: https://www.theseus.fi/bitstream/handle/10024/746943/Salmela_Tero.pdf?sequence=3&isAllowed=y

20. Vanhove S. Learning GDScript by Developing a Game with Godot 4: A Fun Introduction to Programming in GDScript 2.0 and Game Development Using the Godot Engine. Packt Publishing, 2024. 350 p. URL: https://books.google.com.ua/books?id=x_0CEQAAQBAJ

21. Vuorela J. Differences Between C# and GDScript in Godot Game Engine : thesis. 2024. URL: https://www.theseus.fi/bitstream/handle/10024/874264/Vuorela_Jesse.pdf?sequence=2&isAllowed=y

22. Henning R. Godot 4 for Beginners: Develop Engaging 2D and 3D Games with Godot 4's Scripting and Design Features. Packt Publishing, 2025. URL: https://books.google.com.ua/books?id=qh51EQAAQBAJ

23. Thorn A. Game Development with Godot 4: A Complete Introduction. CRC Press, 2025. URL: https://books.google.com.ua/books?id=Kkt_EQAAQBAJ

24. Felicia P. Godot From Zero to Proficiency (Beginner). Vol. 2. Patrick Felicia, 2021. URL: https://books.google.com.ua/books?id=-DIqEAAAQBAJ

25. Alybaev A. Comparative Analysis of Unity and Godot for 2D Game Development. Preprints. 2025. URL: https://www.preprints.org/manuscript/202511.1981

26. Felicia P. Godot From Zero to Proficiency (Beginner). Vol. 2. Patrick Felicia, 2021. URL: https://books.google.com.ua/books?id=-DIqEAAAQBAJ

27. Son N. Making a 2D Game with Godot 4 Engine : thesis. 2024. URL: https://www.theseus.fi/bitstream/handle/10024/862142/Son_Nguyen.pdf?sequence=4&isAllowed=y

28. Abinaya P. 3D Puzzle Game Using Godot Engine. International Journal of Engineering Development and Research. 2021. Vol. 9, No. 2. P. 152–154. URL: https://rjwave.org/ijedr/viewpaperforall.php?paper=IJEDR2102023

29. Johnson J. Godot 4 Game Development Cookbook: Over 50 Solid Recipes for Building High-Quality 2D and 3D Games with Improved Performance. Packt Publishing, 2023. URL: https://books.google.com.ua/books?id=FNS-EAAAQBAJ

30. Truong C., Nilsson A. A Visual System for Simplifying Exergame Development : thesis. 2025. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1973325&dswid=2390

31. Shafeah M., Bayat R. Designing Micro Exertion Game for 2-Minute Active Breaks with ML-Agent-Based Skeletal Tracking Act the Godot Engine : thesis. 2023. URL: https://www.diva-portal.org/smash/record.jsf?dswid=2390&pid=diva2%3A1778812

32. Царук П. Р., Іванців Б. Я. Програмна бібліотека алгоритмів обробки та розпізнавання цифрових зображень. ІКСМ весна 2025. С. 134. URL: https://ki.wunu.edu.ua/conference/archive/2025_1.pdf#page=134

33. Nanthapodej R., et al. Hybrid Differential Evolution Algorithm and Adaptive Large Neighborhood Search to Solve Parallel Machine Scheduling to Minimize Energy Consumption in Consideration of Machine-Load Balance Problems. Sustainability. 2021. Vol. 13, No. 10. P. 5470. URL: https://www.mdpi.com/2071-1050/13/10/5470

34. Li P., Cao J. A Virtual Machine Consolidation Algorithm Based on Dynamic Load Mean and Multi-Objective Optimization in Cloud Computing. Sensors. 2022. Vol. 22, No. 23. P. 9154. URL: https://www.mdpi.com/1424-8220/22/23/9154

35. Ma P., et al. Review of Family-Level Short-Term Load Forecasting and Its Application in Household Energy Management System. Energies. 2023. Vol. 16, No. 15. P. 5809. URL: https://www.mdpi.com/1996-1073/16/15/5809

36. Flores J. L. G., Cieza-Mostacero S. E. Artificial Intelligence in Video Game Development with Accessibilities in Godot Engine. TEM Journal. 2025. Vol. 14, No. 4. P. 3403. URL: https://www.ceeol.com/search/article-detail?id=1384102

