Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ МЕТОДУ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ РІВНІВ ВІДЕОГРИ-ПЛАТФОРМЕРА
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-1

Алмакаєва Анастасія Євгеніївна
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Творошенко І. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Алмакаєвій Анастасії Євгеніївні
(прізвище, ім'я, по батькові)

1. Тема роботи «Розроблення та дослідження методу автоматичної генерації рівнів відеогри-платформера»

затверджена наказом по університету від «23» жовтня 2020 року № 1428Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 23 листопада 2020 р.

3. Вихідні дані до роботи Математичні моделі та алгоритми автоматичної генерації, перелік використовуваних програмних засобів: Unity, Adobe Photoshop, Adobe Illustrator, теоретичні відомості про методи автоматичної генерації рівнів у відеоіграх.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз та порівняння платформ для розробки відеоігор.
2. Аналіз та порівняння методів для автоматичного створення рівнів у відеоіграх.
3. Розробка алгоритму автоматичної генерації рівнів у відеогрі.
4. Створення гри за розробленим алгоритмом.
5. Дослідження методу автоматичної генерації рівня.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Приклад генерування рівня за допомогою генератора випадкових чисел, приклад генерування рівнів за допомогою шаблонів, ілюстрації процесу створення додатку, ілюстрації роботи додатку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	23.10.2020	
2	Аналіз завдання, підбір літератури	23.10.20-24.10.20	
3	Аналіз літератури з досліджуваної проблеми	25.10.20-27.10.20	
4	Аналіз технічних засобів розробки відеоігор	28.10.20-29.10.20	
5	Розробка методу автоматичної генерації рівня	30.10.20-01.11.20	
6	Програмна реалізація	02.11.20-09.11.20	
7	Оформлення пояснювальної записки	10.11.20-15.11.20	
8	Перевірка на плагіат	15.11.20	
9	Рецензування	16.11.20	
10	Підготовка презентації та доповіді	16.11.20	
11	Занесення роботи в електронний архів	29.11.20	
12	Попередній захист атестаційної роботи	30.11.20	

Дата видачі завдання _23_ __листопада__2020 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Творошенко І. С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: 78 с., 40 рис., 40 джерел.

ВІДЕОГРА, АРКАДА, ПЛАТФОРМЕР, ФРЕЙМВОРК, МІДКОРНИЙ.

Метою дослідження є розробка методу, що дозволяє автоматично генерувати карту рівня у відеогрі, розміщувати на ній персонажів-ворогів, бонуси, пастки та елементи декору.

Об’єктом дослідження є автоматичне генерування рівнів у комп’ютерних відеоіграх-платформерах.

Використано методи числового моделювання та аналітичного обґрунтування. Проведено дослідження методів автоматичної генерації карти рівня та методів розміщення персонажів на цих рівнях. Досліджено методи автоматичної генерації .

У результаті роботи розроблено алгоритм генерування рівнів з обмежувальними умовами та здійснена програмна реалізація відеогри з можливістю безкінечного генерування рівнів без втручання розробника.

VIDEO GAME, ARCADE, PLATFORMER, FRAMEWORK, MEDCOR.

The purpose of the study is to develop a method that allows you to automatically generate a level map in the video game, place on it the characters of enemies, bonuses, traps and decor elements.

The object of research is the automatic generation of levels in computer video games-platformers.

Methods of numerical modeling and analytical substantiation are used. A study of the methods of automatic generation of the level map and methods of placing characters at these levels. Methods of automatic generation are investigated.

As a result of work the algorithm of generation of levels with restrictive conditions is developed and the software implementation of video game with a possibility of infinite generation of levels without development of the developer is carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПК – персональний комп’ютер

ШІ – штучний інтелект

ОС – операційна система

NPC – non-player character, неігровий пеонаж

UI – user interface, інтерфейс користувача

ВСТУП

Відеогра – це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотну інформацію з відеопристрою. Електронні пристрої, які використовуються для того щоб грати, називаються ігровими платформами. Наприклад, до таких платформ належать персональний комп'ютер та гральна консоль. Пристрій введення, який використовується для керування грою, називається ігровим контролером. Це може бути, наприклад, джойстик, клавіатура та мишка, геймпад або сенсорний екран [1].

Нерідко відеоігри в повсякденному житті та пресі називають комп'ютерними іграми. Проте розробники та дослідники таких ігор послуговуються терміном «відеогра», виходячи з їх специфіки та місця в ієрархії ігор загалом:

– гра – діяльність із розважальною і/або навчальною метою за встановленими правилами. Включає в себе такі різновиди як дитячі, спортивні, настільні, азартні, рольові та, окрім інших, електронні ігри;

– електронна гра – гра, що відбувається за допомогою електронних пристроїв, програмованих чи непрограмованих, які утворюють інтерактивну систему. Такими можуть бути пейнтбольні автомати, слотові машини, електромеханічні ігри, лотереї, аудіоігри і відеоігри. Оскільки відеоігри на програмованих пристроях складають основну їх частину, нерідко терміни «електронна гра», «комп'ютерна гра» і «відеогра» вживаються як тотожні;

– комп'ютерна гра – гра, яка забезпечується програмно керованим електронним пристроєм – комп'ютером. Це також може бути слотовий автомат, пейнтбольна машина, аудіо чи відеогра;

– відеогра – гра, яка відбувається через керування візуальними образами на моніторі чи іншому дисплеї. Може забезпечуватися як програмованим, так і не програмованим електронним пристроєм: аркадним

ігровим автоматом, ігровою приставкою, персональним комп'ютером. Зрідка під відеогрою розуміється гра тільки для ігрової приставки [2].

Комп'ютерні ігри можуть бути класифіковані за кількома ознаками:

– платформа. Гра може належати як до однієї платформі, так і бути мультиплатформовою;

– жанр. Гра може належати як до одного, так і до кількох жанрів, а в унікальних випадках – відкривати новий або бути поза всяких жанрів;

– кількість гравців і спосіб їх взаємодії. Гра може бути однокористувацькою – розрахованою на гру одної людини, або багатокористувацькою – розрахованої на одночасну гру кількох людей; а також вестися на одному комп'ютері, через інтернет, електронну пошту, або масово;

– візуальне уявлення. Гра може як використовувати графічні засоби оформлення, так і навпаки, бути текстової. Гра також може бути двомірної або тривимірної. Є й звукові гри – в них замість візуального представлення використовуються звуки [3].

Ця робота присвячена саме створенню аркадної гри – це така, у якій навмисно спрощений ігровий процес та короткий ігровий час, але підвищена інтенсивність, порівняно з іншими жанрами. До аркадних ігор відносяться платформери – це жанр відеоігор, ігровий процес в якому складається зі стрибків персонажа по різноманітних платформах та через перешкоди, збирання предметів, інколи необхідних для завершення рівня.

Розробка відеогри (англ. Video game development) – це процес створення відеогри, яким займається розробник відеоігор (англ. video game developer), котрий може бути як однією людиною так і компанією з сотнями співробітників. Гра може розроблятися як силами кількох людей з обмеженим бюджетом, так і спираючись на фінансування від видавця (англ. video game publisher) [4]. Тривалість та вартість розробки залежить від складності проекту. Зазвичай розробкою займається такий штат фахівців: геймдизайнери, менеджери, програмісти, художники, звукорежисери,

композитори, звукоінженери, сценаристи, актори озвучування, дизайнери рівнів, піар-менеджери та тестери. Але існують також незалежні ігри або інді-ігри (англ. indie games від independent – «незалежний») – відеоігри, створені незалежно від фінансової підтримки великих видавців. Часто ці ігри дешеві або безкоштовні, багато незалежних ігор мають невеликий розмір і тому поширюються через інтернет за допомогою цифрової дистрибуції.. Більшість спочатку вільних ігор також належить до цієї категорії. Флагманами інді-індустрії, конкуруючими з іграми ААА рівня, вважаються такі інді-ігри, як Braid, World of Goo, Super Meat Boy і Minecraft (з 2014 р. більше не є інді-грою, бо проект почала фінансувати компанія Microsoft). При цьому остання створена одним розробником, були продані десятки мільйонів копій цієї гри, і вона є найбільш продаваною інді-грою за версією книги рекордів Гіннесса. Через те, що інді-ігри створюються одним або невеликим колективом однодумців, тому потребують набагато більших зусиль від кожного учасника процесу створення відеогри [5].

Складність розробки саме гри-платформера полягає у тому, щоб створити велику кількість різноманітних зовні рівнів, що складаються із одного й того ж набору блоків. Це слугувало причиною створення даної роботи: спрощення розробки гри-платформера та економія ресурсів для цього (часу, обладнання, грошових вкладень, залучення сторонніх спеціалістів у ігровій галузі). Ще однією метою роботи є логічне розміщення на рівнях унікальних ігрових блоків з метою урізноманітнення ігрового процесу, підвищення рівня зацікавленості та залученості користувача в процес гри.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ПЛАТФОРМ ДЛЯ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ РІВНІВ ВІДЕОІГОР

1.1 Аналіз основних методів для автоматичної генерації рівнів відеоігор

Розробка відеогри це складний багатопрофільний процес, що потребує наявність багатьох спеціалістів у різних сферах та злагодженої роботи кожного члена команди. Вона має низку послідовних етапів, загалом їх є три: розробка програмного (джерельного) коду, розробка контенту (малюнки, моделі, музика) та розробка ігрових механік. Їм передує проектування (препродакшен) – генерування геймдизайнером ідей щодо майбутньої гри, вибір жанру, тематики, особливостей ігрового процесу, розробка сценарію, образів персонажів та оточуючого середовища. Менеджер координує дії різних людей, залучених до розробки, складає план їхньої роботи, встановлює терміни її виконання, планує витрати. Готова гра в свою чергу має пройти низку етапів, в ході яких потрапляє до гравців і підтримує інтерес до себе [6]. Індустрія відеоігор включає у себе багато людей з різними професіями та ролями:

– програмісти, які відповідають за технічні можливості гри;

– художники – це ті, хто створюють графіку для візуальних елементів відеоігор, таких як персонажі, транспортні засоби, реквізит, декорації, фон, об’єкти, кольори, текстури і одяг. Всі ці речі складають зовнішній вигляд гри. Хоча ця робота орієнтована на комп’ютерну графіку, часто ігрові художники як і раніше використовують традиційний ручний ескіз на початкових етапах роботи. Ігрові художники поділяються на підрозділи:

1) для великих проектів наймають концепт художників. Вони як правило, використовуючи ручку і папір, а деякі – комп’ютерне програмне забезпечення, створюють ескізи ідей для ігрових світів, персонажів, об’єктів, транспортних засобів, меблів, одягу і т.д. Хоч і не беруть участь у створенні

фактичної ігрової графіки, їх концепція допомагає сформувати зовнішній вигляд майбутньої гри;

2) моделювальники – вони моделюють персонажів, об’єкти і навколишній світ гри, в тому числі форми життя, декорації, рослинність, меблі, транспортні засоби і т. д. вони балансують візуальні деталі з обмеженнями технології гри;

3) художники 2D текстур, які створюють і застосовують текстури до персонажів, оточення і ігровим предметів, таким як поверхні стін і будівель. Це високопрофесійна область, що вимагає чималих знань в області освітлення, перспективи, матеріалів і візуальних ефектів;

4) художники оточення – вони працюють з навколишнім середовищем (ігровим світом). Вони виконують 3D-моделювання, текстурування, створюють складні багатошарові шейдери і деякі прості анімації. Вони беруть побудовані і текстуровані 3D-об’єкти і будують навколишній світ – екстер’єри, інтер’єри, дороги, мости, гігантські ландшафти, скелясті схили, глибокі ліси тощо;

5) художники освітлення – дбають про висвітлення, і є еквівалентом режисера фотографії в світі кіно. Вони створюють і поміщають освітлення на всі рівні гри, регулюють колір, інтенсивність і згасання, щоб зробити навколишній світ більш реалістичним і створити настрій;

6) художники ефектів (FX Artist) – працюють з комбінацією 2D і 3D інструментів, систем частинок і вогнів, оживляючи будь-яку область. Художники ефектів покликані створювати спалаху від пострілів, погодні ефекти, іскристі проводи, поточну воду, дим, пил, пар і багато іншого;

– аніматори – це художники-артисти, які за допомогою певних прийомів і технологій створюють ілюзію руху персонажа мультфільму або комп’ютерної гри. Персонажі при цьому рухаються, розмовляють, відчувають і виявляють різні почуття. Все це аніматор повинен зобразити, користуючись різними професійними прийомами. Чим більше живим і яскравим вийдуть персонаж – тим вище якість роботи аніматора;

– композиторів та звукорежисерів, які створюють звукове оформлення та музичний супровід, який нерідко видається окремим накладом. Хоча «композитор комп'ютерних ігор» з першого погляду звучить якось несерйозно, це одна з найважливіших та складних професій. Адже, на відміну від кінокомпозитора, мета ігрового – передати атмосферу, настрій, та емоції неживих персонажів в неіснуючому всесвіті;

– за успішне завершення роботи над проектом відповідають продюсери. Це поняття досить розмивчате та дуже сильно відрізняється в залежності від сфери діяльності продюсера. Якщо коротко, ігровий продюсер стежить, щоб все, що повинно було бути зроблено, виявилося зроблено;

– після завершення роботи над технічною частиною піар-менеджери проводять рекламну компанію для поширення готового продукту. Цей підрозіл також відповідає за формування іміджу компанії-розробника. Наприклад, компанія CD Projekt RED, що створила культову гру «Відьмак», користується довірою у споживачів. Вони можуть собі дозволити пропонувати оформлення предзамовлення продуктів, які ще находяться на початкових етапах розробки. Відеоігра «Cyberpunk 2077» надійде в продаж 19-го листопада 2020 року, а купити її за предзамовленням можна було ще 16-го квітня. При цьому покупець отримає як бонус комікс за темою гри, оригінальний саундтрек, буклет з добірку ігрових ілюстрацій, шпалери для робочого столу та книгу правил для настільної гри. Усе це допомагає менше звертатися до інвесторів та забезпечувати стабільну роботу компанії [7].

1.2 Етапи розробки відеоігор

Можна виділити чотири основні етапи розробки відеогри: написання коду, розробка контенту, розробка ігрової механіки, тестування.

1.2.1 Написання коду

Оскільки відеогра є комп’ютерною програмою, її робота, технічні можливості, контент та ігровий процес, забезпечується програмним кодом. Розробка гри включає ті ж етапи, що і розробка програмного забезпечення, але передбачає більше роботи над контентом і створення ігрових механік.

Сучасні ігри здебільшого засновані на готових програмних модулях – ігрових рушіях, де вже реалізовані базові функції, здатні зв’язувати воєдино графіку, звук, об’єкти і їх рухи. Щоб налаштувати рушій для реалізації конкретного задуму програмісти доопрацьовують його, додаючи потрібні функції. Існують як вільні ігрові рушії, доступні будь-кому, так і ті, що вимагають отримання ліцензії на їх використання. Крім того рушії різняться за ліцензіями. Для незалежних розробників їх використання може бути значно дешевшим.

Деякі рушії розраховані на створення ігор конкретного жанру, інші – універсальні. Не всі рушії можуть забезпечити однакові внутрішньоігрові можливості та рівень графіки. Частина рушіїв дозволяють створювати ігри для різних платформ, так Unreal Development Kit підтримує розробку інтерактивних творів для PC, Xbox 360, PlayStation 3, Wii та Android [8].

Деякі ігри створюються в спеціальних програмах, які вже мають початкові ресурси, дії, та не вимагають знання мов програмування. Прикладами таких програм є Game Maker, Construct, RPG Maker.

Ролі окремих дисциплін програмування включають:

– фізика – програмування ігрового механізму, включаючи імітацію фізики, зіткнення, переміщення предметів тощо;

– штучний інтелект (ШІ) – виробництво комп’ютерних агентів із використанням ігрових методів ШІ, таких як сценарії, планування, рішення на основі правил тощо;

– графіка – управління використанням графічного вмісту та міркувань пам’яті; виробництво графічного движка, інтеграція моделей, текстур для роботи на фізичному двигуні;

– звук – інтеграція музики, мови, звуків ефектів у відповідні місця та час;

– ігровий процес – реалізація різних ігрових правил та особливостей (іноді їх називають загальними);

– сценарії – розробка та обслуговування системи команд високого рівня для різних ігрових завдань, таких як ШІ, тригери редактора рівнів тощо;

– UI – створення елементів інтерфейсу користувача, таких як меню опцій, HUD, системи довідки та зворотного зв’язку тощо.

– обробка введення – кореляція обробки та сумісності різних пристроїв введення, таких як клавіатура, миша, геймпад тощо;

– мережеві комунікації – управління входами та виходами даних для локального та інтернет-ігрового процесу;

– ігрові інструменти – виробництво інструментів, що супроводжують розвиток гри, особливо для дизайнерів та сценаристів [9].

1.2.2 Розробка контенту

Відеогра передбачає створення графіки, звуків та внутрішньоігрових текстів. Концепт-арти виконуються на папері або комп’ютері, зазвичай в кількох варіантах. На основі концепт-артів художниками затверджуються і створюються двовимірні або тривимірні моделі персонажів, предмети та декорації. Для цього художники працюють в програмах, призначених для роботи із графікою.

Щоб моделі рухалися, вони анімуються в інших спеціальних програмах. Створюються різні набори рухів, які відтворюватимуться залежно

від конкретних дій гравця з допомогою програмного коду. У випадку двовимірної графіки це набори спрайтів, де кожна картинка є окремими кадром [10]. Для реалізації реалістичних рухів чи емоцій може застосовуватися захоплення руху живих акторів. Після фіксування руху датчиками вони переносяться на комп'ютерного персонажа.

Візуальні ефекти роблять гру видовищнішою і задають стиль. Серед них деякі додають реалістичності, як відкидання тіней, заломлення світла, постріли і вибухи. Інші позначають стани і дії персонажа, які визначають стиль виконання гри. Деякі ігри цілком виконуються в стилі коміксу або кіноплівки. За реалізацію картинки і звуку відповідають графічний і звуковий рушії.

Для звукового оформлення пишеться музика і відбувається озвучування персонажів [11]. Крім того для повноцінного звукового оформлення потрібні ефекти, як кроки, звуки пострілів. Вони можуть обиратися з вільних бібліотек чи записуватися окремо. Де-які композитори спеціалізуються на створенні музики до ігор. Музика може виконуватися професійними оркестрами, мати пісенний супровід. Діалоги персонажів часто озвучуються спеціальними акторами на студіях озвучування.

Часом ігри містять відеовставки, створені в програмах двовимірної чи тривимірної анімації. Вони потрібні для більшого занурення гравця в ігровий процес та розкриття особливостей ігрового всесвіту та персонажів. Інколи для відеовставок знімаються живі актори і будуються декорації. Є актори, які спеціалізуються саме на зйомках в таких відеовставках або озвучуванні персонажів [12]. Сюжет, діалоги, додаткові тексти пишуться сценаристами і відповідальними за це письменниками. Потім вони переходять у руки редакторів, що стежать за збереженням законів ігрового всесвіту та лексичною досконалістю текстів.

1.2.3 Розробка ігрової механіки

Ігрова механіка визначає насиченість ігрового процесу, правила, за якими грається відеогра. Основою механіки є ігрові об’єкти, такі як персонажі, об’єкти, з якими вони можуть маніпулювати, декорації. Частиною ігрової механіки є управління, якими чином гравець керує персонажем та ігровим світом. Наприклад, як задається напрям руху, як активізується взаємодія з віртуальними предметами. Крім того на етапі розробки механіки створюється користувацький інтерфейс, який інформує гравця і дозволяє взаємодіяти зі світом гри.

Ігри зазвичай поділяються на рівні (локації) [13], щоб комп’ютер не навантажувався обчисленням всього світу гри. В новітніх іграх світ часто моделюється так, щоб не мати чітких поділів на локації. Дизайнер рівнів розміщує готові об’єкти в ігровому світі та продумує їх рухи. Компонування рівнів визначає наскільки цікавою буде гра, які можливості будуть у гравця вирішити конкретну ситуацію.

За взаємодію об’єктів, яка відбувається без контрою гравця, відповідає фізичний рушій. До прикладу, він реалізує закони інерції, гравітацію, поведінку рідин, властивості предметів. ШІ відповідає за поведінку персонажів, як вони реагуватимуть на дії гравця [14]. Багато подій в грі відбуваються за скриптами [15]. Самі події придумуються сценаристами, а скрипти реалізуються програмістами.

1.2.4 Тестування

Тестування ігор – це процес тестування програмного забезпечення для контролю якості відеоігор. Основною функцією тестування гри є виявлення та документування дефектів програмного забезпечення (відомих як помилки) [16]. Інтерактивне тестування розважального програмного

забезпечення – це високотехнологічна галузь, що вимагає обчислювальних знань, аналітичної компетентності, навичок критичної оцінки та витривалості. В останні роки сфера ігрового тестування потрапила в зону ризику через надзвичайну напругу та неприбутковість як у фінансовому, так і в емоційному плані.

Забезпечення якості є найважливішим компонентом у розробці ігор [17], хоча індустрія відеоігор не має стандартної методології. Натомість розробники та видавці мають власні методи. Маленькі розробники, як правило, не мають персоналу з контролю якості; однак великі компанії можуть наймати команди з контролю якості на повний робочий день. Гучні комерційні ігри професійно та ефективно перевірені відділом контролю якості видавця [18].

Тестування починається відразу після написання першого коду і збільшується в міру просування гри до її завершення [19]. Основна команда контролю якості буде стежити за грою з моменту її першого подання до контролю якості до пізнішої публікації.

1.3 Аналіз основних платформ для автоматичної генерації рівнів відеоігор

Більшість інструментальних засобів для розробки ігор можна поділити на 3 группи [20]:

– фреймворк – програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об’єднання різних компонентів великого програмного проекту. Фреймворк відрізняється від поняття бібліотеки тим, що бібліотека може бути використана в програмному продукті просто як набір підпрограм близькою функціональності, не впливаючи на архітектуру програмного продукту і не накладаючи на неї ніяких обмежень. У той час як фреймворк диктує правила

побудови архітектури додатку, задаючи на початковому етапі розробки поведінку за умовчанням, каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог;

– ігровий рушій – центральний програмний компонент комп'ютерних та відеоігор або інших інтерактивних додатків з графікою, оброблюваної в реальному часі. Він забезпечує основні технології, спрощує розробку і часто дає грі можливість запускатися на декількох платформах, таких як ігрові консолі та настільні операційні системи, наприклад, GNU / Linux, Mac OS X і Microsoft Windows. Основну функціональність зазвичай забезпечує ігровий рушій, що включає рушій рендерингу, фізичний рушій, звук, систему скриптів, анімацію, ШІ, мережевий код, управління пам'яттю та багатопоточність. Часто на процесі розробки можна заощадити за рахунок повторного використання одного ігрового рушію для створення безлічі різних ігор;

– конструктор ігор – програма, яка об'єднує в собі ігровий рушій і інтегроване середовище розробки, і, як правило, включає в себе редактор рівнів, що працює за принципом WYSIWYG. Такі програми значно спрощують процес розробки ігор, роблячи його доступним починаючим розробникам, і можуть бути використані в початковому вивченні програмування.

1.3.1 XNA Framework

Microsoft XNA (англ. XNA's Not Acronymed) – набір інструментів з керованим середовищем часу виконання (.NET), створений Microsoft для полегшення розробки комп'ютерних ігор. Мета XNA в спробі звільнити розробку ігор від написання «повторюваного шаблонного коду» і об'єднати різні аспекти розробки ігор в одній системі. Набір інструментів XNA був анонсований 24 березня 2004 р. на Game Developers Conference в Сан-Хосе,

Каліфорнія. Перший Community Technology Preview XNA Build був випущений 14 березня 2006 р..

Пакет Microsoft XNA, за словами представників Microsoft, дозволить розробникам ігор уникнути багатьох технічних труднощів, що виникають при написанні коду, а також забезпечить істотне зниження вартості кінцевої продукції. Крім того, завдяки XNA програмісти зможуть створювати принципово нові ігри з високоякісною графікою.

XNA Framework ґрунтується на реалізації . NET Compact Framework 2.0 для розробки для Xbox 360 і . NET Framework 2.0 на Windows. Він включає великий набір бібліотек класів, специфічних для розробки ігор, що підтримує максимальне повторне використання коду на всіх цільових платформах. Фреймворк виконується на модифікації Common Language Runtime, що оптимізована для ігор.

XNA Framework приховує низькорівневі технологічні деталі, пов’язані з розробкою гри. Таким чином, фреймворк піклується про різницю між платформами, дозволяючи розробникам приділяти більше уваги смисловому вмісту гри. XNA Framework інтегрується з декількома інструментами, такими як XACT, для допомоги в створенні контенту. XNA Framework надає підтримку створення та двомірних, і тривимірних ігор і дозволяє використовувати можливості контролерів Xbox 360. Десктопні програми можуть поширюватися безкоштовно під поточним ліцензуванням Microsoft. Існує проект MonoGame, що представляє собою кроссплатформенную open-source реалізацію XNA з додатковими можливостями [21].

1.3.2 Construct 2

Construct 2 – це заснований на HTML5 конструктор 2D ігор для ПК, IOS та Android, розроблений компанією Scirra. Конструктор спрямовано в першу чергу на людей, які не розуміються в програмуванні, дозволяючи

швидко створювати ігри способом Drag-and-drop з використанням візуального редактора та логічної системи, заснованої на принципі поведінки та реакції. Construct 2 є прямим нащадком попередньої версії програми, Construct Classic [22].

Інтерфейс програми має візуальний редактор, можливо створити гру без навичок програмування. У редакторі є «події» (англ. Events) і «дії» (англ. Actions), що створюють логіку ігор.

Одним з основних плюсів даного конструктора є кроссплатформеність. З ним можна створювати гри у вигляді простого HTML5 сайту і мобільного застосування (підтримуються iOS, Android, Windows Phone, Tizen, Blackberry та інші). Також можна розміщувати власний додаток в Chrome Web Store. З нещодавна є створення ігор під Wii U.

Також Construct 2 зручний в тестуванні. При покупці є можливість тестувати проекти прямо по WiFi. Все, що для цього потрібно, це два пристрої в одній мережі. Запускається гра на комп'ютері, вбивається на iPad ip адреса комп'ютера і все – можна грати.

Розробники Construct 2 забезпечують та навіть заохочують створення плагінів від сторонніх розробників. Так, на офіційному сайті можна знайти поради та уроки з написання та налаштування плагінів для коректної роботи. Всі плагіни Construct 2 написано на Javascript.

1.3.3 Adventure Game Studio (AGS)

Adventure Game Studio (AGS) – безкоштовний конструктор 2D-ігор, переважно квестів. Поєднує в собі візуальні інструменти, що дозволяють налаштувати різні параметри гри, а також редактор Сі-подібного скриптової мови, який служить для опису ігрової логіки [23].

Саме середовище розробки ігор працює на операційній системі Windows, починаючи з версії 2.70 – програма більше не компілює гри під

DOS (в DOSBox ігри запускаються без звуку). Також існують спеціальні порти для запуску AGS ігор на Mac OS (проект розпочато і покинутий в 2005 році) і Linux, але запустити на них гри вкрай складно. В кінці 2011 року з'явилися порти для запуску AGS ігор на PSP і Android.

Програма має підтримку досить великої кількості мультимедійних форматів: S3M, MOD, XM, MIDI, WAV, OGG, MP3, AVI – версія 2.61. У версії 2.72 є підтримка формату IT.

Є можливість запуску в повноекранному і віконному режимі, є підтримка різних ефектів зі звуком, скролінг кімнат. У комплект з конструктором входить повноцінна документація англійською мовою, в якій є докладне керівництво і приклад створення гри.

У AGS можна створювати ігри від 256 кольорів із роздільною здатністю 320×200 (в стилі старих класичних квестів) до 32-бітних з дозволом понад 800×600.

Також для 16-бітних і 32-бітових кольорів можна використовувати альфа-канал. Існують наступні роздільні здатності: 320×200, 320×240, 640×400, 640×480 і 800×600.

Крім того, AGS має можливості, яких немає у деяких інших движків – це запуск ігор без програми, можливість використання модулів, шаблонів і плагінів, а також можливість переведення гри на інші мови.

1.3.4 Game Editor

Game Editor – конструктор двовимірних ігор. Він підтримує розробку для багатьох платформ, включно з iPhone, iPad, Mac OS X, Windows (95-Windows 7), Linux, Windows-смартфони, GP2X, Pocket PC і Handheld PC. Сумісність зі всіма цими платформами було відзначено Game Discovery, популярним сайтом для розробників ігор, що виділяє його серед інших

подібних програмних засобів, таких як 3D Gamemaker, Darkbasic і Game Maker.

Game Editor створив Makslane Rodrigues, який розробляє його з 2002. Поточна версія – 1.4.0. Поширюється під подвійною ліцензією: GPL і комерційною. Умови GPL стосуються зокрема й кінцевих файлів скомпільованої в Game Editor гри, тому, якщо ви не бажаєте поширювати написану на ньому гру з відкритими початковими кодами, слід придбати комерційну ліцензію [24].

1.3.5 Godot Engine

Godot Engine – відкритий багатоплатформовий 2D та 3D гральний рушій, що розробляється співавторством Godot Engine Community. До публічного релізу у вигляді відкритого ПЗ рушій використовувався всередині деяких компаній Латинської Америки. Оточення розробника працює на Windows, Linux, OS X, BSD і Haiku та може експортувати ігрові проекти на ПК, консолі, мобільні та веб платформи [25].

Що стосується мов, то якщо не розглядати низькорівневий спосіб, найбільш популярні варіанти – це скриптові мови GDScript і C#, а також візуальний скриптинг для чогось простого. GDScript краще інтегрований в движок, має більше прикладів, не вимагає запуску зовнішнього середовища, а в плані загальної структури тут буде відбуватися все те ж, що у варіанті C# – оголошення змінних, виклик функції ініціалізації, цикл процесів, цикл фізичних процесів і так далі. Так що вибір GDScript, як основної скриптової мови, розробки має сенс. Починаючи працювати на C#, отримаємо хіба що більшу акуратність і подробиці запису, втрачаючи в лаконічності, але посилюючи розбірливість і контроль над ситуацією – фігурні скобочки замість використання табуляції, позначки кінця рядка, більш формальну типізацію.

1.3.6 Novashell Game Creation System

Novashell Game Creation System – це конструктор 2-мірних ігор професійного рівня, від Robinson Technologies, розповсюджуваний під відкритою ліцензією в стилі zlib / libpng. Побудований поверх бібліотеки-ігрового движка ClanLib. Дозволяє створення багатоплатформових ігор, що працюють під Windows, OS X і Linux [26].

Як фізичного движка використовує Box2D, як вбудованої мови програмування – Lua (хоча, як і в інших конструкторах ігор, дозволяє візуальне побудова ігор без програмування). У Novashell також включена готова підсистема штучного інтелекту. Як і в багатьох інших подібних системах, тут ви можете шукати шляхи за алгоритмом A *.

На відміну від таких програм, як Game Maker, Game Editor або Scirra Construct, тут немає поділу на IDE і режим гри. Завантаживши гру і натиснувши клавішу F1, ви потрапляєте в режим її редагування. Втім через цю особливість редагування lua-скриптів в самій Game Creation System неможливо – для цього викликається зовнішній редактор.

1.3.7 Stencyl

Stencyl – багатоплатформовий конструктор ігор. Середовище розробки Stencyl працює в інтеграції з онлайновим магазином компонентів ігр StencylForge і сайтом Stencyl.com, на якому розташовані навчальні матеріали, форуми користувачів конструктора і опубліковані ними гри. Дозволяє створювати гри для платформ iOS, Android, настільних комп’ютерів під управлінням Windows, Linux і Mac OS, а також ігор в форматі Adobe Flash і HTML 5.

Основні блоки, з яких потім складається гра, можна розділити на наступні типи [27]:

– актори: це може бути гравець або ворог, дерево або щось ще. У актора зазвичай є форма і картинка або серія картинок, яка створить анімацію;

– сцени: у гри може бути безліч сцен. Зазвичай у гри є основне меню або початкова сцена, а також сцени для різних рівнів гри і фінальна сцена;

– поводження: це готові можливості, які можна дати акторам або сценам. Також є можливість створити свою поведінку, щоб зробити вашу гру унікальною;

– події: це настроюються блоки команд, які можна створити і пов’язати з вашим героєм. Ви можете створити події, використовуючи спеціальний редактор Stencyl’s Event Editor.

1.3.8 Game Maker

Game Maker – один з найвідоміших конструкторів ігор. Написаний на Delphi Доступний для ОС Windows. Будучи професором Утрехтського університету Марк Овермарс почав розробляти Game Maker як навчальний 29 посібник для своїх студентів [28]. Створення ігор в ньому не потребує попереднього знайомства з будь-якою з мов програмування.

Гра в Game maker будується як набір ігрових об’єктів. За їх зовнішній вигляд відповідають спрайт, а поведінка задається шляхом опису реакцій на події. Для цього можна використовувати графічне представлення програм (близьке до блок-схемами) у вигляді послідовності іконок-дій. Програмування за допомогою дій відбувається в режимі drag-n-drop. Наприклад, для того щоб почати умовний оператор, потрібно перетягнути на панель дій восьмикутник з іконкою, що означає тип перевірки, а потім, можливо, ввести будь-які значення в форму, що з’явилася. Для більш просунутих користувачів є скриптова мова GML схожий на JavaScript і C++,

є можливість створення власних бібліотек дій, використовуючи Library Maker.

Розрахований в основному на створення двовимірних (2D) ігор будь-яких жанрів. Також підійде для створення різних презентацій і т.п. Game Maker розповсюджується на умовах Shareware, безкоштовна версія обмежена в функціональності, а при запуску ігор на стрічці завантаження гри показується логотип Game Maker.

Остання версія 8.1. Більше Game Maker не підтримується, його місце зайняло багатоплатформений розвиток проекту – Game Maker: Studio.

1.3.9 J.U.R.P.E.

J.U.R.P.E. розшифровується як Java Universal Role Playing Engine – це вільний (відкритий вихідний код поширюється) движок для розробки ігор жанру RPG [29]. Ігри на базі Jurpe засновані на системі прокачування персонажа, блуканням по підземеллях в напівсхематичному вигляді. Зовнішній вигляд ігор залишає бажати кращого, тому що багато елементів даних ігор реалізовані схематично.

Створення гри грунтується на деяких налаштуваннях. Ви починаєте з створення персонажа, вибираючи його показники, навички та ін. В іграх гравцеві надається можливість переміщатися по шестикутника, брати завдання, купувати зброю, боротися і мн. ін. Продумана до дрібниць система бою, досвіду і т.д. Якщо вас цікавить такий жанр ігор, то це середовище розробки буде надзвичайно доречним.

Сам движок написаний мовою Java, що робить його мультиплатформенним. Ви можете працювати з движком в Windows, Linux і на будь-який інший операційній системі, яка підтримує Java-додатки.

Конструктор поставляється з повноцінною демо-грою, в якій показані всі можливості.

Движок Jurpe і Jurpedemo поширюються згідно з ліцензією GNU/GPL – вільне поширення.

Движок все ще розвивається, тому ви можете прийняти в цьому участь, або спостерігати за оновленнями і нововведеннями на офіційній сторінці. Там же є інструкція по створенню гри (англійською мовою).

1.4 Постановка задачі дослідження

Актуальність проблеми розробки автоматичної генерації рівнів гри-платформера для ПК викликана складністю процесу розробки відеоігор, що потребує багато часу. Це слугувало причиною створення даної роботи: спрощення розробки гри-платформера та економія ресурсів для цього.

Об’єктом дослідження є автоматичне генерування рівнів у комп’ютерних відеоіграх-платформерах.

Метою дослідження є розробка методу, що дозволяє автоматично генерувати карту рівня у відеогрі, розміщувати на ній персонажів-ворогів, бонуси, пастки та елементи декору.

Для оптимізації роботи над автоматичним створенням відеогри необхідно вирішити такі завдання:

– розробити моделі гравця та ігрових блоків;

– інтегрувати моделі у середовище розробки Unity, щоб вони набули фізичних властивостей;

– розробити метод розміщення блоків, з яких складається рівень та алгоритм на основі цього методу;

– розробити алгоритм розміщення унікальних елементів рівня;

– розробити алгоритми закінчення гри та програшу;

– реалізувати розроблені алгоритми;

– створити логіку гри на початковому та кінцевому блоці, логіку програшу гравця, а також меню.

2 РОЗРОБЛЕННЯ МЕТОДУ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ РІВНІВ ДЛЯ ВІДЕОГРИ-ПЛАТФОРМЕРА

2.1 Визначення вимог та проєктування функціональної моделі для автоматичної генерації рівнів відеогри-платформера

Популярність онлайн-ігор і кіберспорту стрімко зростає, і мережеві підключення грають тут найважливішу роль [30]. Джо Камелло (Joe Cumello) з Ciena розповідає про те, що мережеві оператори і онлайн-гіганти звернули увагу на бум в сфері комп’ютерних ігор. Вони створюють платформи і мережеві пропозиції, що покращують ігровий процес і розповсюджують гри повсюдно.

Близько 400 мільйонів людей проводять час за переглядом і трансляціями ігрових змагань [31]. Популярність кіберспорту стрімко зростає. Раніше батьки прагнули обмежити час, який діти проводять за комп'ютером. Сьогодні ж багато хто з них хочуть побачити нікнейм своєї дитини в таблиці кращих гравців Fortnite.

Змінилися не тільки батьки і геймери. Сьогодні на цей тренд працює вже ціла індустрія, а щорічний обсяг глобального ігрового ринку становить 152. млрд доларів. Для порівняння: глобальний дохід від кіновиробництва та кінопрокату в 2018 році склав 136 млрд. доларів. В ігровій індустрії сьогодні працюють найбільші і найпопулярніші бренди всього світу.

2.1.1 Як зробити гру цікавою?

Розважальна функція ігор – абстрактна річ. І щоб створювати ігрові світи, в які користувачеві захочеться поринати з головою знову і знову, доведеться дуже багато продумати. Геймдев відноситься до індустрії розваг, а значить розробника ігор – розважати людей. Піднімати їм настрій і

надавати впевненості в собі. Створювати барвисті, захоплюючі ігрові світи, куди їм захочеться зануритися з головою. За виконання ігрових завдань гравці можуть отримати реальні нагороди (розвиток сюжету, красивий арт, класні кат-сцени).

Суть розробки програмного забезпечення полягає в створенні продукту, який служить визначеній меті. Не потрібно створювати дані для ПО крім тих випадків, коли мова йде про довідкових документах. Кінцевий користувач сам створює ці дані – відгуки про книжки, відомості, інвойси, облікові записи, цифровий арт і т.д.

Це все до того, що розважальна функція ігор – річ дуже суб'єктивна. І щоб виконати це неоднозначне людське бажання, доводиться розробляти ПО як інструментарій для подальшого виробництва ігрового контенту.