37. Mäkelä H. Development of a 3D Mahjong Video Game in Godot Engine : thesis. 2021. URL: https://www.theseus.fi/bitstream/handle/10024/502957/thesis_makela_henri.pdf?sequence=2&isAllowed=y

38. Alchimowicz S., Plechawska-Wójcik M. Analiza porównawcza wydajności implementacji wykorzystującej wybrane języki skryptowe w silniku gier Godot. Journal of Computer Sciences Institute. 2024. Vol. 31. URL: https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-720c5ab9-06ce-4725-b6dc-c4304577e58e

39. Pablo R. Développement ergonomique d'un support de formation mobilisant les outils de l'industrie du futur : diss. Université de Lorraine, 2023. URL: https://docnum.univ-lorraine.fr/public/DDOC_T_2023_0181_PABLO.pdf

40. Нагорнюк О., Дрозд А. Метод оптимізації продуктивності та захищеності кіберфізичних систем реального часу на основі балансування завдань і ресурсів. Herald of Khmelnytskyi National University. Technical Sciences. 2026. Vol. 361, No. 1. P. 575–585. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/2448/2399

41. Molina J. B., et al. BECKETT – A Flight Dynamics System Based on GODOT. Proceedings of the International Symposium on Space Flight Dynamics (ISSFD). 2024. URL: https://issfd.org/ISSFD_2024/ISSFD2024_7-1.pdf

42. Gago P., et al. RODDAS: Robust Orbit Determination and Data Association Library for SSA Based on GODOT. Proceedings of the 9th Space Debris Conference. ESA, 2024. URL: https://conference.sdo.esoc.esa.int/proceedings/sdc9/paper/311/SDC9-paper311.pdf

43. Chappe N., et al. An Optimised Flow for Futures: From Theory to Practice. arXiv preprint arXiv:2107.07298. 2021. URL: https://arxiv.org/abs/2107.07298

ДОДАТКИ

Додаток А

```
extends Node2D

## Candy Clicker — основна логіка гри
## Candy1 у центрі: клік дає цукерки, прокачка — більше за клік.
## Candy2–6 у магазині: купуєш/прокачуєш — дають пасивний дохід.

# Цукерки (валюта)
var candy_count: float = 0.0

# Candy1: сила кліку (скільки дає один клік)
var candy1_level: int = 1
var candies_per_click: int = 1

# Candy2–11: рівень 0 = не куплено, 1+ = куплено і дає пасивний дохід
var candy_levels: Array[int] = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# Пасивний дохід за секунду (сума від усіх Candy2–11)
var passive_income_per_second: float = 0.0

# Базові ціни
const CANDY1_UPGRADE_BASE: int = 10
const CANDY2_BASE: int = 50
const CANDY3_BASE: int = 120
const CANDY4_BASE: int = 300
const CANDY5_BASE: int = 700
const CANDY6_BASE: int = 1500
const CANDY7_BASE: int = 3000
```

```
const CANDY8_BASE: int = 6000
const CANDY9_BASE: int = 12000
const CANDY10_BASE: int = 24000
const CANDY11_BASE: int = 48000

# Пасивний дохід за рівень (цукерок/с за одну покупку) — Candy2..Candy11
const PASSIVE_RATE: Array[float] = [0.5, 1.0, 2.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0, 160.0]

# Назви цукерок у магазині (Candy2..Candy11)
const CANDY_NAMES: Array[String] = ["Желейка", "Мармеладка", "Кексик", "Ледяник", "Пончик", "Ріжок", "Шок. ріжок", "Радуж. ріжок", "Стаканчик", "Пломбір"]

# Шлях збереження (user:// — папка даних гри на пристрої)
const SAVE_PATH: String = "user://candy_clicker_save.json"
const AUTOSAVE_INTERVAL: float = 30.0
# Максимум офлайн-часу для нарахування (7 днів), щоб уникнути переповнення при зміні годинника
const MAX_OFFLINE_SECONDS: float = 7.0 * 24.0 * 3600.0
var _autosave_timer: float = 0.0
```