Для цього потрібні [32]:

– інструментарій (ПО) для створення інтерактивного геймплея;

– класи гравців, вороги, сюжет, з якими гравцеві буде цікаво взаємодіяти;

– потрібно переконатися, що дані і відповідне ПО оптимізовані для зручності гравців.

Вимоги до функціонала постійно змінюються. Варто приступити до роботи над чимось одним, скажімо, функціоналом бою, як незабаром виявиться, що для нього потрібно створити систему предметів, динамічне оточення, яке до того ж може впливати на здатності певного класу персонажа.

Інновації можуть бути «ітеративними», коли створення чогось нового відбувається на базі вже відомих формул. У розробці ігор в тій чи іншій мірі використовуються обидва підходи. І хоча розробники можуть бути прив'язані до певних формул (так, гоночна гра має на увазі своєрідні треки, бонуси і т.д.), в кінцевому рахунку, не вийде розробити такий самий продукт, як у інших представників індустрії.

Може виявитися, що гра зовсім неоригінальна, або механіки настільки різні, що їх не варто використовувати разом, або вона цікава, але в ній важко розібратися. Певний рівень досвіду, звичайно, допомагає згладити кути – старожилам індустрії буде легше обійти підводні камені, ніж новачкам. Але в будь-якому творчому процесі трапляються несподіванки. В цьому і полягає ідея свіжого, новаторського підходу.

2.1.2 Еволюція вимог до функціонала

Розробка починається з розстановки пріоритетів на основі того, наскільки важливою для гри є окрема функція. Потім створюються прототипи того, що має бути у грі, використовуючи вибіркові дані. У цьому розділі розглянемо це на прикладі компоненти системи бою у грі Monster Roller.

На самому початку планувалося реалізувати кілька ідей:

– геймплей багато в чому мав нагадувати RPG, але також включав би в себе елементи балансу карткових ігор;

– потрібно було додати барабан слот-машини, який би видавав бонуси на базі джек-потів (тоді ще 4 в ряд);

– розглядалися коефіцієнти ставок;

– «утримання» – функція, яка не дає слоту обертатися;

– можливість перемикатися між монстрами під час ходу;

– застосування предметів на монстрів.

І це далеко не повний список. Вимоги до функціонала можуть включати в себе, крім іншого, докладний опис того, що в них увійде (власне, дані), як вони будуть відображатися (інтерфейс), а також показники ефективності, що демонструють, чи працює функціонал так, як задумано.

Для прикладу розглянемо систему предметів:

1. Якими будуть особливості предмета? Чи буде він використовуватися на команду гравця, команду противника, або на обидві? Чи буде у нього термін дії? На які змінні можуть впливати предмети? Рівень здоров'я? Сила атаки? Показник регенерації? У відсотках або цілих числах?

2. Чи можна використовувати предмети після бою або вони витрачаються в бою?

3. Чи є вартість використання предмета? Як реалізується ця вартість? Чи буде це число ходів в бою (вартість часу) або щось інше? Чи потрібно буде купувати предмети в магазині? Якщо так, то за тверду або м'яку валюту?

І так далі. Документ, що містить відповіді на ці питання, називається списком вимог до функціонала.

Щоб не заплутатися під час розробки, не витрачати час на функції, що потім не будуть присутні у фінальній версії гри, щоб не пропустити важливу ігрову механіку, потрібно створити схему роботи над додатком.

GDD – рання версія документу, в якому вже на початковому етапі прописується концепція гри: замальовки всесвіту, основна ігрова механіка та саб-механіки гри, функції для залучення, монетизації гравців, ігрова економіка і баланс [33]. На цьому етапі GDD не містить повністю розписаних завдань, по реалізації механік, підготовки інтерфейсу або створенні діалогів для персонажів. Однак в ньому чітко прописана модель «денного бонусу», наприклад. Чи буде це накопичувальний бонус, чи буде він анулюватися, на скільки цінні подарунки в ньому будуть. Продюсер або геймдизайнер повинен чітко уявляти кожен компонент гри, його механіку і призначення.

Кожен розробник має стежити за зручністю та актуальністю ігрового контенту, який прийшов до нього в руки, потім – який він сам створив та додав у актуальну версію. Увесь процес розробки розбивається на циклічні етапи. З кожним етапом гра вдосконалюється за рахунок комунікації кожного спеціаліста та тестувальника. Узагальнена схема роботи над грою схематично зображена на рисунку 2.1.

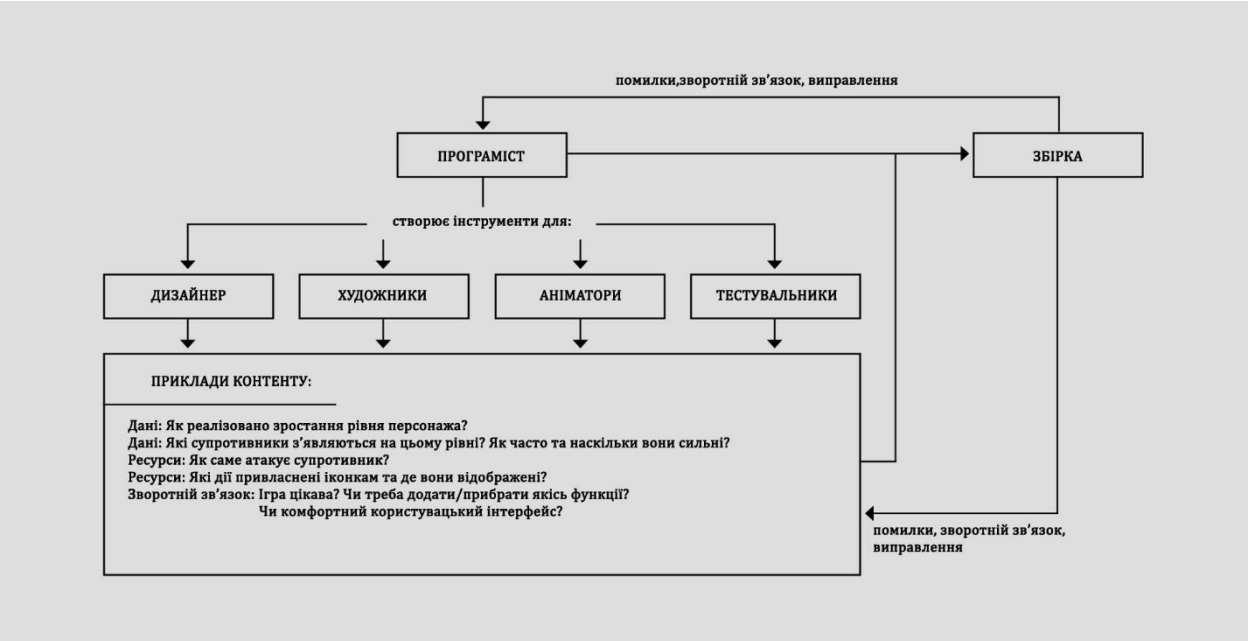


Рисунок 2.1 – Узагальнена схема роботи над грою

2.2 Особливості генерації рівнів відеогри-платформера за допомогою генератора випадкових чисел

Генерація ігрового світу важлива тому, що левел дизайн це витратний процес під час створення ігор. Кожен рівень потрібно опрацьовувати окремо, тестувати і балансувати.

А зробивши ігровий світ, який буде випадковим чином генеруватися, всі подібні проблеми зникають. Якісна автогенерація світу – це дуже корисний інструмент як для інді-ігор, так і для великих проектів.

2.2.1 Перший спосіб

Формування процесуального рівня у платформерській грі Spelunky (рис. 2.2) часто вважається прикладом високого рівня. Розглянемо два приклади генерування рівнів з вихідного коду оригінальної безкоштовної

гри, які ілюструють два шляхи найпростішого підходу до проблеми процедурного покоління.

Рисунок 2.2 – Приклад генерування рівня у відеогрі Spelunky

Перший приклад – розміщення загальних скарбів на рівні. В Spelunky немає системи шаблонів підкімнат для скарбів або алгоритму, який розумно створює відповідні місця для зберігання скарбів на рівні. Він вирішує проблему за допомогою наступного алгоритму: Для будь-якої порожньої поверхні землі на карті ймовірність існування скарбу в цьому просторі прямо пропорційна кількості твердих поверхонь, що прилягають до простору.

Оскільки Spelunky базується на сітці плиток, існує чотири можливі конфігурації:

– плитка із землею внизу та нічим не зверху;

– ліворуч або праворуч (відкритий простір);

– плитка із землею на дні та однією додатковою сусідньою поверхнею (зазвичай це кут кімнати, але іноді і простір);

– тупиковий закуток з доступом лише з одного кардинального напрямку, а також простір, повністю закритий твердими поверхнями.

Зі збільшенням кількості сусідніх поверхонь збільшується ймовірність появи скарбу. Це має інтуїтивний сенс: людина не поклала б скарб посеред відкритої підлоги, але може заховати його в кут кімнати або простору і точно його закопає.

Це надзвичайно елегантно та може здаватися дуже складним у реалізації. Але велика кількість коду Spelunky достатньо проста.

Наступний приклад автоматичної генерації – це розміщення ворога Гігантського павука. У першому наборі печерної плитки перевіряється наявність двох порожніх місць під кожною цегляною плиткою, яка не є стелею самого рівня або у початковій кімнаті, або в нижній половині кімнати. Якщо є два блоки порожніх просторів плитки, таким чином, дозволено генерувати Гігантського павука на цьому рівні, але це ще не зроблено, тоді є шанс 1 із 40 для генерації Гігантського павука і кількох павутинь прямо під цією цеглою. Якщо Гігантського павука вже є на рівні, не повинен генеруватися ще один.

Зазначений алгоритм працює приблизно на 30 рядків коду, порівняно з 1 рядком коду для розміщення скарбів. Хоча один алгоритм набагато коротший за інший, обидва вони прості. І те, і інше не орієнтується на клітинні автомати чи шум Перліна, і не схоже на класичні режими очікування.

Клітинний автомат – це дискретна модель, яка вивчалася в математиці, теорії обчислюваності, фізики, теоретичної біології і мікромеханіки. Включає регулярну решітку клітинок, кожна з яких може перебувати в одному з кінцевого безлічі станів, таких як 1 і 0. Решітка може бути будь-якої розмірності. Для кожної клітинки визначено безліч клітинок, званих околом. Наприклад, окіл може бути визначений як всі клітини на відстані не більше 2 від поточної (окіл фон Неймана рангу 2). Для роботи клітинного автомата потрібно завдання початкового стану всіх клітин і правил переходу чарунків з одного стану в інший. На кожній ітерації, використовуючи правила переходу і стану сусідніх клітин, визначається новий стан кожної. Зазвичай

правила переходу однакові для всіх клітин і застосовуються відразу до всієї решітки.

Perlin noise (Шум Перлина, також іноді Класичний шум Перлина) – математичний алгоритм з генерування процедурної текстури псевдо-випадковим методом [34]. Використовується в комп'ютерній графіці для збільшення реалізму або графічної складності поверхні геометричних об'єктів. Також може використовуватися для генерації ефектів диму, туману і т.д. Шум Перлина – це градієнтний шум, що складається з набору псевдовипадкових одиничних векторів (напрямів градієнта), розташованих в певних точках простору і інтерпольованих функцією згладжування між цими точками. Для генерації шуму Перлина в одновимірному просторі необхідно для кожної точки цього простору обчислити значення шумової функції, використовуючи напрямок градієнта (або нахил) в зазначеній точці.

Обидва алгоритми в Spelunky покладаються на прості перевірки функцій ігрового світу, які вже представлені в ігровій моделі. Жоден алгоритм не вносить додаткові дані або системи в гру.

2.2.2 Другий спосіб

Для цього способу потрібно реалізувати функцію генерації карти простою формулою, що теж включає функцію рандомізації. Змінюючи значення змінних у формулі, можна відкалібрувати карту до прийнятного стану [35].

Карта складається з 128×128 блоків, кожен блок розміром 32×32 пікселя. Вона зберігається у вигляді масиву 128×128, де:

– 0 – порожній блок;

– 1 – земля, персонаж може переміщатися по цьому блоку і може його підірвати;

– 2 – камінь – блок, який не можна підірвати;

– 3 – ящик з коштовностями;

– 4 – шипи, при падінні на які гравець вмирає.

Елементи рівня:

– поверхи: вся карта ділиться на умовні поверхи, відстань між поверхами визначається функцією «Rand»;

– прорізи: між поверхами генерується випадкова кількість проходів випадкової ширини;

– шум: на поверхах утворюються випадкові блоки;

– кімнати: на кожному поверсі формується випадкова кількість прямокутних кімнат з одним або двома виходами або без них (рис. 2.3);

– бігові доріжки: для того, щоб гравець міг розігнатися під час бігу, на кожному поверсі генерується випадкова кількість прямих доріжок для бігу, над деякими з них випадково малюється «стеля» випадкової висоти (рис. 2.4).

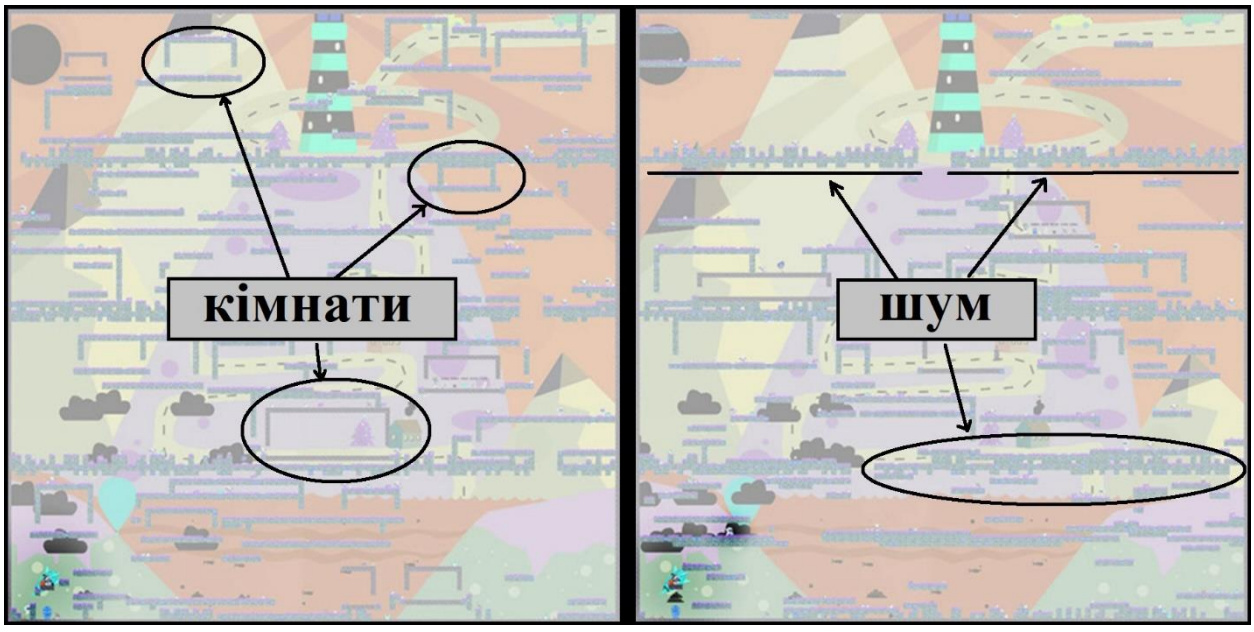


Рисунок 2.3 – Перший приклад генерування рівня за допомогою генератора випадкових чисел

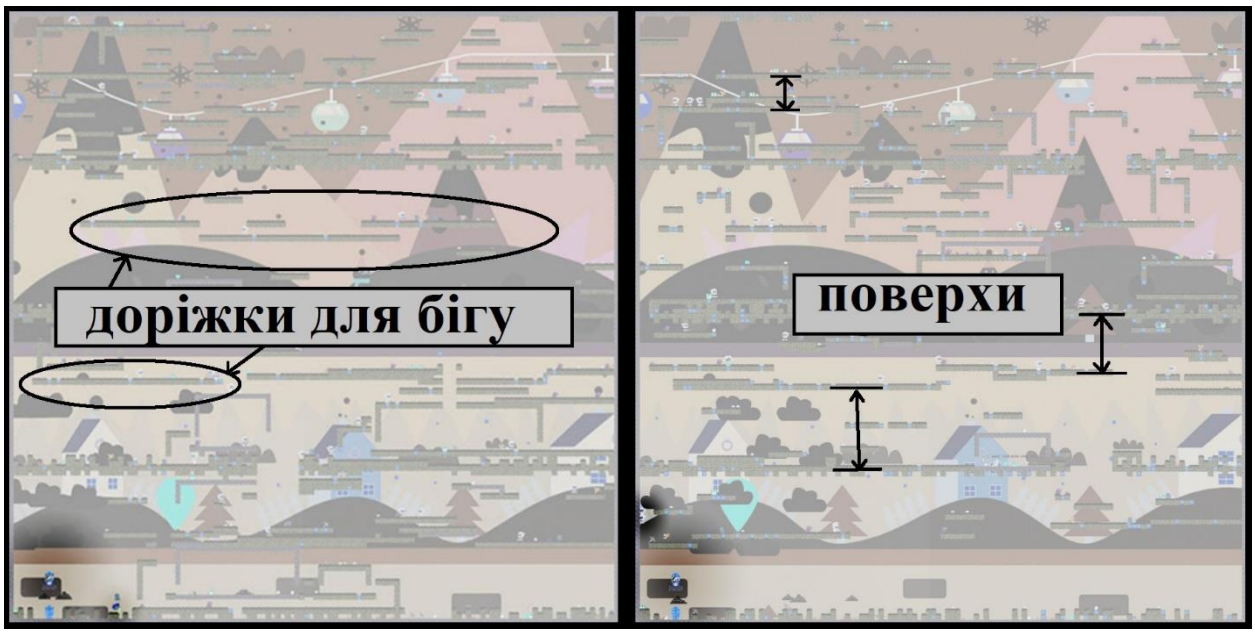


Рисунок 2.4 – Другий приклад генерування рівня за допомогою генератора випадкових чисел

Вимоги, яким повинен відповідати алгоритм:

– для даної гри гравець повинен дійти з точки А до точки В;

– під час просування рівнем гравець не повинен застрягти або наштовхнутися на глухий кут;

– необхідна наявність простору для польотів на реактивному ранці.

Мінуси генерації рівня методом «рандому»:

– рівень є занадто легким, часто отвори між поверхами знаходяться один над одним;

– рівні виглядають занадто одноманітно;

– карти візуально не виглядають цілісними.

Покращуючи складність формули для генерації карти, можна було б домогтися кращого результату, таким чином, зробивши рівні більш сприйнятними для очей користувача та більш схожими на реальні об’єкти з навколишнього середовища.

2.3 Особливості генерації рівнів відеогри-платформера за допомогою використання шляху

Сутність цього методу є у тому, щоб генерувати рівні за деяким шаблоном. Кожен шаблон повинен мати в собі підшаблони.

Шаблон – в мовах програмування, специфікація форми подання та правил редагування елемента даних за допомогою рядка символів, в якій кожен символ вказує на допустимий вид символу, що підлягає виконанню редагування для відповідної позиції значення елемента. Вперше шаблон був введений, як конструкція мови КОБОЛ.

У С ++ можливе створення шаблонів функцій і класів [36]. Шаблони дозволяють створювати параметризрвані класи і функції. Параметром може бути будь-який тип або значення одного з допустимих типів (ціле число, enum, покажчик на будь-який об'єкт з глобально доступним ім'ям, посилання). Хоча шаблони надають коротку форму запису ділянки коду, насправді їх використання не скорочує виконуваний код, так як для кожного набору параметрів компілятор створює окремий екземпляр функції або класу. Як наслідок, зникає можливість спільного використання скомпільованої коду в рамках поділюваних бібліотек.