Додаток Б

```
# Вузли
@onready var candy_button: TextureButton = $CanvasLayer/CandyButton
@onready var label_candy: Label = $CanvasLayer/TopBar/Margin/CandyLabel
@onready var shop_scroll: ScrollContainer = $CanvasLayer/ShopPanel/Margin/ScrollContainer
@onready var shop_vbox: VBoxContainer = $CanvasLayer/ShopPanel/Margin/ScrollContainer/VBox
```

```
@onready var click_particles_template: CPUParticles2D = $CanvasLayer/ClickParticles
@onready var settings_button: TextureButton = $CanvasLayer/SettingsButton
@onready var settings_overlay: Control = $CanvasLayer/SettingsOverlay
@onready var settings_backdrop: Button = $CanvasLayer/SettingsOverlay/BackdropButton
@onready var settings_close_button: TextureButton = $CanvasLayer/SettingsOverlay/Center/WindowBox/CloseButton

# Пул емітерів частинок: кожен клік запускає свій вибух, попередні не перериваються
var click_particles_pool: Array[CPUParticles2D] = []
var _click_particles_index: int = 0
const CLICK_PARTICLES_POOL_SIZE: int = 12

# Кнопки, лейбли та рядки магазину (Candy1..Candy11)
var shop_buttons: Array[Button] = []
var shop_labels: Array[Label] = []
var shop_rows: Array[Control] = []

# Анімація центральної цукерки
var _idle_tween: Tween
const IDLE_SWAY_ANGLE: float = 0.045  # радіани (~2.5°)
const IDLE_SWAY_DURATION: float = 1.4

# Drag-to-scroll магазину: затиснув і потягнув вгору/вниз (миша або палець)
var _shop_drag_pos: Vector2 = Vector2.ZERO
var _shop_drag_active: bool = false
const SHOP_DRAG_THRESHOLD: float = 10.0
```

```
func _ready() -> void:
	_load_game()
	_build_click_particles_pool()
	candy_button.pressed.connect(_on_candy_clicked)
	settings_button.pressed.connect(_on_settings_button_pressed)
	settings_backdrop.pressed.connect(_on_settings_backdrop_pressed)
	settings_close_button.pressed.connect(_on_settings_close_pressed)

	# Збираємо рядки, кнопки та лейбли з магазину (Candy1Row .. Candy11Row)
	for i in range(1, 12):
		var row = shop_vbox.get_node_or_null("Candy%dRow" % i)
		if row and row is Control:
			shop_rows.append(row as Control)
			var hbox = row.get_node_or_null("HBox")
			if hbox:
				var lbl = hbox.get_node_or_null("Label")
				var btn = hbox.get_node_or_null("Button")
				if lbl:
					shop_labels.append(lbl)
				if btn:
					shop_buttons.append(btn)

	btn.pressed.connect(_on_shop_button_pressed.bind(i))
	_update_passive()
	_update_ui()
	_start_candy_idle_animation()
```

```
func _input(event: InputEvent) -> void:
	# Drag-to-scroll магазину: затиснув у списку і потягнув вгору/вниз (миша або палець)
	if not shop_scroll.visible or not shop_scroll.get_global_rect().has_point(_shop_get_event_position(event)):
		if event is InputEventMouseButton or (event is InputEventScreenTouch and not (event as InputEventScreenTouch).pressed):
			_shop_drag_active = false
		return
	# Миша
	if event is InputEventMouseButton:
		var mb := event as InputEventMouseButton
		if mb.button_index == MOUSE_BUTTON_LEFT:
			if mb.pressed:
				_shop_drag_pos = mb.global_position
				_shop_drag_active = false
			else:
				if _shop_drag_active:
					get_viewport().set_input_as_handled()
				_shop_drag_active = false
		return
	if event is InputEventMouseMotion:
		var mm := event as InputEventMouseMotion
		if mm.button_mask & MOUSE_BUTTON_MASK_LEFT:
			if not _shop_drag_active and _shop_drag_pos.distance_to(mm.global_position) > SHOP_DRAG_THRESHOLD:
				_shop_drag_active = true
			if _shop_drag_active:
				_shop_apply_scroll_delta(mm.relative.y)
				get_viewport().set_input_as_handled()
```

```
            return
        # Тач (телефон)
        if event is InputEventScreenTouch:
            var st := event as InputEventScreenTouch
            if st.index != 0:
                return
            if st.pressed:
                _shop_drag_pos = st.position
                _shop_drag_active = false
            else:
                if _shop_drag_active:
                    get_viewport().set_input_as_handled()
                _shop_drag_active = false
            return
        if event is InputEventScreenDrag:
            var sd := event as InputEventScreenDrag
            if sd.index != 0:
                return
            if not _shop_drag_active and _shop_drag_pos.distance_to(sd.position) >
SHOP_DRAG_THRESHOLD:
                _shop_drag_active = true
            if _shop_drag_active:
                _shop_apply_scroll_delta(sd.relative.y)
                get_viewport().set_input_as_handled()

func _shop_get_event_position(event: InputEvent) -> Vector2:
        if event is InputEventMouse:
            return (event as InputEventMouse).global_position
        if event is InputEventScreenTouch:
            return (event as InputEventScreenTouch).position
```

```
        if event is InputEventScreenDrag:
                return (event as InputEventScreenDrag).position
        return Vector2.ZERO

func _shop_apply_scroll_delta(delta_y: float) -> void:
        var vbar := shop_scroll.get_v_scroll_bar()
        var max_scroll := vbar.max_value if vbar else 0.0
        shop_scroll.scroll_vertical = clampi(int(shop_scroll.scroll_vertical + delta_y),
0, int(max_scroll))

func _process(delta: float) -> void:
        candy_count += passive_income_per_second * delta
        _update_ui()
        _autosave_timer += delta
        if _autosave_timer >= AUTOSAVE_INTERVAL:
                _autosave_timer = 0.0
                _save_game()

func _start_candy_idle_animation() -> void:
        # Гойдання вліво-вправо в спокої
        candy_button.rotation = IDLE_SWAY_ANGLE
        _idle_tween = create_tween()
        _idle_tween.set_loops()
        _idle_tween.tween_property(candy_button,          "rotation",          -
IDLE_SWAY_ANGLE,
IDLE_SWAY_DURATION).set_ease(Tween.EASE_IN_OUT).set_trans(Tween.TR
ANS_SINE)
        _idle_tween.tween_property(candy_button, "rotation", IDLE_SWAY_ANGLE,
IDLE_SWAY_DURATION).set_ease(Tween.EASE_IN_OUT).set_trans(Tween.TR
ANS_SINE)
```

```
func _on_candy_clicked() -> void:
	candy_count += candies_per_click
	_play_candy_click_animation()
	_emit_click_particles()

func _play_candy_click_animation() -> void:
	# Ефект натискання: трохи стискається, потім повертається
	var tween := create_tween()
	tween.tween_property(candy_button, "scale", Vector2(0.88, 0.88), 0.05).set_ease(Tween.EASE_OUT).set_trans(Tween.TRANS_BACK)
	tween.tween_property(candy_button, "scale", Vector2(1.0, 1.0), 0.14).set_ease(Tween.EASE_OUT).set_trans(Tween.TRANS_ELASTIC)

func _play_button_press_animation(btn: Control) -> void:
	var tween := create_tween()
	tween.tween_property(btn, "scale", Vector2(0.88, 0.88), 0.05).set_ease(Tween.EASE_OUT).set_trans(Tween.TRANS_BACK)
	tween.tween_property(btn, "scale", Vector2(1.0, 1.0), 0.14).set_ease(Tween.EASE_OUT).set_trans(Tween.TRANS_ELASTIC)


func _build_click_particles_pool() -> void:
	click_particles_pool.clear()
	click_particles_pool.append(click_particles_template)
	var parent := click_particles_template.get_parent()
	for i in range(CLICK_PARTICLES_POOL_SIZE - 1):
		var dup: CPUParticles2D = click_particles_template.duplicate() as CPUParticles2D
		dup.emitting = false
```

```
            parent.add_child(dup)
            click_particles_pool.append(dup)
```