Спочатку ігрове поле ділиться на 64 квадрати, тобто отримуємо поле 8×8. Для кожного з цих квадратів треба завантажити значення з шаблону. Надалі квадрати будемо називати «кімнатами». Отже, тепер карта складається з 64 кімнат.

Сутність гри полягає в тому, щоб пробратися з точки А (точка входу) в точку В (точка виходу), переміщуючись при цьому з однієї кімнати в іншу. У даному випадку, точка входу знаходиться у верхньому ряду кімнат, а точка виходу – внизу карти.

Далі потрібно прокласти маршрут (шлях), за яким гравець моє рухатися. Для цього кімнати розділяються на чотири типи:

– 0: кімната, яка не перебуває на шляху основного маршруту;

– 1: кімната, з якої гарантовано є вихід зліва і справа;

– 2: кімната, з якої гарантовано є вихід зліва, справа і внизу;

– 3: кімната, з якої гарантовано є вихід зліва, справа і вгорі.

Даний алгоритм генерації шляху нагадує пристрій машини Тьюринга. Керуючий пристрій (головка запису-читання) переміщується по матриці кімнат. Число можливих станів керуючого пристрою скінчено і точно задано, стани відповідають типам кімнат. Алгоритм має такі етапи:

– розмістити першу кімнату у верхньому ряду, тип кімнати може бути 1 або 2. Для прикладу обирається тип 1;

– далі треба вирішити, в якому напрямку буде рухатися гравець. Для цього береться навмання число від 1 до 5: число 1 та 2 – це поворот наліво, 2 та 3 – поворот направо, число 5 – це рух униз. При цьому тип найпершої кімнати змінюється з 1 на 2 (тепер у стартової кімнати є вихід внизу);

– якщо під час руху шлях приводить до межі екрану зліва чи справа, треба змінити напрям на протилежний. Варто зазначити, що кожен раз, коли гравець рухається вниз, потрібно перезаписати тип поточної кімнати з 1 на 2 (тому що у цієї кімнати додається вихід знизу;

– після переміщення в наступну кімнату треба перевірити, з якої кімнати тільки що перемістився гравець. Якщо тип кімнати був 2, треба його залишити (з черговим виходом вниз) або замінити на тип 3 (вихід зверху, зліва і справа). Кімнати типу 2 і 3 мають виходи зліва і справа, тож можна повторювати алгоритм знову;

– якщо гравець знаходиться в нижньому ряду та алгоритм пропонує рухатися ще нижче, треба розмістити вихід з кімнати і завершити алгоритм.

На даному етапі був сформований шлях руху для персонажа (рис. 2.5). Кімнати, що не були пройдені гравцем заповнюються нулями (0 – кімната, яка може не мати виходів взагалі).

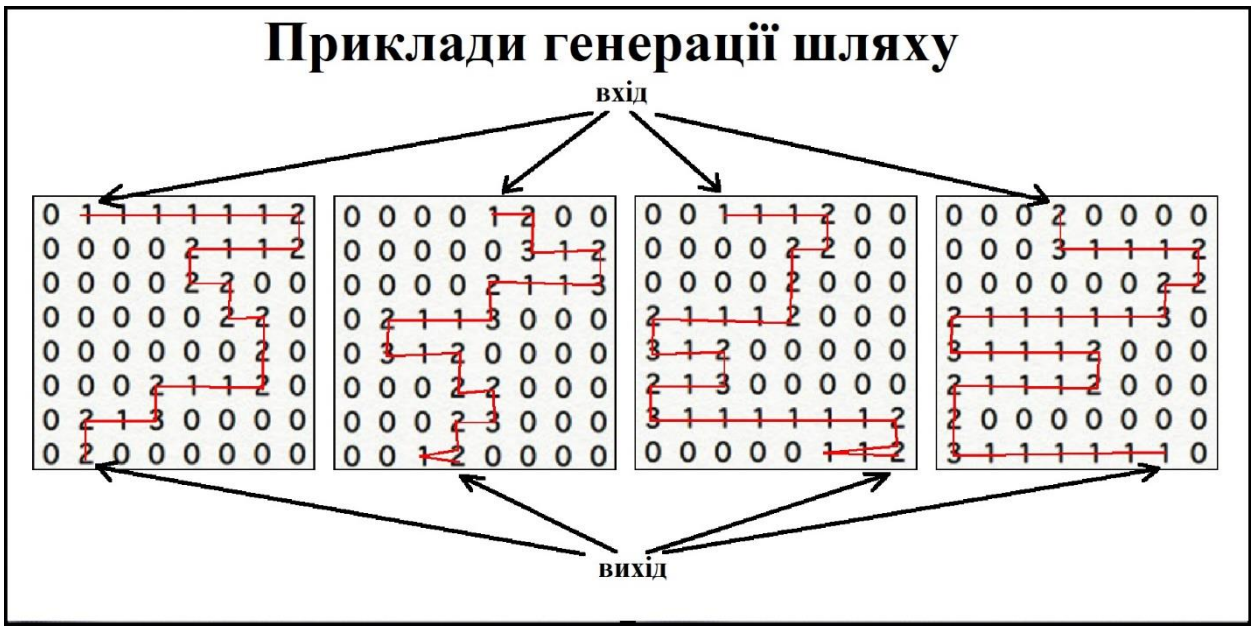
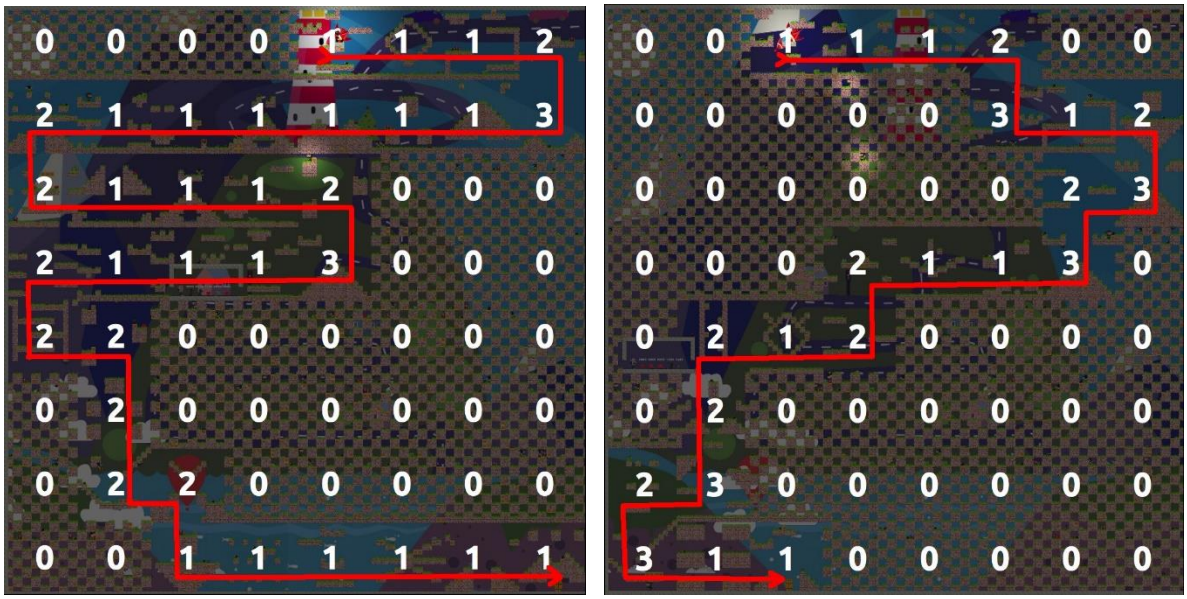


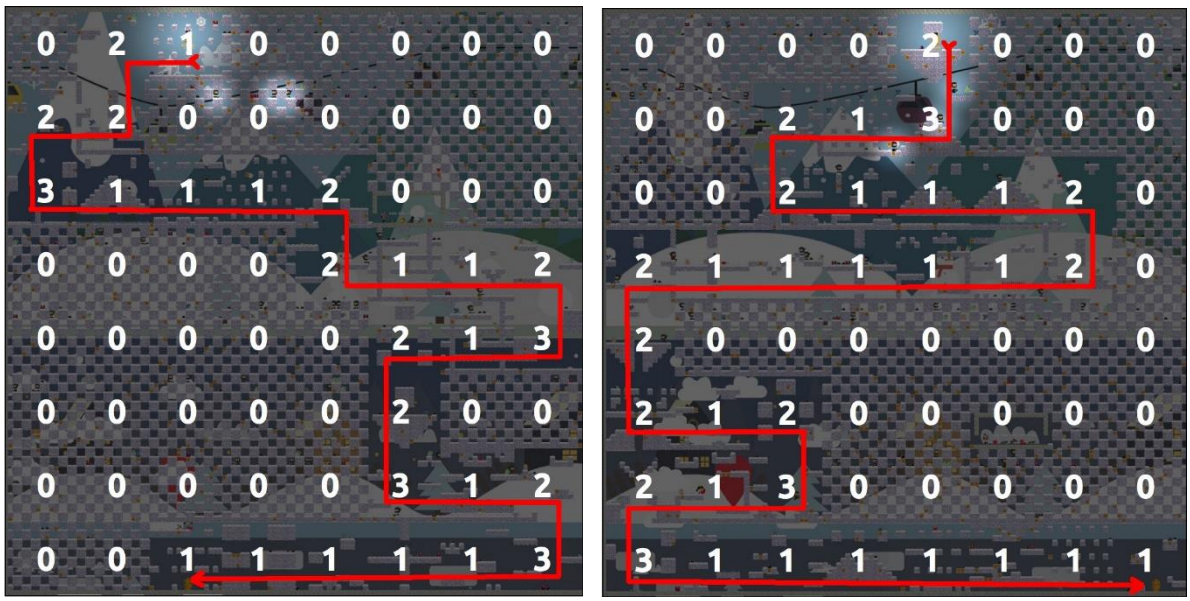
Рисунок 2.5 – Приклад генерації руху персонажа

Можна впливати на алгоритм та керувати складністю рівнів. Щоб алгоритм генерував шлях з іншими характеристиками, наприклад, менш складні або, навпаки, більш заплутані.

Остаточний вигляд рівнів зображений на рисунку 2.6.



а) б)



в) г)

Рисунок 2.6 – Приклади генерування рівнів методом запам’ятовування шляху: а) перший спосіб; б) другий спосіб; в) третій спосіб; г) четвертий спосіб

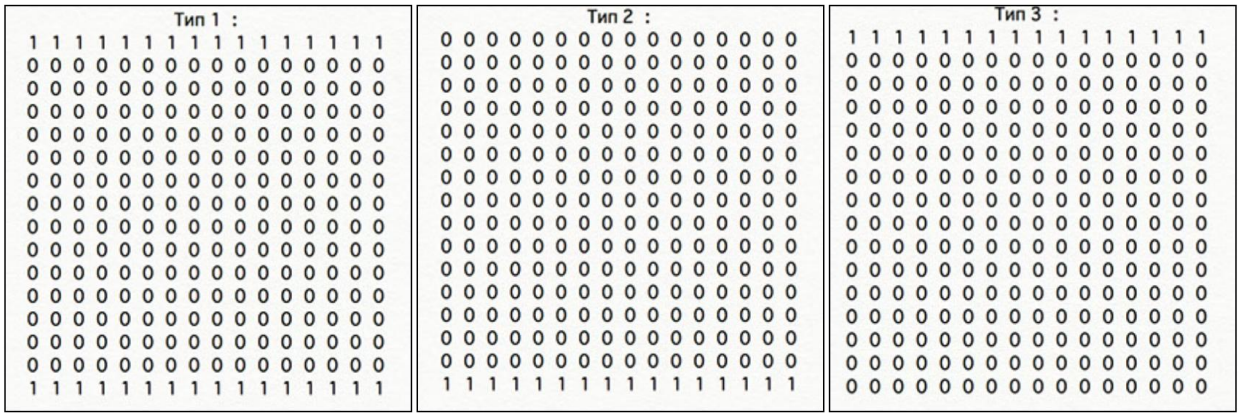
Якщо на першому пункті алгоритму вибирати випадкове число від 1 до 7, при випаданні числа від 1 до 3 – рухатися вліво, при числі від 4 до 6 – вправо, а при числі 7 – вниз, то під час генерації на кожному поверсі буде більше кімнат типу «1», це зробить шлях більш заплутаним. Випадкове число від 1 до 3, за аналогією зробить шлях менш звивистим.

Кожна кімната складається з 16×16 квадратів, всього є 4 типи кімнат, для кожного типу кімнат шаблони будуть знаходитися в окремому .plist файлі: room0.plist, room1.plist, room2.plist, room3.plist. Для початку визначимо обмеження для типів кімнат (рис. 2.7):

– кімнати типу 1 не мають виходів знизу і зверху, тобто незалежно від шаблону, виходи зверху і знизу для кімнат типу 1 можна відразу закрити;

– для кімнат типу 2 слід закрити вихід знизу;

– для кімнати типу 3 – вихід зверху.



а) б) в)

Рисунок 2.7 – Кімната: а) тип 1; б) тип 2; в) тип 3

Щоразу для кожної кімнати типу 1 програма обирає будь-який шаблон з файлу room1.plist і для цього шаблону в 50% випадків робить дзеркальне відображення зліва-направо.

При створенні карти з шаблону користуються такими правилами:

– 0 – порожній блок;

– 1 – земля, персонаж може переміщатися по цьому блоку і може його підірвати;

– 2 – камінь – блок, який не можна підірвати;

– 3 – ящик з коштовностями;

– 4 – шипи, при падінні на які гравець вмирає;

– 7 – «підшаблон» – блок висотою 3 і шириною 5;

– 8 – 75% ймовірність того, що тут буде земля. 25% – ймовірність порожнечі;

– 9 – 50% ймовірність того, що тут буде земля.

Якщо збирати карту з шаблонів, то на великих картах будуть повторювані ділянки з блоками. Потрібно зробити карти в рамках одного шаблону несхожими один на одного. Тож шаблонність можна маскувати.

Розглянемо приклад шаблонів для кімнат типу 1 на рисунку 2.8.

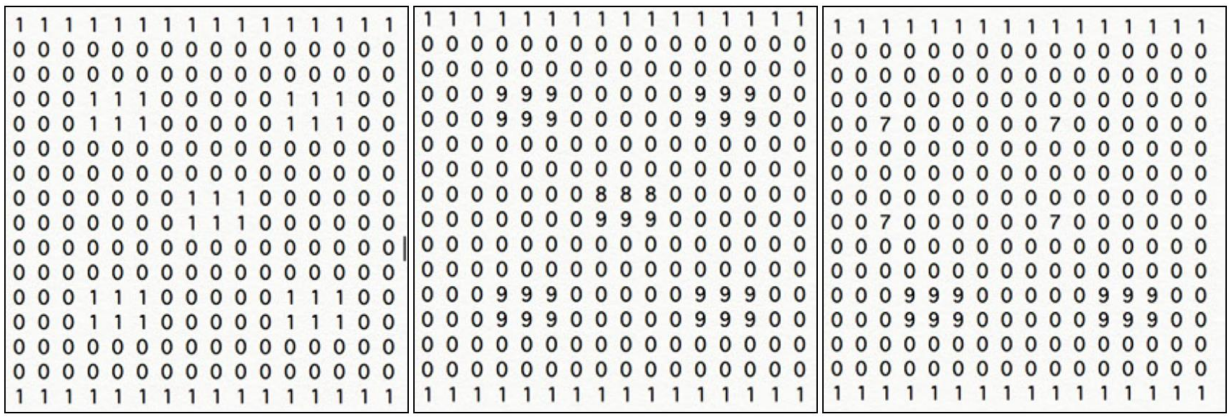


Рисунок 2.8 – Приклади шаблонів для кімнати 1 типу

Для цього, крім віддзеркалення шаблонів по горизонталі, вводяться цифри 8 і 9 в шаблонах, які при ймовірності 75% і 50% відповідно, замінюються на одиницю (блок землі). Якщо головка запису-читання натикається на цифру 7, то ця ділянка шаблону замінюється на підшаблон, який зчитується за схожими з шаблоном правилам.

Підшаблони також введені, щоб зробити два однакових шаблони несхожими один на одного [37]. Універсальними підшаблонами є класи, структури, інтерфейси і методи, які мають прототипи (параметри типів) для одного або декількох типів, які вони зберігають або використовують. Клас універсальної колекції може використовувати параметр типу заповнювач для типу об’єктів, які в ньому зберігаються. Параметри типу відображаються як типи його полів і типи параметрів його методів.

У шаблонах присутні цифри 7. Якщо алгоритм натикається на цифру 7, то він заповнює шаблон будь-яким підшаблоном з файлу boxTemplate.plist.

Приклади шалонів:

00100	11111	00100	11111
01110	01110	11111	00100
00100	00100	00100	11111

Для підшаблонів також в 50% випадків робимо дзеркальне відображення зліва-направо. Чим більше шаблонів і підшаблонов буде у грі, тим більш різноманітно будуть виглядати карти рівнів.

На рисунку 2.9 використовувався один і той самий шаблон, який завдяки цифрам 7, 8, 9 виглядає кожного разу по-різному.

Метод використання шаблонів є універсальним та підходить для створення не тільки аркадних відеоігор, а й для інших жанрів.

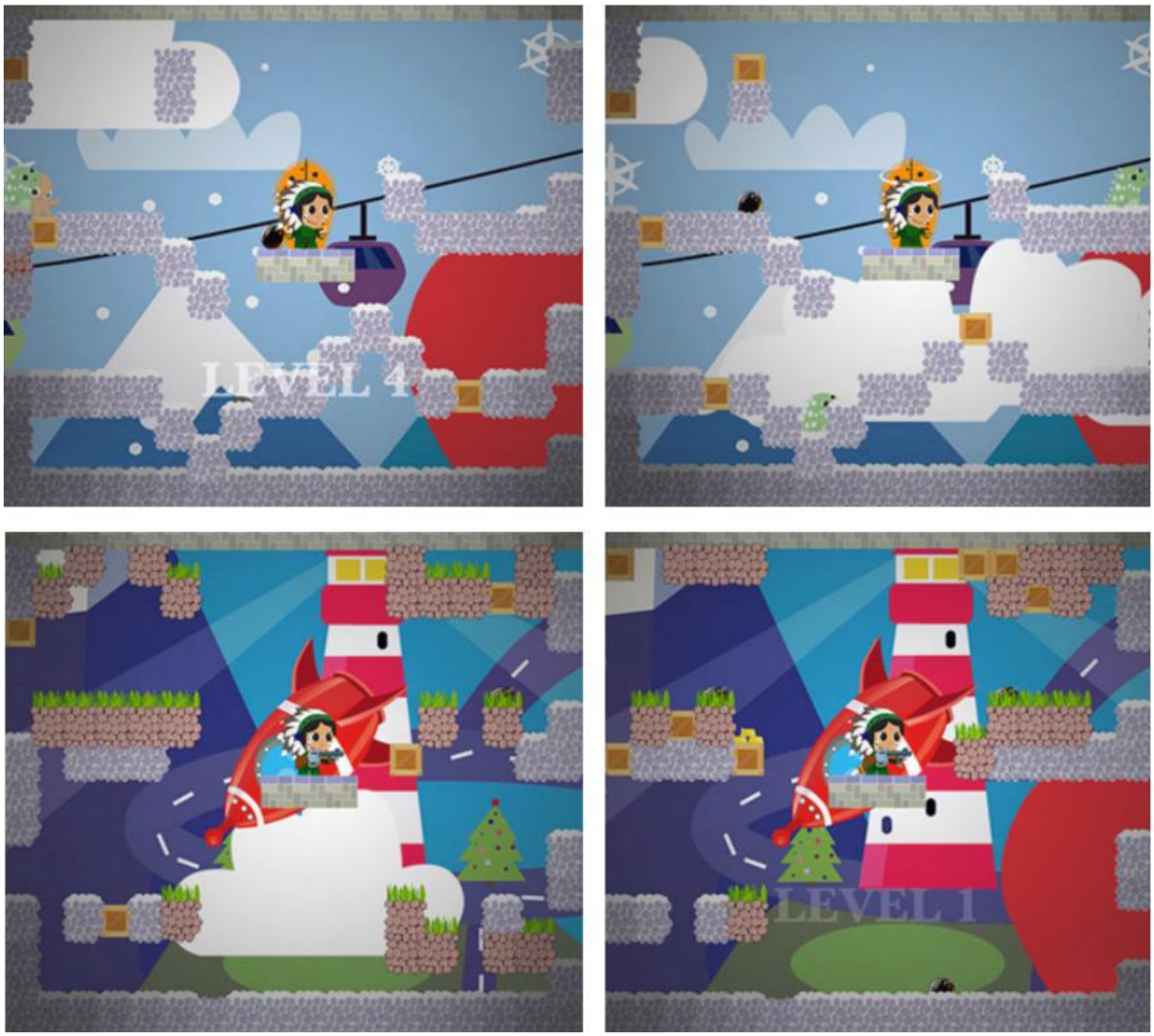


Рисунок 2.9 – Результат використання підшаблону

2.4 Розроблення структурної та функціональної моделей автоматичної генерації рівнів відеогри-платформера

Уся гра має складатися з одних й тих самих елементів, які розташовуються на рівнях в різному порядку та різній кількості.

Рівень додатку, що розробляється, обов’язково має містити:

– ігрового персонажа;

– супротивників;

– персонажа-помічника;

– пастки;

– корисні предмети, що можна відшукати (lootable items, loot) та взяти з собою на наступний рівень;

– блоки рівня;

– вхід та вихід на новий рівень.

Супротивники та персонаж-помічник називаються Non-Player Character (NPC) – це персонажі в рольових іграх, яким керує не гравець, а комп’ютер або майстер. У комп’ютерних іграх поведінку таких персонажів визначається програмно.

У комп’ютерних і настільних рольових іграх терміном «NPC» позначаються персонажі, які взаємодіють з гравцем, незалежно від їх ставлення до ігрового персонажу. NPC можуть бути дружніми, нейтральними і ворожими. Неігрові персонажі служать важливим засобом створення ігрової атмосфери, мотивують гравців здійснювати ті чи інші дії і є основним джерелом інформації про ігровому світі і сюжеті гри.

Супротивники мають атакувати головного героя, коли той потрапляє в їх радіус атаки. Персонаж-помічник в свою чергу захищає гравця, допомагає йому атакувати супротивників, забираючи на себе частину атак. Гравець може захищати помічника або залишити його, використовуючи фору часу, щоб достатися цілі.

Наявність персонажа-помічника розширює границі ігрового інтересу гравця, даючи йому можливість будувати стратегію проходження рівня з найменшими втратами або якнайшвидше, збираючи усі бонуси на рівні або вбиваючи якнайбільше супротивників. Це також розширить варіанти підготовки до рівня, адже з’явиться можливість покращувати не тільки свого персонажа, але й помічника.

Пастки в свою чергу можуть впливати на гравця в різні способи:

– забирати частину здоров’я або ціле життя;

– затримувати гравця на декілька ходів;

– випускати на ігрове поле нового супротивника;

– блокувати якусь здібність персонажа. Наприклад, збирати бонус, відновляти здоров’я, атакувати;

– відкидати гравця в довільне місце, далі від цілі.

Функціональна модель головного персонажа схематично описана на рис 2.10.

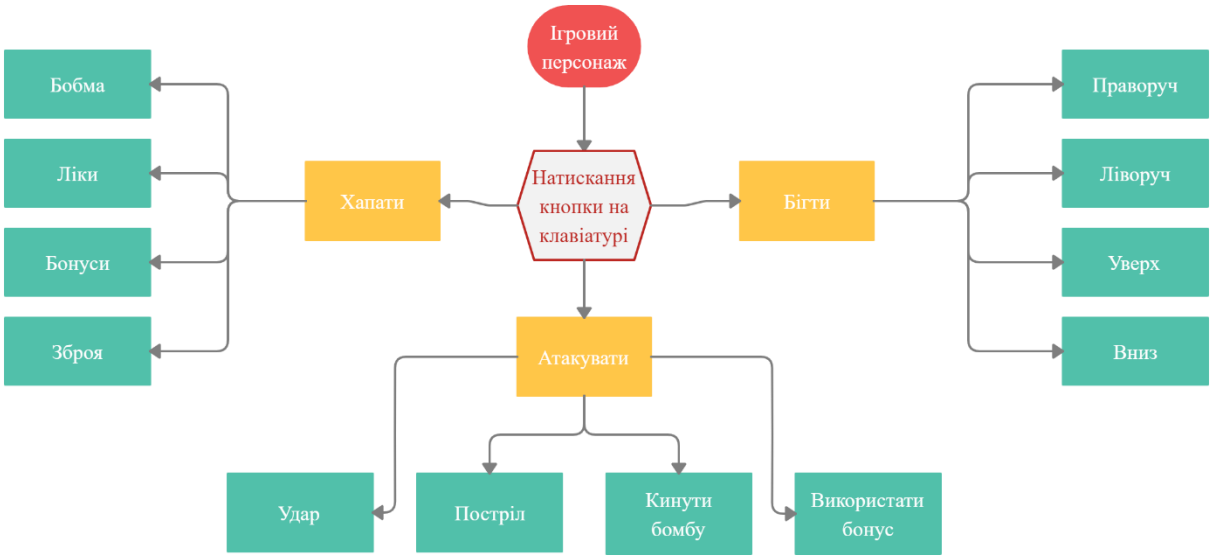


Рисунок 2.10 – Функціональна модель персонажа

3 ДОСЛІДЖЕННЯ МЕТОДУ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ РІВНІВ ДЛЯ ВІДЕОГРИ-ПЛАТФОРМЕРА

3.1 Вибір інструментальних засобів та інформаційних технологій для створення алгоритму автоматичної генерації рівнів у відеогрі-платформері

У рамках передатестаційної практики була розроблена гра-платформер для ПК. Для реалізації було обране середовище Unity. Це обумовлено тим, що в Unity вбудовані методи для описання фізичних властивостей об’єктів.

Unity – багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосунки працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.

Є можливість створювати інтернет-додатки за допомогою спеціального під’єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

3.1.1 Технічні характеристики Unity

Технічні характеристики Unity:

– сценарії на C#, JavaScript та Boo;

– ігровий рушій повністю пов’язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі;

– робота з ресурсами можлива через звичайний Drag&Drop.

– система успадкування об’єктів;

– підтримка імпортування великої кількості форматів файлів;

– вбудований генератор ландшафтів;

– вбудована підтримка мережі;

– існує рішення для спільної розробки – Asset Server. Також можна використовувати зручний для користувача спосіб контролю версій. Наприклад, SVN або Source Gear.

3.1.2 Функціональні можливості

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C#, JavaScript (модифікація), Boo. Проект в Unity ділиться на сцени (рівні) – окремі файли, що містять свої ігрові світи зі своїм набором об’єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об’єкти (моделі), так і порожні ігрові об’єкти – тобто ті, які не мають моделі. Об’єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у них є назва (в Unity допускається наявність двох і більше об’єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов’язково присутній компонент Transform – він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об’єктів з видимою геометрією також за умовчанням присутній компонент Mesh Renderer, що робить модель видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об’єктів; дочірні об’єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об’єкта. Скрипти в редакторі прикріплюються до об’єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на

модель текстуру прикріпити не можна – буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того, він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

3.1.3 Рендеринг

Графічний рушій використовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власне API для Wii [38]. Також підтримуються bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічні тіні з використанням shadow maps, render-to-texture та повноекранні ефекти post-processing.

Unity підтримує файли 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance.

В ігровий проект Unity можна імпортувати об’єкти цих програм та робити налаштовування за допомогою графічного інтерфейсу.

Для написання шейдерів використовується ShaderLab, що підтримує шейдерні програми написані на GLSL або Cg. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант для конкретної відеокарти.

Unity також має вбудовану підтримку фізичного рушія Nvidia PhysX (колишнього Ageia), підтримку симуляції одягу в системі реального часу на довільній та прив’язаній полігональній сітці (починаючи з Unity 3.0), підтримку системи ray casts та шарів зіткнення.

3.1.4 Скрипти

Скриптова система ігрового рушія зроблена на Mono – вільний відкритий проект з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python). Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для зневадження скриптів.

У версії 5.2 вбудована можливість редагувати скрипти у середовищі Visual Studio [39].

3.1.5 Asset Tracking

В Unity включено систему контролю версій Unity Asset Server для ігрових об’єктів та скриптів. Система використовує PostgreSQL, роботу зі звуком, побудовану на основі бібліотеки FMOD, відеопрогравач із кодеком Theora, рушій для побудови ландшафтів рослинності, вбудовану систему карт освітлення, мережу для мультиплеєру (RakNet) та вбудовані навігаційні меші для пошуку шляху [40].

3.2 Програмна реалізація

Процес створення даної відеогри поділяється на такі етапи:

– створення сцен;

– створення зображень об’єктів;

– створення анімацій об’єктів;

– впровадження об’єктів у середовище розробки шляхом присвоювання їм маси, швидкості, розмірів, положення на екрані;

– створення коллайдерів для кожного об’єкта;

– створення тригерів на початковому та кінцевому блоках, тригеру програшу;

– створення методу автоматичної генерації рівня;

– створення методу розміщення ворогів на рівні;

– створення методу розміщення бонусів на рівні;

– реалізація створених алгоритмів.

3.2.1 Створення сцен

У додатку було створено чотири сцени:

– Defeat – сцена програшу;

– Game – основна сцена гри;

– Menu – меню гри;

– Win – виграш.

На рисунку 3.1 зображена послідовність дій для створення нової сцени.

У кожній сцені є SceneController. Він відповідає за відкриття всіх необхідних посилань і ініціалізацію ключових об’єктів. У певному сенсі, його можна вважати точкою входу сцени. Для подання аргументів використовується клас SceneArgs і у кожної сцени є свій клас, що представляє її аргументи і є спадкоємцем SceneArgs.

Під час виклику OpenSceneWithArgs () передається тип контролера (TController) сцени, яку потрібно завантажити, тип параметрів (TArgs) і, власне, самі параметри (sceneArgs). В першу чергу, SceneManager перевіряє, чи є у TController атрибут SceneControllerAttribute. Він повинен бути, тому саме він визначає, до якої сцен прив’язаний контролер TController. Якщо все пройшло без помилок, то буде викликаний метод з Unity API LoadSceneAsync () і йому буде передано ім’я сцени, яке береться з атрибуту SceneControllerAttribute.

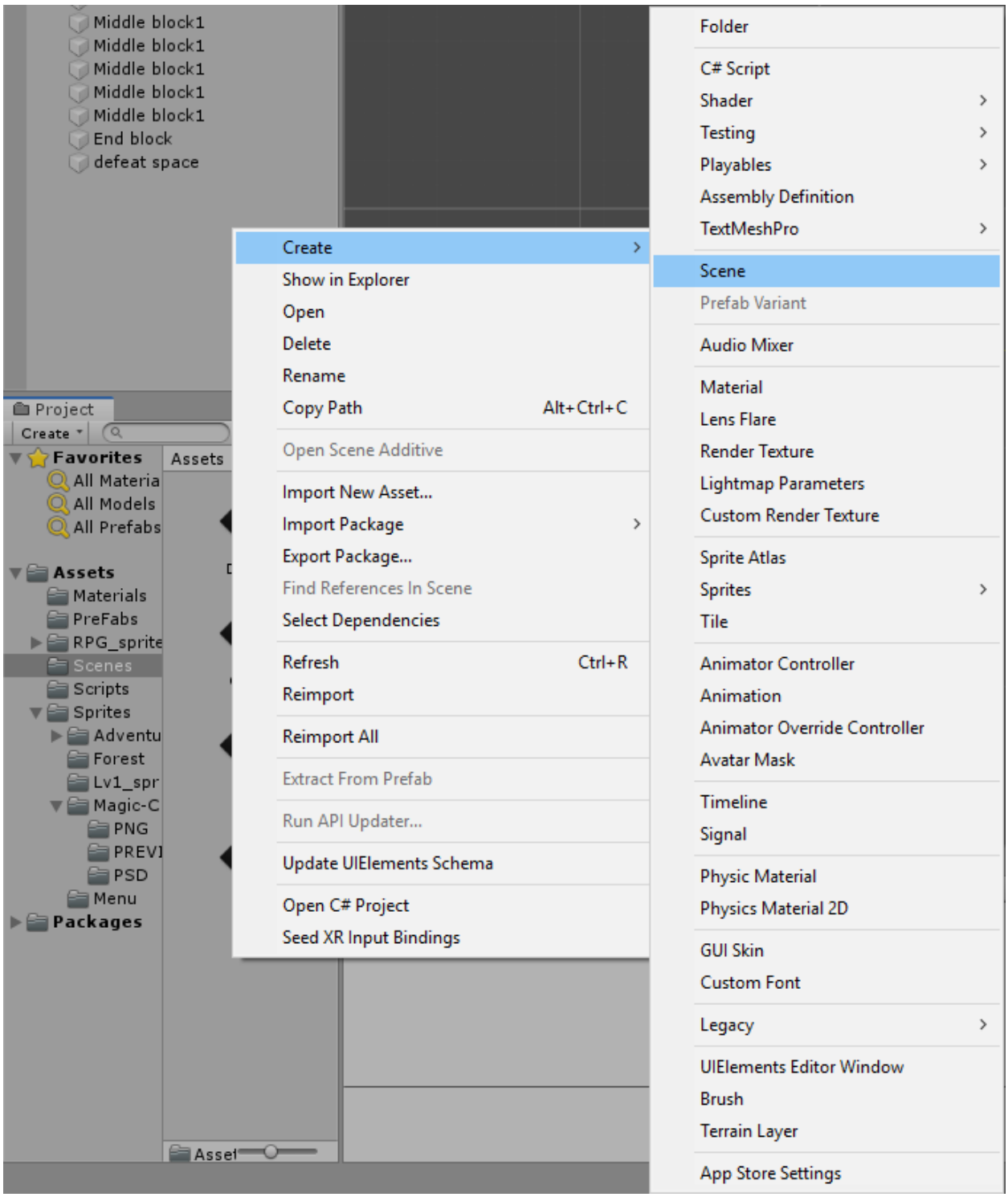


Рисунок 3.1 – Створення сцени

На сцені меню (рис. 3.2) реалізується скрипт для переходу на сцену гри да допомогою кнопки Play, налаштувань – Settings, авторів гри – Credits та вихід з программи – Exit.

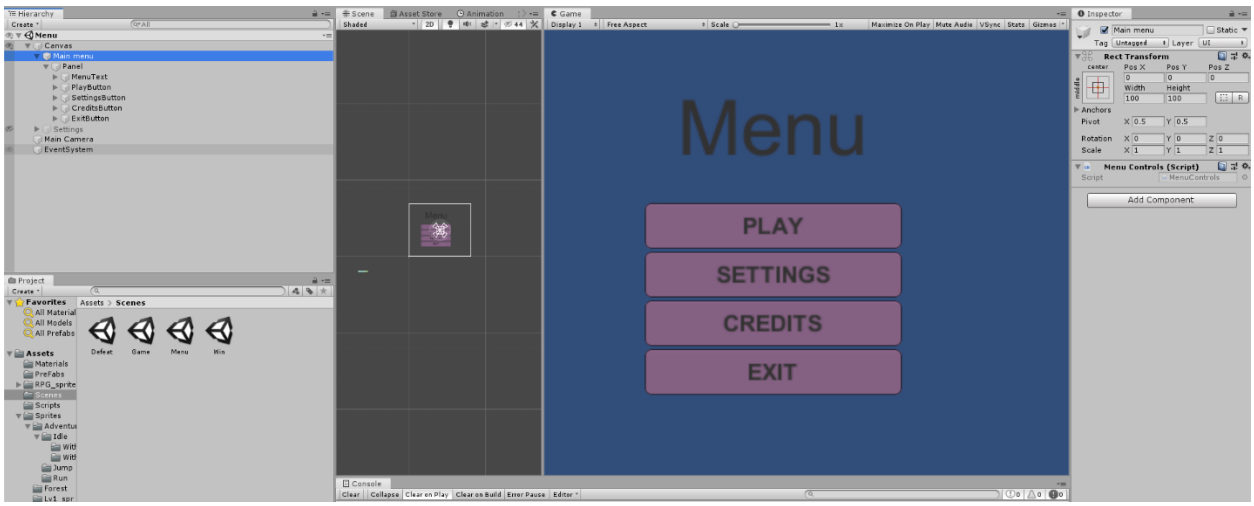


Рисунок 3.2 – Сцена меню

На сцені гри (рис. 3.3) реалізуються основні скрипти (контроль персонажа, генерація рівня, тощо) та деякі об’єкти, які не входять до процесу генерації (фон, камера, персонаж). Сцену гри можна модифікувати вручну, переміщуючи елементи за домопогою миші або змінюючи параметри у спеціальному вікні Inspector.

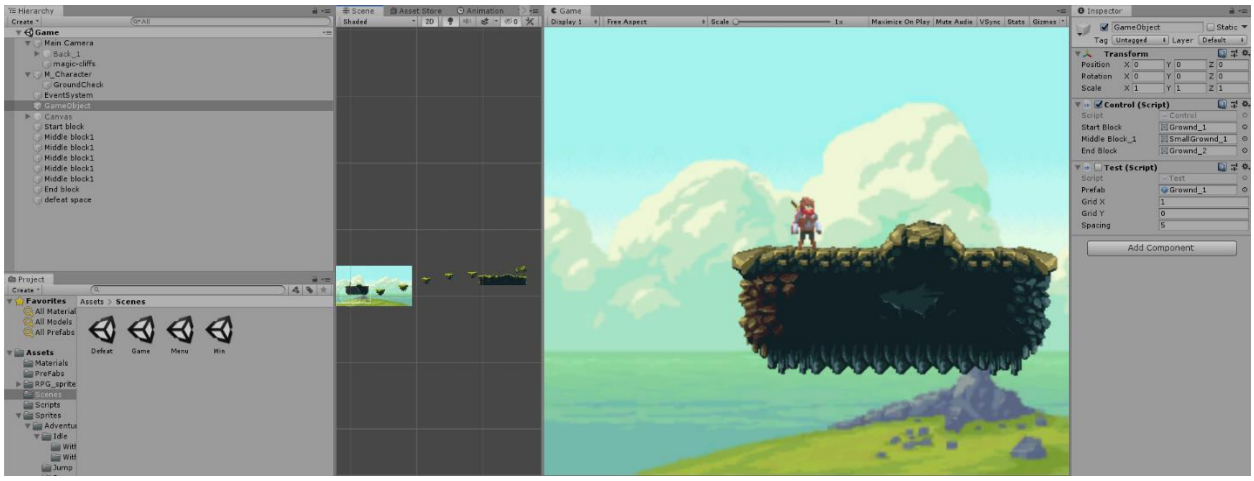


Рисунок 3.3 – Сцена гри

На сцені програшу (рис. 3.4) реалізується скрипт для переходу в головне меню за допомогою кнопки Menu та перезапуску рівня – Restart. Під час перезапуску складність рівня не підвищується, а гравець знову потрапляє на блок початку рівня.