Додаток В

```
func _emit_click_particles() -> void:
    # Кожен клік — окремий емітер з пулу, щоб багато кліків не переривали один одного
    var center := candy_button.global_position + candy_button.size * 0.5
    _click_particles_index = (_click_particles_index + 1) % click_particles_pool.size()
    var emitter: CPUParticles2D = click_particles_pool[_click_particles_index]
    emitter.position = center
    emitter.restart()

func _on_settings_button_pressed() -> void:
    _play_button_press_animation(settings_button)
    settings_overlay.visible = true

func _on_settings_backdrop_pressed() -> void:
    settings_overlay.visible = false

func _on_settings_close_pressed() -> void:
    _play_button_press_animation(settings_close_button)
    await get_tree().create_timer(0.1).timeout
    settings_overlay.visible = false

func _on_shop_button_pressed(candy_index: int) -> void:
    # candy_index: 1 = Candy1 (сила кліку), 2..6 = Candy2..Candy6 (купити/прокачати)
    if candy_index == 1:
        _buy_candy1_upgrade()
```

```
        else:
                _buy_or_upgrade_candy(candy_index)

func _buy_candy1_upgrade() -> void:
        var cost := _get_candy1_cost()
        if candy_count >= cost:
                candy_count -= cost
                candy1_level += 1
                candies_per_click = candy1_level

func _buy_or_upgrade_candy(candy_index: int) -> void:
        var idx := candy_index - 2  # 0..4 для Candy2..Candy6
        if idx < 0 or idx >= candy_levels.size():
                return
        var cost := _get_candy_cost(candy_index)
        if candy_count < cost:
                return
        candy_count -= cost
        candy_levels[idx] += 1
        _update_passive()

func _get_candy1_cost() -> int:
        return CANDY1_UPGRADE_BASE * candy1_level

func _get_candy_cost(candy_index: int) -> int:
        var idx := candy_index - 2
        if idx < 0 or idx >= candy_levels.size():
                return 0
        var level := candy_levels[idx]
        var base := 0
```

```
        match candy_index:
                2: base = CANDY2_BASE
                3: base = CANDY3_BASE
                4: base = CANDY4_BASE
                5: base = CANDY5_BASE
                6: base = CANDY6_BASE
                7: base = CANDY7_BASE
                8: base = CANDY8_BASE
                9: base = CANDY9_BASE
                10: base = CANDY10_BASE
                11: base = CANDY11_BASE
        if level == 0:
                return base
        return base * (level + 1)

func _update_passive() -> void:
        passive_income_per_second = 0.0
        for i in range(candy_levels.size()):
                var rate: float = PASSIVE_RATE[i] if i < PASSIVE_RATE.size() else
0.5
                passive_income_per_second += candy_levels[i] * rate

func _get_passive_rate(candy_index: int) -> float:
        var idx := candy_index - 2
        if idx < 0 or idx >= PASSIVE_RATE.size():
                return 0.0
        return PASSIVE_RATE[idx]

func _get_candy_name(candy_index: int) -> String:
        if candy_index == 1:
```

```
        return ""
    var idx := candy_index - 2
    if idx < 0 or idx >= CANDY_NAMES.size():
        return "Цукерка %d" % candy_index
    return CANDY_NAMES[idx]

func _get_candy_description(candy_index: int) -> String:
    # Перший рядок — назва, другий — ефект/рівень (формат тис/млн для великих чисел)
    if candy_index == 1:
        return "Сила кліку\n+%s за клік" % _format_candy_count(float(candies_per_click))
    var idx := candy_index - 2
    if idx < 0 or idx >= candy_levels.size():
        return ""
    var name_str := _get_candy_name(candy_index)
    var level := candy_levels[idx]
    var rate := _get_passive_rate(candy_index)
    if level == 0:
        return "%s\nне куплено" % name_str
    var add := level * rate
    return "%s\n+%s/с\nрів. %d" % [name_str, _format_candy_count(add), level]

func _get_shop_button_text(candy_index: int) -> String:
    if candy_index == 1:
        return "Прокачати (%s)" % _format_candy_count(float(_get_candy1_cost()))
    var idx := candy_index - 2
    if idx < 0 or idx >= candy_levels.size():
        return ""
```

```
	var level := candy_levels[idx]
	var cost := _get_candy_cost(candy_index)
	var cost_str := _format_candy_count(float(cost))