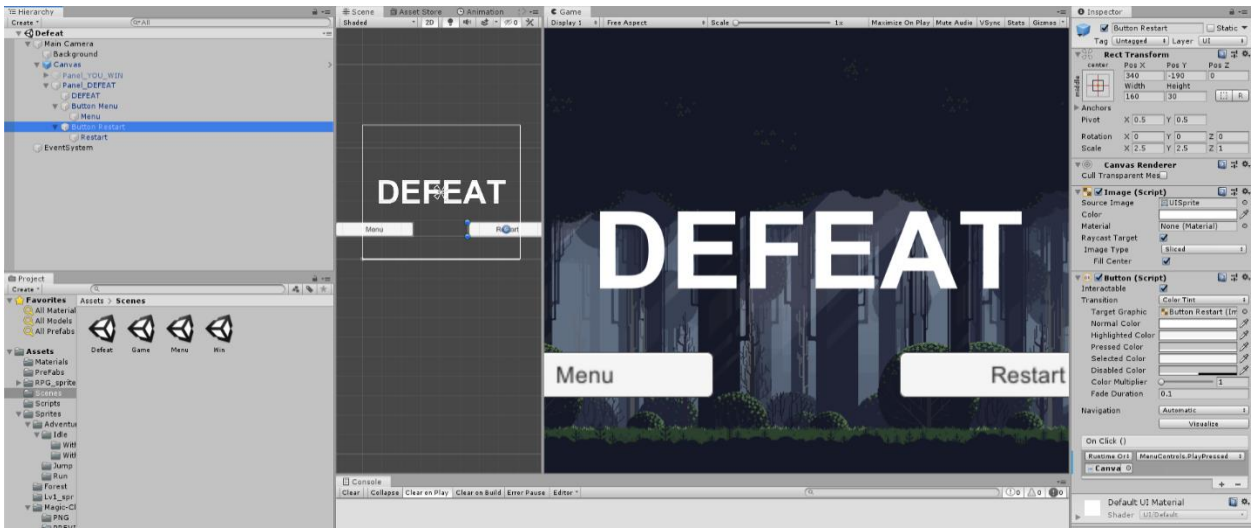


Рисунок 3.4 – Сцена програшу

На сцені перемоги (рис. 3.5) реалізується скрипт для переходу на наступний рівень за допомогою кнопки Next Level і меню – Menu. Під час переходу на наступний рівень з цієї сцени гравець потрапляє на початковий блок, а складність рівня подвищується. Складність рівня обчислюється кількістю блоків на шляху від початку гри до кінця.

Кожен рівень відрізняється унікальною кількістю та положенням блоків У розробляємій відеогрі не може бути непрохідних рівнів, оскільки ця можливість обмежена алгоритмом.

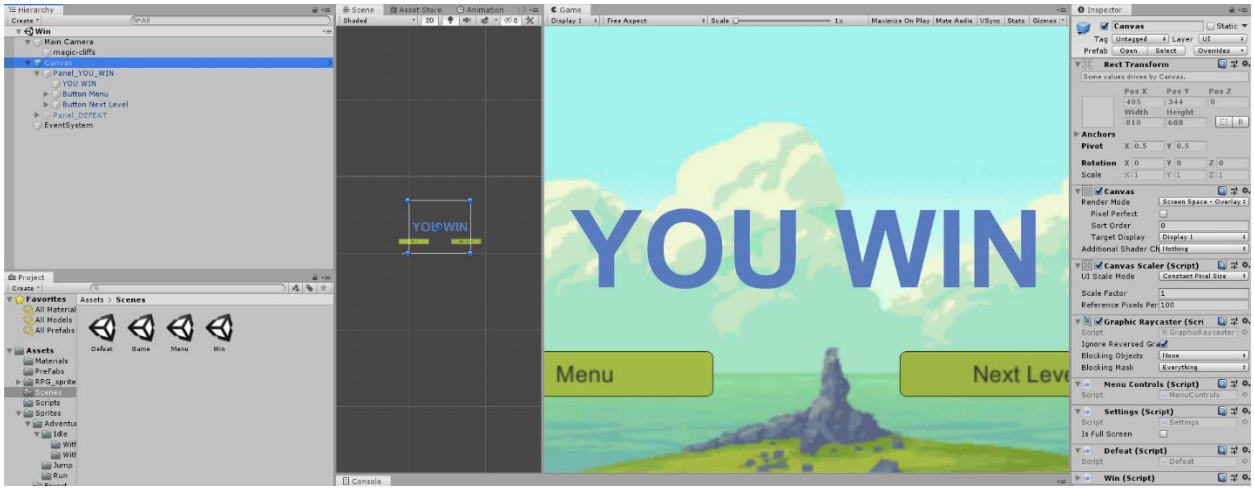


Рисунок 3.5 – Сцена перемоги

3.2.2 Створення зображень об’єктів

Створюється папка Sprites. У ній будуть зберігатися зображення, які використовуються в даному додатку. Створити зображення можна у контролері Inspector (рис. 3.6).

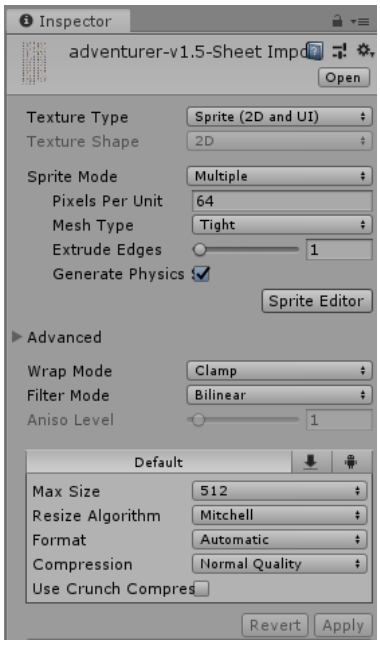


Рисунок 3.6 – Контроллер властивостей зображення

Можна використовувати готовий пакет зображень, з якого за допомогою інструменту Sprite Editor (рис. 3.7) вирізати потрібні.

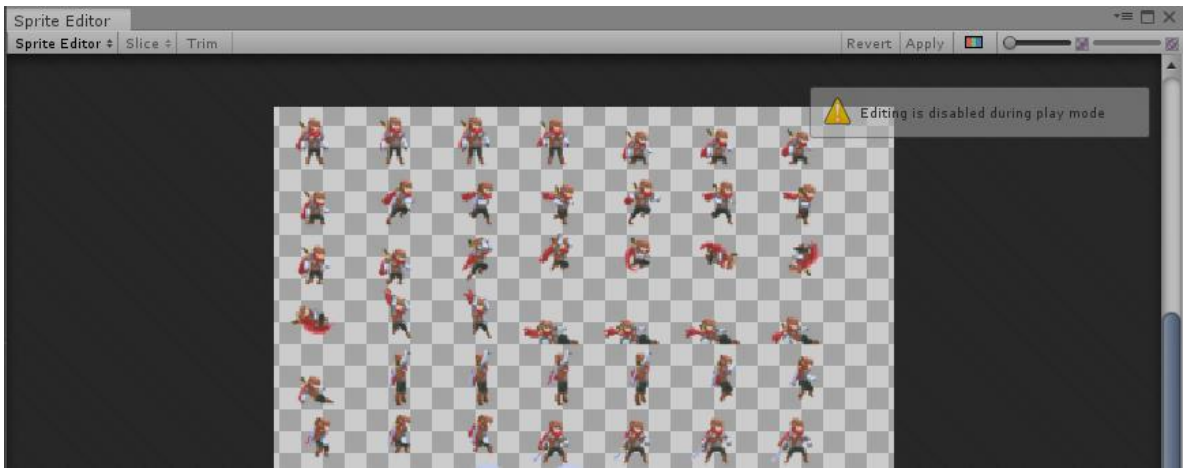


Рисунок 3.7 – Меню редактора зображень

3.2.3 Створення анімації об’єктів

Щоб створити аніміцію, обрається персонажа Character у вікні Hierarchy. Далі у вікні Animation натискається кнопка для створення нової анімації і оберається Create New Clip (рис. 3.8).

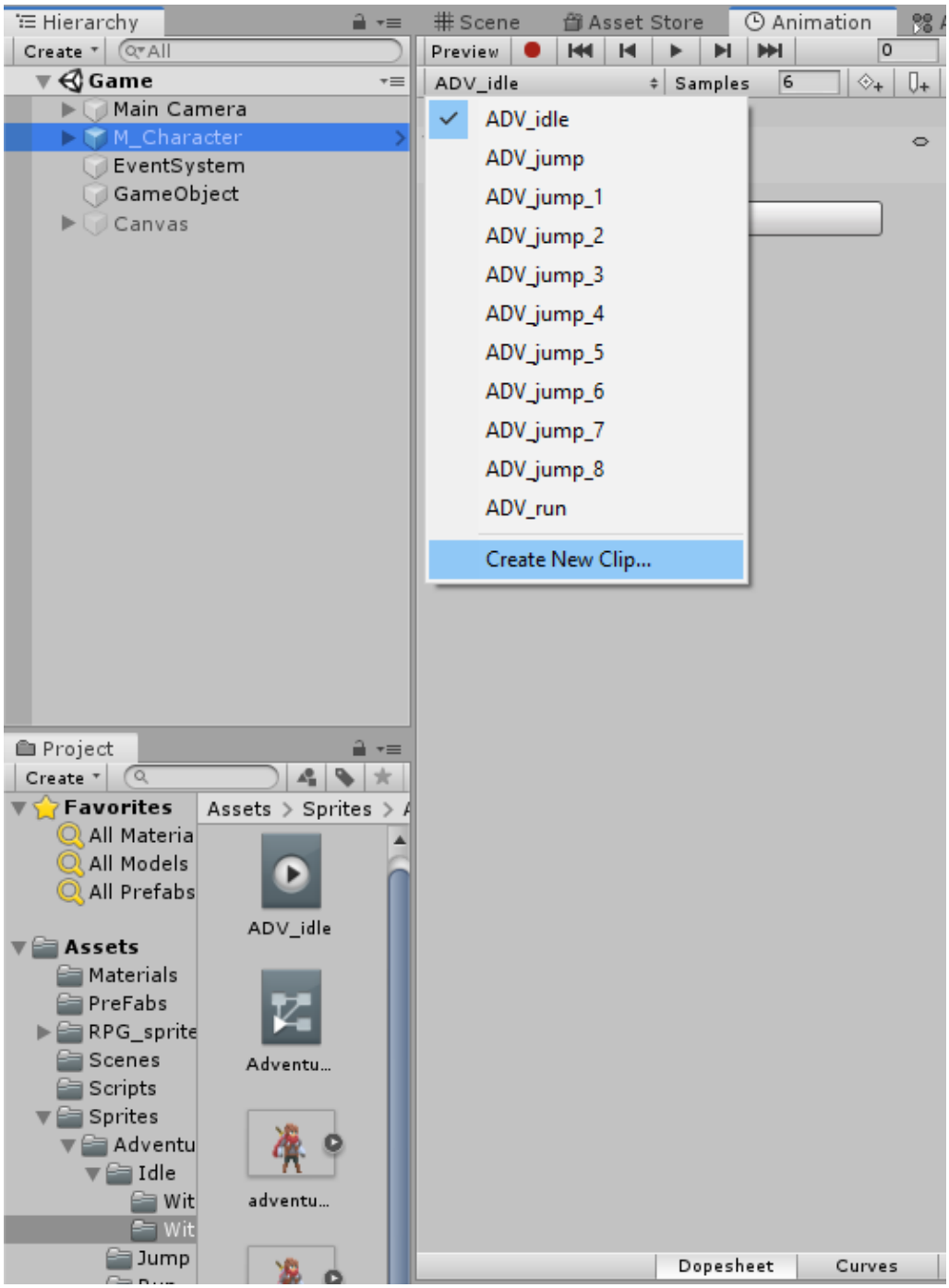


Рисунок 3.8 – Створення анімації

У стандартному діалозі збереження файлу створюється папка Animations, у якій будуть зберігатися файли анімації (рис. 3.9).

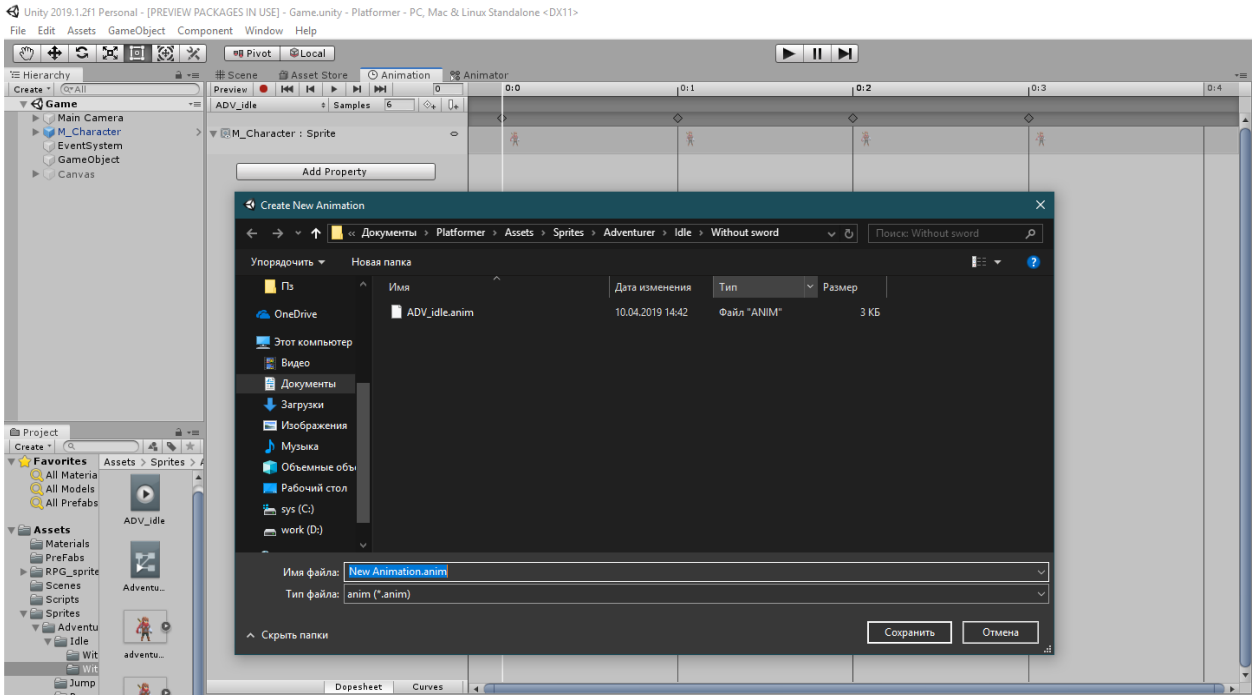


Рисунок 3.9 – Збереження анімації

Після збереження, у вікні Animator з’явиться анімація Idle у вигляді прямокутника помаранчового кольору. Помаранчевий колір означає, що анімація буде за замовчуванням – вона є стартовою для персонажа.

У папці Sprites розгортається спрайт Idle, виділяється перше зображення Idle_0, затискається Shift і виділяється останнє зображення Idle_7. Усі виділені зображення копіюються у вікно Animation. Задається значення Sample 6 – цей параметр означає кількість кадрів анімації в секунду.

Для якісної анімації необхідно, щоб вона відображалася зі швидкістю не менше 24 кадрів в секунду, проте, в створюваному додатку анімація складається з невеликого числа кадрів, тож за значення 24 вона буде відображатися занадто швидко.

Аналогічно за прикладом анімації спокою зроблені анімації бігу і стрибка з обертанням у повітрі.

У вікні Animator (рис. 3.10) відображені чотири анімації: Any State, ADV_idle, ADV_run і Jump. Задаються умови переходу з анімації ADV_idle в анімацію ADV_run, тобто зі стану спокою в стан бігу. У нижньому лівому кутку у полі Parameters натискається кнопка «плюс», обирається Float і новому параметру задається назва Speed. Створився параметр типу «число з плаваючою комою», що позначає швидкість переміщення персонажа. Саме в залежності від значення цього параметра буде відбуватися перемикання з анімації спокою в анімацію бігу. Далі натискається правою кнопкою миші на анімацію ADV_idle, обирається Make Transition і натискається лівою кнопкою миші на анімацію ADV_run. Між анімаціями з’явиться лінія зі стрілкою. Натискається лінія зі стрілкою.

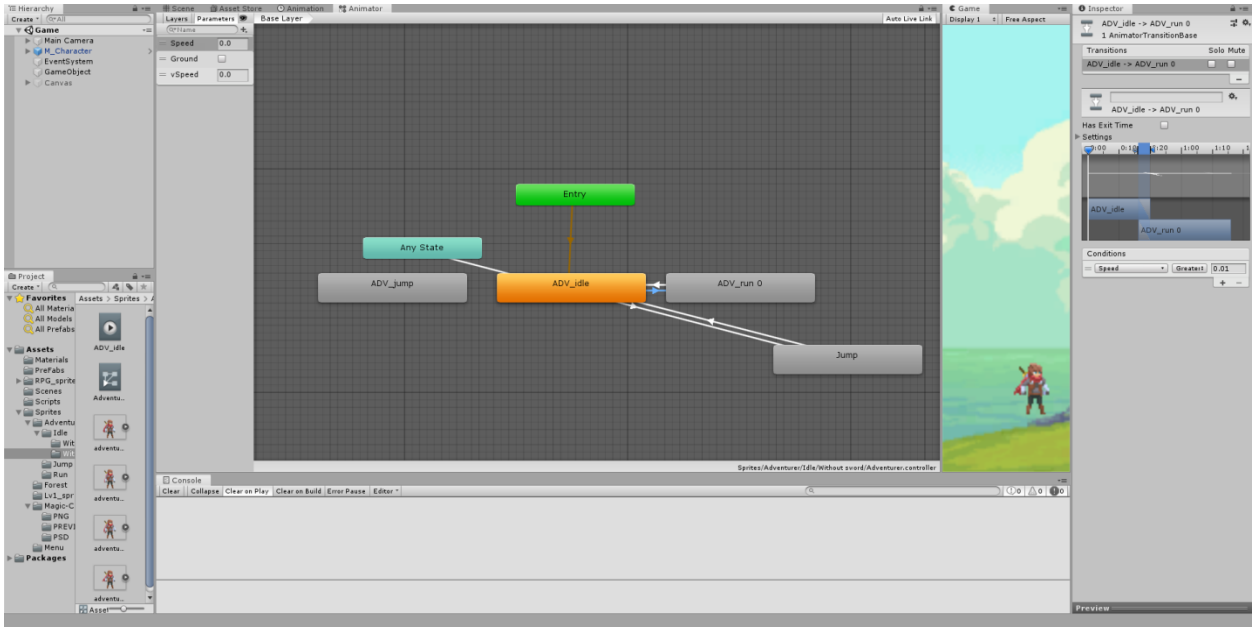


Рисунок 3.10 – Діаграма змінення станів анімацій

У вікні Inspector відобразяться властивості переходу між анімаціями. Далі треба перейти в розділ Conditions. Обирається параметр Exit Time і моняється на Speed. Друге поле Greater залишається без змін, а в третьому вводиться значення 0.01.

Створена умова переходу із анімації спокою в анімацію бігу – вона відбувається, коли значення параметра швидкості стає трохи більше нуля.

3.2.4 Створення об’єктів

Існує два основних способи створення об’єктів у Unity :

– зробити вручну модель у середовищі Unity та перетягнути її з вікна Game у папку Prefab (рис. 3.11). Після цього код створеного об’єкту з’явиться у Visual Studio;

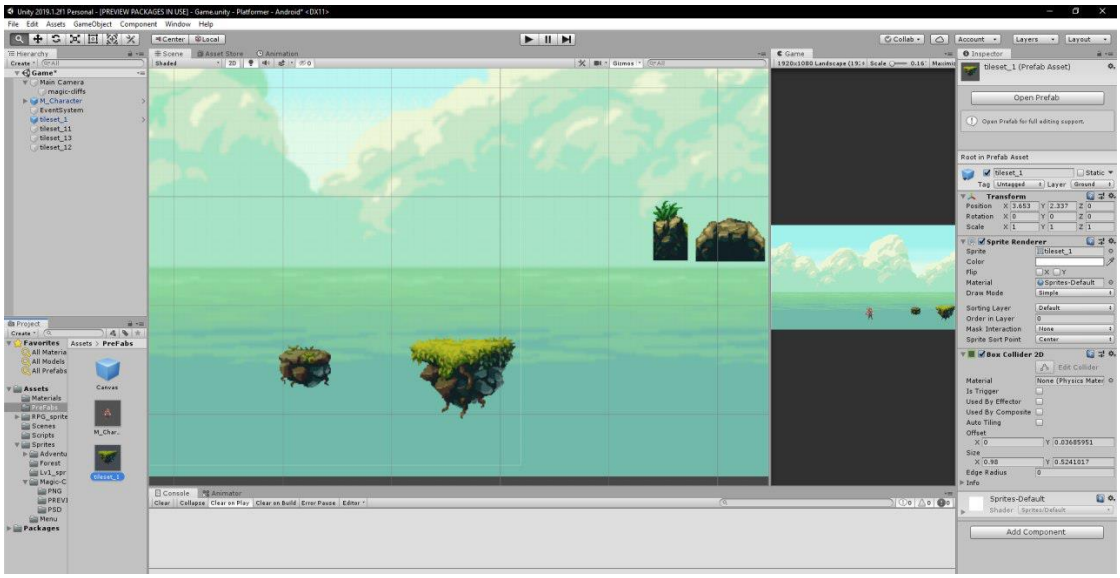


Рисунок 3.11 – Створення префабу

– написати код об’єкту в Visual Studio (рис. 3.12). Створити порожній об’єкт на сцені гри і додати до нього скрипт (рис. 3.13).

```
newBlock = new GameObject("Broken brige");
renderer = newBlock.AddComponent<SpriteRenderer>();
renderer.sprite = this.brokenbrige;

BoxCollider2D brokenbrige1 = newBlock.AddComponent<BoxCollider2D>();

var bzone1 = newBlock.GetComponent<BoxCollider2D>();
brokenbrige1.offset = new Vector2(-0.38f, 0);
brokenbrige1.size = new Vector2(0.2f, 0.3f);

BoxCollider2D brokenbrige2 = newBlock.AddComponent<BoxCollider2D>();

var bzone2 = newBlock.GetComponent<BoxCollider2D>();
brokenbrige2.offset = new Vector2(0.38f, 0);
brokenbrige2.size = new Vector2(0.2f, 0.3f);

newBlock.layer = 8;
```

Рисунок 3.12 – Код нового об’єкту

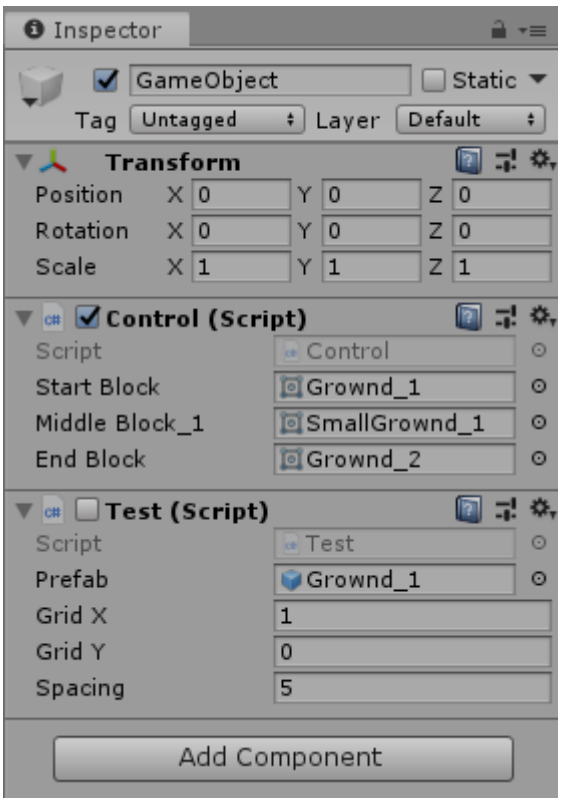


Рисунок 3.13 – Скрипт Control

3.2.5 Створення коллайдерів та тригерів

Для присвоювання об’єкту нового коллайдера використовується метод Block.AddComponent<BoxCollider2D>(). Положення коллайдеру задається зміненням параметру за допомогою collider.offset = new Vector(*x*,*y*), де *x* – це положення на осі *X*, а *y* – на осі *Y*. Розмір змінюється параметром collider.size = new Vector(*x*, *y*)

Створення активного тригеру виконується за допомогою змінення параметра командою defeat.isTriggered = true. Потрібно створити методи, які активуються під час потрапляння гравця на тригер.

На рисунку 3.14 зображені методи для тригеру програшу. Коли об’єкт гравця стає на тригер програшу, він знищується, а сцена замінюється на сцену Defeat. При використанні SceneManager.LoadScene завантаження не відбувається відразу, воно завершується в наступному кадрі. Така

напівсінхронна поведінка може викликати загальмованість кадру і може призвести до плутанини, оскільки завантаження не завершується негайно.

```
public class Defeat : MonoBehaviour
{
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            Destroy(other.gameObject,.2f);
            SceneManager.LoadScene("Defeat");
        }
    }
}
```

Рисунок 3.14 – Методи тригеру програшу

Оскільки завантаження встановлене для завершення в наступному візуалізованому кадрі, виклик SceneManager.LoadScene змушує завершити всі попередні операції, навіть якщо AsyncOperation.allowSceneActivation встановлено в false. Цього можна уникнути, використовуючи замість LoadSceneAsync.

3.2.6 Реалізація методу автоматичного створення рівня

Розроблений метод створюється на основі алгоритму генерації випадкових чисел з обмеженнями та відповідати таким умовам:

– блоки землі не повинні бути нижче координати y = -3;

– зона програшу має бути нижче усіх інших блоків рівня;

– зона програшу має створюватися завдовжки в півтори довжини усього рівня;

– відстань між блоками не може бути більше відстані стрибка гравця;

– кожен наступний рівень повинен мати в собі більше елементів, аніж попередній.

Позиція на вісі *X* кожного наступного блоку розраховується відносно позиції попереднього блоку за наступною формулою:

$$px = sx + 2,$$

де *px* – це положення нового блоку на осі *x*;

sx – довжина попереднього блоку.

Позиція на вісі *Y* кожного блоку розраховується за алгоритмом:

– якщо позиція попереднього блоку більше аніж мінус 3, використовується формула:

$$py = sy * \eta,$$

де *py* – це позиція попереднього блоку на вісі *Y*;

sy – висота попереднього блоку;

η – випадкова величина у діапазоні (-0,5; 1), що обчислюється за нормальним законом розподілу:

$$f(x;\ \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

де μ– математичне сподівання;

σ^2 – дисперсія випадкової величини. Параметр σ також називається стандартним відхиленням;

– якщо позиція попереднього блоку менше або дорівнює мінус 3, діапазон випадкової величини змінюється на (-0,5; 0,5);

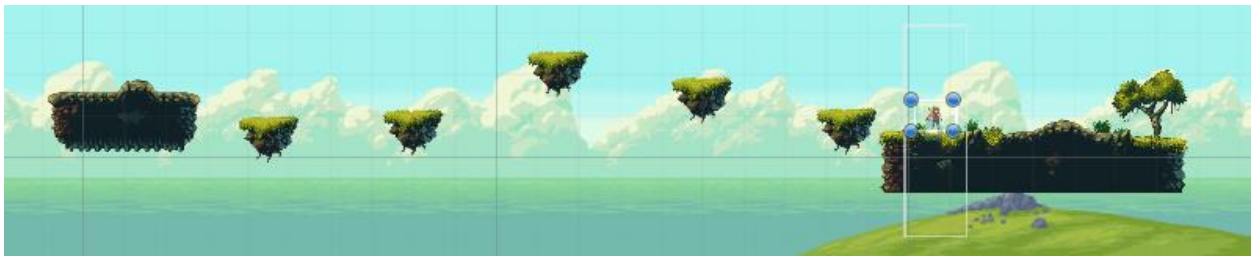
– вороги розташовуються на рівні рівномірно. Їх кількість не перевищує довжині рівня *l*/10.

Від’ємне значення випадкової величини при множенні на від’ємне значення позиції попереднього блоку у результаті дає додатне значення позиції нового блоку.

Зона програшу є тригером, що знаходиться на позначці мінус 4 но вісі *Y*. Він має стандартну задану вручну висоту, а його довжина обчислюється за формулою:

$$l = 1{,}5 * s,$$

де s – це різниця між координатами останнього блоку та першого – тобто довжина рівня. Вона змінюється кожен раз, коли попередній рівень успішно пройдено гравцем (рис. 3.15).



а)

б)

Рисунок 3.15 – Змінення складності гри: а) перший рівень; б) другий рівень

Умови розташування бонусів на рівні:

– якщо відстань між безпечними для гравця місцями на блоках більше довжини стрибка персонажу, на нижчому блоці встановлюється «бонус стрибка». Якщо блоки знаходяться на одній висоті, бонус має бути з обох сторін порожнечі;

– кожна пастка та ворог мають змінну, яка відповідає їх складності відносно один від одного. Ці змінні сумуються. Коли сума добігає значення 20, з’являється «бонус здоров’я»;

– «бонус здоров’я» також може бути схований в місцях великого скупчення пасток в елементах декору, які можна зруйнувати. В такому випадку бонус анімується та інколи з’являється в полі зору гравця;

– «бонус додаткового життя» схований в найвищій доступній точці рівня на відстані (20, 50) від ворога;

– елементи декору можуть з’явитися тільки в логічних місцях. Наприклад, кактус може бути розташованим на блоці з сухою землею, з меншою вірогідністю – на траві, а ніколи – на металевому блоці;

– на кожній третій частині рівня розташовані «будинки збереження». Якщо гравець помирає, він почне не з початку рівня, а з останнього будиночку.

На рис. 3.16 – рис. 3.18 зображені приклади розташування на рівні ворогів, пасток, бонусів та інших предметів.

Рисунок 3.16 – Приклад генерації рівня

Рисунок 3.17 – Приклад генерації рівня

Бонуси зазвичай ховаються за декором, що можна зруйнувати. Вони трохи рухаються, щоб привернути увагу гравця. Декор може захистити персонажа, але вороги можуть його зруйнувати.

Рисунок 3.18 – Приклад генерації рівня

3.3 Інструкція користувача

Щоб запустити додаток, треба обрати файл platformer.exe та двічі натиснути на нього лівою кнопкою миші. Відкриється вікно Platformer Configuration (рис. 3.19), у якому можна налаштувати гру у відповідності до характеристик вашого ПК. Опція Screen resolution (рис. 3.20) дозволяє налаштувати розмір вікна додатку відповідно до роздільної здатності екрану. Щоб перевести гру у віконний режим, треба поставити прапорець біля функції Windowed.

Graphics quality – це налаштування якості зображення (рис. 3.21). Від вибору режиму залежить різкість, яскравість, конрастність та головне – деталізованість отримуємої картинки. Якщо до ПК підключено декілька моніторів, за допомогою опції Select monitor можна обрати потрібний та вивести на нього зображення. У вкладці Input можна ознайомитися з клавішами керування персонажем. Після закінчення налаштування додатку треба натиснути кнопку Play.

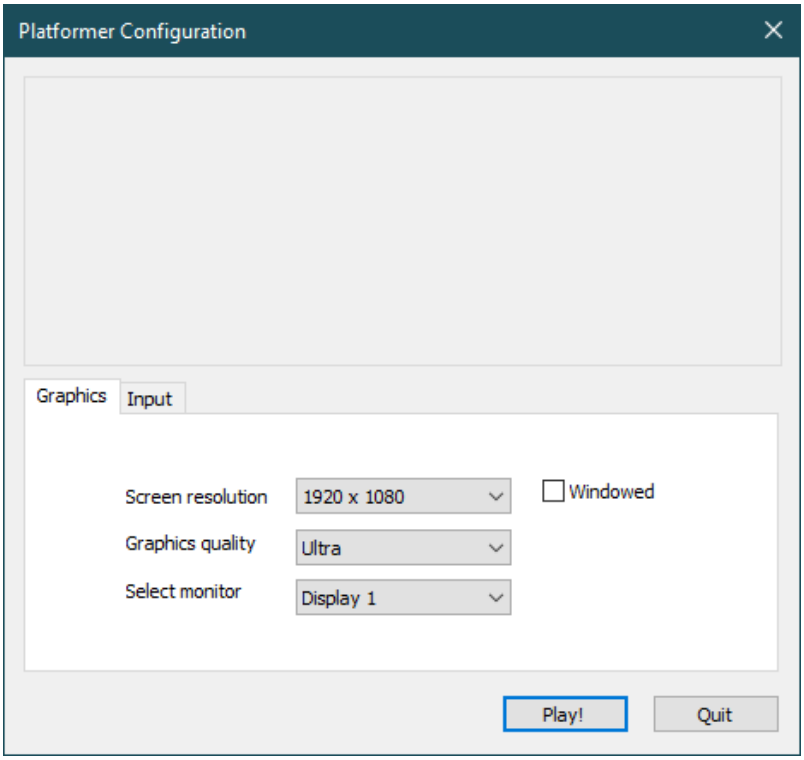


Рисунок 3.19 – Вікно налаштувань додатку

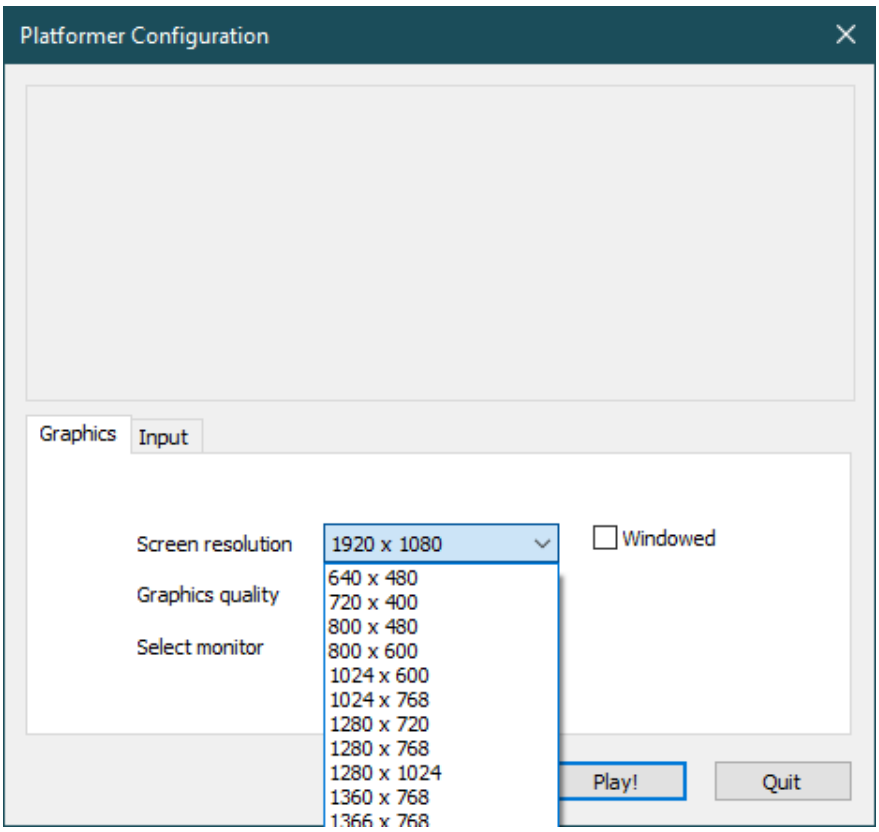


Рисунок 3.20 – Налаштування розміру вікна

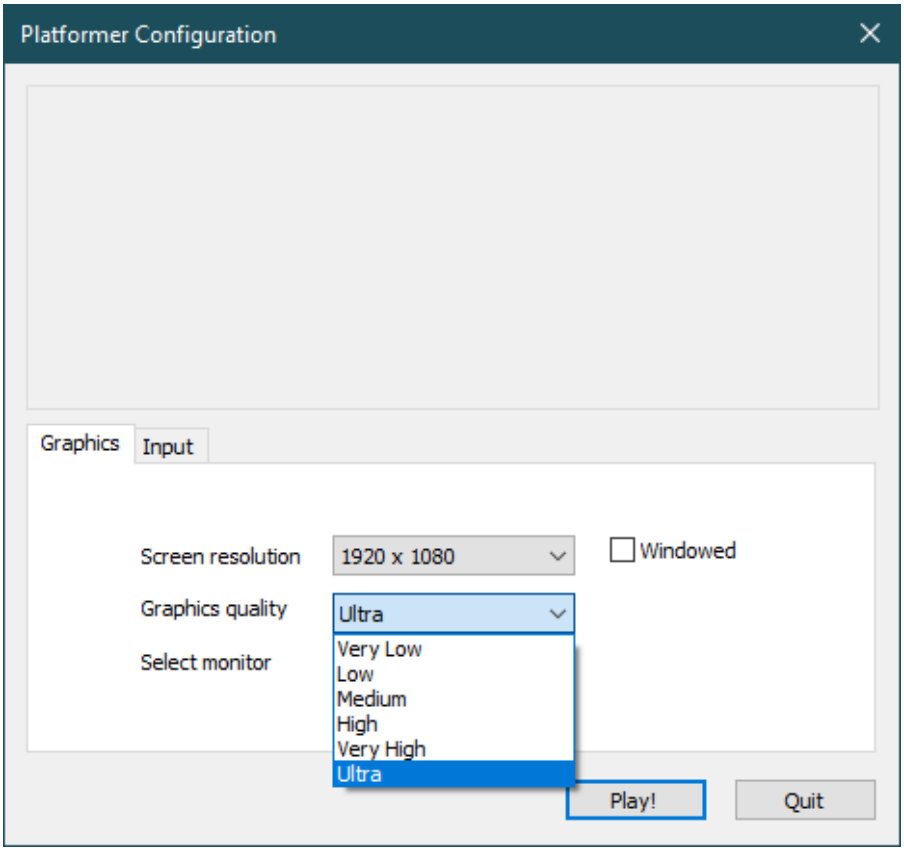


Рисунок 3.21 – Налаштування якості зображення

У вікні меню гри натиснути кнопку Play (рис. 3.22).