	if level == 0:
		return "Купити (%s)" % cost_str
	return "Прокачати (%s)" % cost_str

func _is_shop_row_visible(row_index: int) -> bool:
	# Candy1Row=0, Candy2Row=1 завжди видимі; Candy3Row=2 видима якщо куплено Candy2, тощо
	if row_index <= 1:
		return true
	var prev_candy_idx := row_index - 2  # candy_levels[0]=Candy2, тому рядок 2 (Candy3) відкривається при candy_levels[0]>=1
	if prev_candy_idx < 0 or prev_candy_idx >= candy_levels.size():
		return true
	return candy_levels[prev_candy_idx] >= 1

func _format_candy_count(value: float) -> String:
	# Короткий формат: одна цифра після крапки; від 1000 — "X.X тис"
	var n := int(value)
	if n < 1000:
		return str(n)
	if n < 1_000_000:
		# Від 1 тис: "10.0 тис", "612.0 тис" — одна цифра після крапки
		var thousands := n / 1000.0
		return "%.1f тис" % thousands
	if n < 1_000_000_000:
		var millions := n / 1_000_000.0
		return "%.1f млн" % millions
```

```
        if n < 1_000_000_000_000:
                var billions := n / 1_000_000_000.0
                return "%.1f млр" % billions
        var trillions := n / 1_000_000_000_000.0
        return "%.1f трл" % trillions

func _update_ui() -> void:
        label_candy.text = "Цукерки: " + _format_candy_count(candy_count)

        for i in range(shop_rows.size()):
                shop_rows[i].visible = _is_shop_row_visible(i)

        for i in range(shop_labels.size()):
                if i < shop_labels.size():
                        shop_labels[i].text = _get_candy_description(i + 1)
        for i in range(shop_buttons.size()):
                if i < shop_buttons.size():
                        var idx := i + 1
                        shop_buttons[i].text = _get_shop_button_text(idx)
                        shop_buttons[i].disabled = candy_count < _get_shop_button_cost(idx)

func _get_shop_button_cost(candy_index: int) -> int:
        if candy_index == 1:
                return _get_candy1_cost()
        return _get_candy_cost(candy_index)

func _save_game() -> void:
        var data: Dictionary = {
                "candy_count": candy_count,
```

```
            "candy1_level": candy1_level,
            "candy_levels": candy_levels.duplicate(),
            "saved_at": Time.get_unix_time_from_system()
        }
        var json_string := JSON.stringify(data)
        var file := FileAccess.open(SAVE_PATH, FileAccess.WRITE)
        if file == null:
            push_error("Candy Clicker: не вдалося зберегти гру: %s" %
error_string(FileAccess.get_open_error()))
            return
        file.store_string(json_string)
        file.close()

func _load_game() -> void:
        if not FileAccess.file_exists(SAVE_PATH):
            return
        var file := FileAccess.open(SAVE_PATH, FileAccess.READ)
        if file == null:
            push_error("Candy Clicker: не вдалося відкрити збереження: %s" %
error_string(FileAccess.get_open_error()))
            return
        var json := JSON.new()
        var parse_err := json.parse(file.get_as_text())
        file.close()
        if parse_err != OK:
            push_error("Candy Clicker: помилка читання збереження (невалідний
JSON)")
            return
        var data: Variant = json.get_data()
        if typeof(data) != TYPE_DICTIONARY:
```

```
        return
    candy_count = float(data.get("candy_count", 0.0))
    candy1_level = int(data.get("candy1_level", 1))
    candies_per_click = candy1_level
    var arr = data.get("candy_levels", [])
    for i in range(candy_levels.size()):
        if i < arr.size():
            candy_levels[i] = int(arr[i])
        else:
            candy_levels[i] = 0
    _update_passive()
    # Офлайн-прогрес: нарахувати пасивний дохід за час, поки гра була закрита (за системним часом)
    var saved_at: float = float(data.get("saved_at", 0.0))
    if saved_at > 0.0:
        var now := Time.get_unix_time_from_system()
        var elapsed: float = now - saved_at
        elapsed = clampf(elapsed, 0.0, MAX_OFFLINE_SECONDS)
        candy_count += passive_income_per_second * elapsed

func _notification(what: int) -> void:
    if what == NOTIFICATION_WM_CLOSE_REQUEST:
        _save_game()
    # На Android/iOS: зберігати при переході у фон (Home, перемикання додатку)
    if what == NOTIFICATION_APPLICATION_PAUSED:
        _save_game()
```