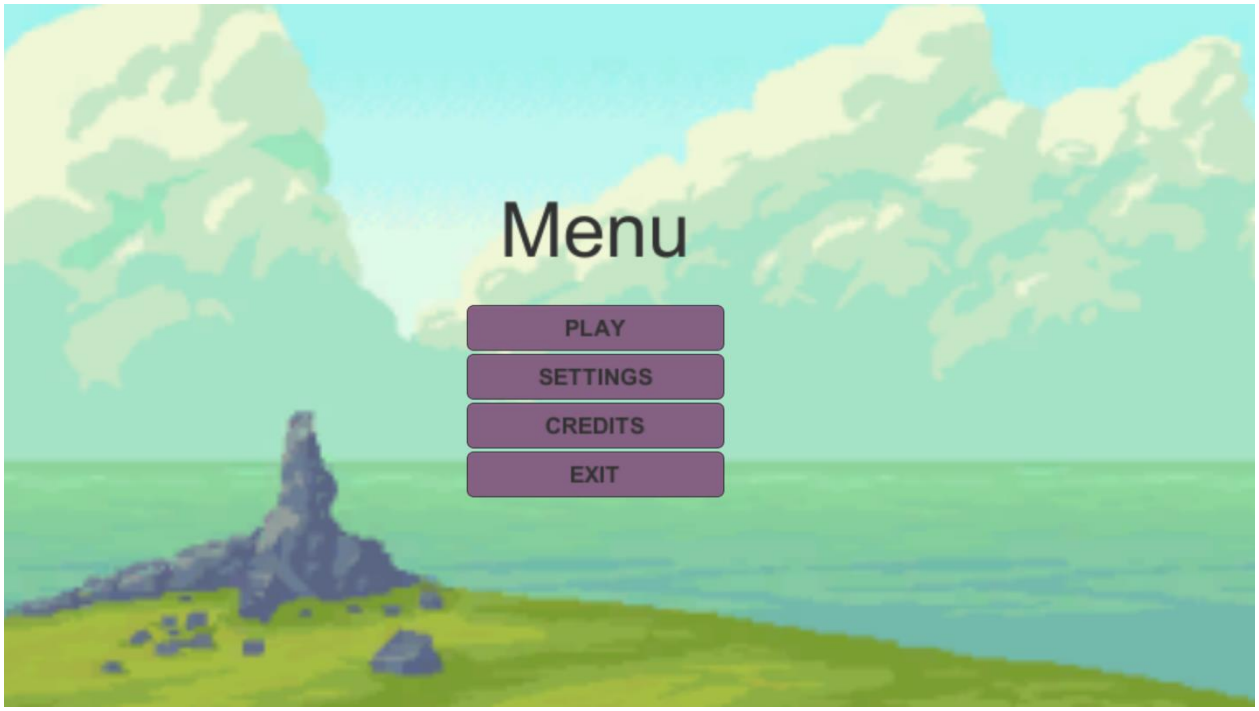


Рисунок 3.22 – Меню додатку

Управління персонажем здіснюється клавішами *W* та *D* або стрілками. Щоб стрибнути треба натиснути пробіл (рис. 3.23).

Рисунок 3.23 – Основне вікно додатку

Якщо персонаж впаде з землі у прірву, відкриється вікно програшу. Щоб спробувати пройти рівень знову, треба натиснути Restart (рис. 3.24).

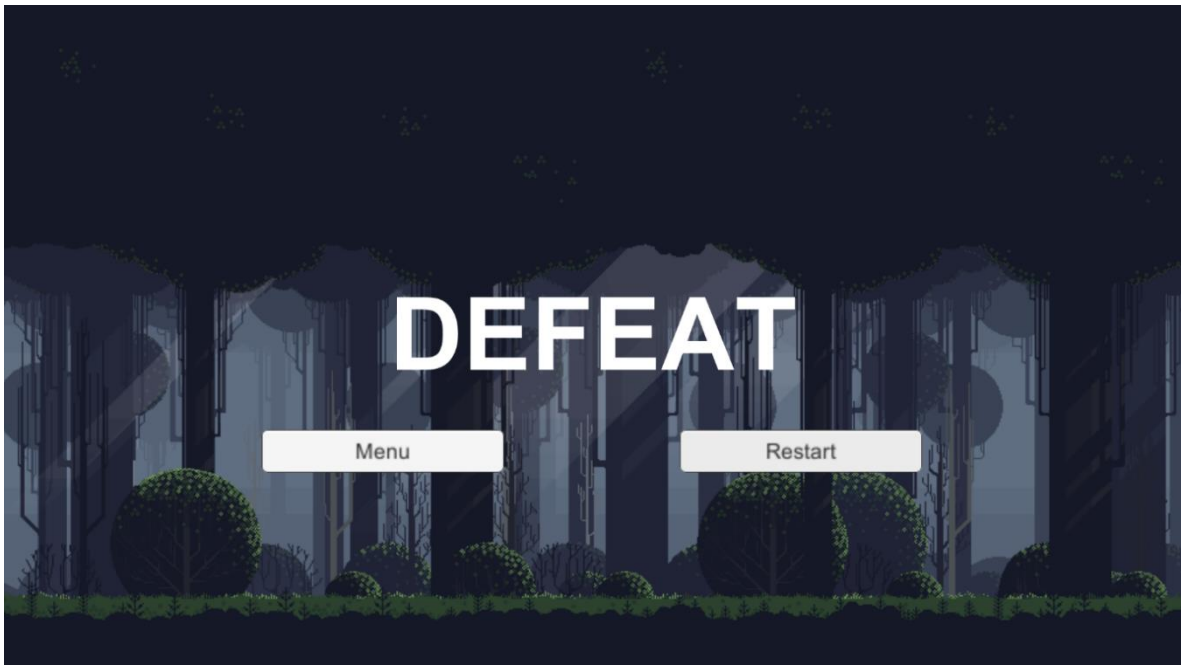


Рисунок 3.24 – Вікно програшу

Щоб завершити рівень, треба дійти до останнього блоку (рис. 3.25).

Рисунок 3.25 – Кінець рівня

Після потрапляння персонажа на останній блок, відкривається вікно виграшу. Next Level – кнопка для переходу на наступний рівень (рис. 3.26).

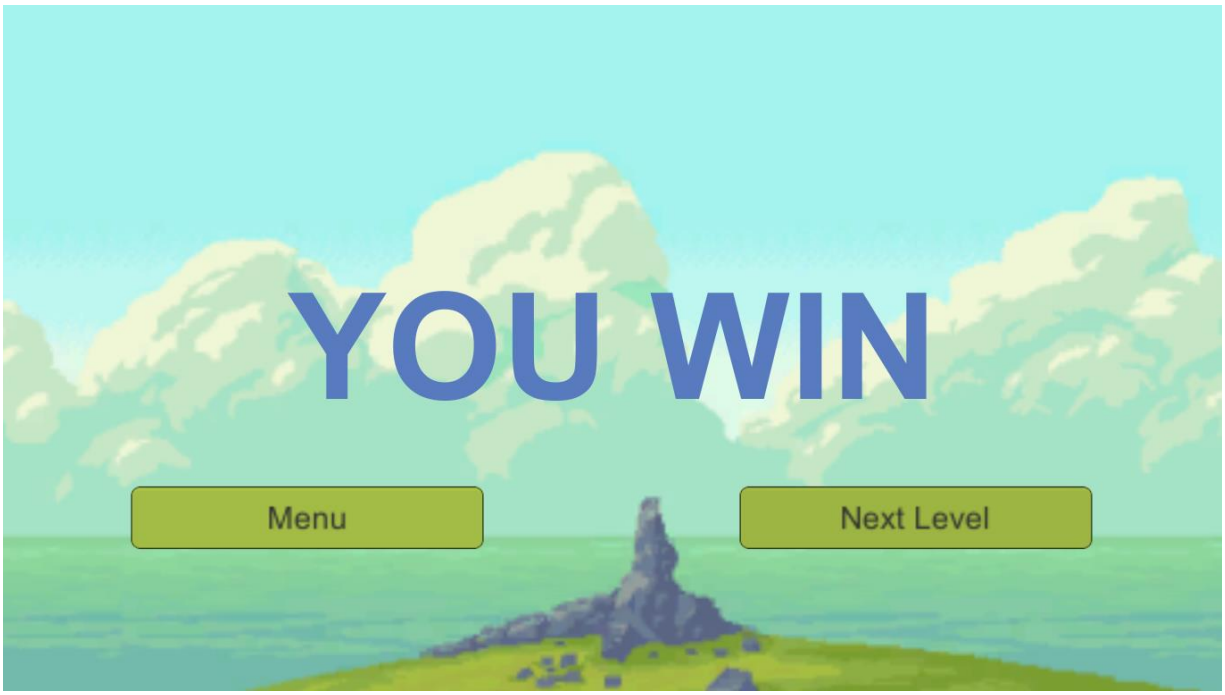


Рисунок 3.26 – Вікно виграшу

3.4 Тестування розробленого програмного засобу

Реалізована відеогра відповідає таким вимогам:

– керування інтуїтивно зрозуміле для користувача;

– користувач може відрізнити безпечні об’єкти та пастки;

– рівні є різноманітними та з прогресуючою складністю;

– рівні можливо пройти;

– під час тестування не було знайдено місць, з гравець не зможе вийти;

– кожен об’єкт має чіткі границі, які відповідають візуальним розмірам зображення об’єкту;

– на площі порожнього фону немає невидимих об’єктів, через які не може пройти персонаж.

В процесі тестування було виявлено такі недоліки:

– персонаж міг перестрибнути коллайдер кінця рівня, впасти в прірву після блоку фінішу та програти. Проблему усунуто шляхом додавання на кінці блоку фінішу вузького високого вертикального коллайдера, через який персонаж не може пройти та через який неможливо перестрибнути;

– прогресуюча складність рівнів та збільшення їх розмірів подовжують час проходження. Це заважатиме користувачам грати та може призвести до безкінечного збільшення часу гри. Мають бути знайдені шляхи покращення схеми будування рівня з метою обмеження максимально допустимого часу проходження рівня.

3.5 Результати дослідження методу автоматичної генерації рівнів відеогри-платформера

У створеному програмному засобі спосіб розміщення бонусів на рівні схожий на алгоритм, розглянутий у пункті 2.2.1. У ньому для будь-якої порожньої поверхні землі на карті ймовірність існування скарбу в цьому просторі прямо пропорційна кількості твердих поверхонь, що прилягають до простору. В створеному додатку цей алгоритм доповнений додатковими умовами, такими як наявність навколо пасток та ворогів. Це робить даний спосіб розміщення бонусів більш логічним з точки зору користувача та урізноманітнює ігровий рівень.

Алгоритм розміщує персонажів-ворогів на рівні рівномірно, що дає можливість користувачу встигнути відпочити та поповнити запаси життя персонажа. Персонаж має можливість двічі померти під час проходження одного рівня. Це припущення було введено для полегшення гри.

Розроблена відеогра розрахована та буде цікавою для гравців у віці від 7 до 40 років, які належать до групи мідкорних (Midcore) гравців. Мідкорні ігри призначені для людей, які дуже люблять грати, але у них не вистачає часу, щоб робити це часто. Вони займають проміжне місце в

градації гравців між кежуал-гравцями (мало грають і користуються простими іграми з низьким рівнем залучення) і хардкор-гравцями, які є самими справжніми фанатами ігор, витрачають на них безліч часу і грошей. З точки зору ринку ігор мідкор-гравці – це основний середній споживчий сегмент. До них відносяться школярі, які грають тільки на перервах між уроками, люди середнього віку, які заходять пограти в транспорті дорогою на навчання/роботу. Великий сегмент платоспроможних гравців мідкорних ігор – це домогосподарки з країн з високим рівнем життя.

3.6 Перспективи подальшої роботи

Проблема безкінечності рівнів у створеній відеогрі може бути вирішена шляхом додавання різних локацій. Для цього потрібно встановити граничну кількість рівнів для однієї локації, наприклад 20. Через 20 рівнів зовнішній вигляд гри буде змінюватися, а складність рівнів обнулятиметься.

Алгоритми гри при цьому способі залишаються незмінними, але з бази даних завантажуються зображення усіх елементів рівня, відповідних до дизайну нової локації. Наприклад, якщо першим ігровим всесвітом у створеному додатку був «Ліс», наступними можуть бути «Пустеля», «Марс», «Джунглі», «Велике місто», «Місяць», «Океан» тощо.

Постійна зміна зовнішнього вигляду відеогри викликатиме інтерес, стимулюватиме користувача частіше заходити у гру та проводити у ній більше часу.

Додаток можна використовувати для розвивання уваги та швидкості реакції користувача, для навчання аналізу різних поверхонь та об’єктів, а також методам створювання стратегій.

ВИСНОВКИ

У рамках дипломної роботи був розроблений і реалізований метод розміщення блоків, з яких складається рівень, та алгоритм на основі цього методу. Розроблений алгоритм дає можливість автоматично створювати різноманітні рівні у іграх без витрати додаткових ресурсів. Розроблена версія додатку була покращена за рахунок додавання в нього різноманітних блоків, пасток, неігрових персонажів, що ускладнюють рух за ігровою площиною.

Результати даного дослідження апробовано на 24-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» [41] та на IV Міжнародній науково-практичній конференції «Integration of scientific bases into practice» (Стокгольм, Швеція) [42]. Результати роботи можна використовувати у навчанні людей, які працюють з технікою. Онлайн-навчання сьогодні – можливість продуктивного професійного зростання і цікавого розвитку, яке замінює великі дорогі механічні установки. На даний момент гейміфікація вже використовується у навчанні пілотів, водіїв, гонщиків, лікарів, поліцейських, тощо. За допомогою автоматичного створення рівнів у іграх, які навчають спеціалістів у різних сферах, можна занурити людину в непередбачувані умови, що постійно та непрогнозовано змінюються.

Термін «гейміфікація» є набором заходів для створення у людини звикання до певного продукту або послуги за рахунок використання принципів ігровий психології. Інтерактивна освіта дозволяє віртуально створити умови, які неможливо відтворити на Землі. Це дає можливість, наприклад, навчати космонавтів. Під час такого навчання вони можуть звикнути до умов космосу, щоб потім бути готовими до надзвичайних подій: не тільки знати теоретичну частину, а й набути практичних навичок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Game development software engineering process life cycle: a systematic review. Journal of Software Engineering Research and Development, 4(1), 6.

2. Творошенко, И. С. (2004). Структура и функции интеллектуальных средств принятия решений в сложных системах. *Искусственный интеллект*, (4), 462-470.

3. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). IMAGE CLASSIFICATION METHOD MODIFICATION BASED ON MODEL OF LOGIC PROCESSING OF BIT DESCRIPTION WEIGHTS VECTOR. Telecommunications and Radio Engineering, 79(1).

4. Bethke, E. (2003). Game development and production. Wordware Publishing, Inc..

5. Творошенко, И. С. (2010). Анализ процессов принятия решений в интеллектуальных системах. Системи обробки інформації, (2), 248-253.

6. Кучеренко, Е. И., & Творошенко, И. С. (2010). Прикладные аспекты моделирования нечетких процессов в сложных системах. Збірник наукових праць Харківського університету Повітряних сил, (1), 127-131.

7. Pascarella, L., Palomba, F., Di Penta, M., & Bacchelli, A. (2018, May). How is video game development different from software development in open source?. In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR) (pp. 392-402). IEEE.

8. Кучеренко, Є. І., & Творошенко, І. С. (2011). Оперативне оцінювання простору станів складних розподілених об’єктів з використанням нечіткої інтервальної логіки. Штучний інтелект.

9. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Vlasenko, N. V. (2020). USING FUZZY CLUSTERING IN STRUCTURAL METHODS OF IMAGE CLASSIFICATION. Telecommunications and Radio Engineering, 79(9).

10. Кучеренко, Є. І., Кучеренко, В. Є., Глушенкова, І. С., & Творошенко, І. С. (2012). Методи, моделі та інформаційні технології оцінювання станів складних об’єктів: монографія.

11. Freedman, E. (2018). Engineering Queerness in the Game Development Pipeline. Game Studies, 18(3).

12. Кучеренко, Е. И., & Творошенко, И. С. (2003). Процессы принятия решений в сложных системах на основе нечетких интервальных представлений. Вісник Національного технічного університету" ХПІ". Тематичний випуск: Системний аналіз, управління та інформаційні технології.-Х.: НТУ" ХПІ, 1(7), 79-86.

13. Фленов, М. Е. (2012). Библия C# (2-е издание). БХВ-Петербург.

14. Ahmad, M. A., Tvoroshenko, I., Baker, J. H., & Lyashenko, V. (2019). Computational Complexity of the Accessory Function Setting Mechanism in Fuzzy Intellectual Systems.

15. Агуров, П. В. (2007). *C#. Сборник рецептов*. БХВ-Петербург.

16. Кучеренко, Є. І., Творошенко, І. С., & Анопрієнко, Т. В. (2016). Моделювання та оцінювання станів складних об’єктів із застосуванням формальної логіки. Системи обробки інформації, (2), 76-82.

17. Rosyid, H. A., Palmerlee, M., & Chen, K. (2018). Deploying learning materials to game content for serious education game development: A case study. Entertainment computing, 26, 1-9.

18. Кучеренко, Е. И. (2005). Интеллектуальные технологии в задачах принятия решений технологических комплексов на основе нечеткой интервальной логики. Восточно-Европейский журнал передовых технологий, (2), 92-96.

19. Кучеренко, Е. И., Корниловский, А. В., & Творошенко, И. С. (2010). О методах настройки функций принадлежности в нечетких системах. Системы управления, навигации и связи, (1), 13.

20. Almeida, F. (2018). Visual C# .NET: Console Applications and Windows Forms.

21. Бодянский, Е. В., Кучеренко, Е. И., & Творошенко, И. С. (2004). О синтезе нечетких алгоритмов на основе композиции фрагментов правил и моделей. АСУ и приборы автоматики, (128), 19-28.

22. Албахари, Д., & Албахари, Б. (2014). C# 5.0 Справочник. Полное описание языка.

23. Альфред, В. (2015). Ахо Компиляторы. Принципы, технологии и инструментарий/Альфред В. Ахо и др. *М.: Вильямс*, *689*.

24. Вагнер, Б. С. (2013). Эффективное программирование/Билл Вагнер. *М.: ЛОРИ, 320*.

25. Троелсен, Э., & Джепикс, Ф. (2019). *Язык программирования C# 7 и платформы. NET и. NET Core*. Litres.

26. Ahmad, M. A., Tvoroshenko, I., Baker, J. H., & Lyashenko, V. (2019). Modeling the Structure of Intellectual Means of Decision-Making Using a System-Oriented NFO Approach.

27. Tvoroshenko, I. S., & Gorokhovatsky, V. O. (2019). Intelligent classification of biophysical system states using fuzzy interval logic. Telecommunications and Radio Engineering, 78(14).

28. Deitel, P., & Deitel, H. (2016). *Visual C# how to program*. Pearson.

29. Sharp, J. (2018). *Microsoft visual C# step by step*. Microsoft Press.

30. Tvoroshenko, I., Ahmad, M. A., Mustafa, S. K., Lyashenko, V., & Alharbi, A. R. (2020). Modification of Models Intensive Development Ontologies by Fuzzy Logic.

31. Babker, A. M., Abd Elgadir, A. A., Tvoroshenko, I., & Lyashenko, V. (2019). Information technologies of the processing of the spaces of the states of a complex biophysical object in the intellectual medical system health.

32. Tvoroshenko, I. S., & Kramarenko, O. O. (2019). Software determination of the optimal route by geoinformation technologies. Radio Electronics, Computer Science, Control, (3), 131-142.

33. Tvoroshenko, I. S., & Gorokhovatskyi, V. O. (2020). EFFECTIVE TUNING OF MEMBERSHIP FUNCTION PARAMETERS IN FUZZY

SYSTEMS BASED ON MULTI-VALUED INTERVAL LOGIC. Telecommunications and Radio Engineering, 79(2).

34. Tvoroshenko, I. S. (2010). Analysis of Decision-Making Processes in Intelligent Systems, Information Processing Systems, 2.

35. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for Making Reliable Decisions on a Variety of Effective Factors using Fuzzy Logic. Int. J. Adv. Comput. Sci. Appl, 11, 43-50.

36. Gorokhovatskyi, V., & Tvoroshenko, I. (2020). Image Classification Based on the Kohonen Network and the Data Space Modification.

37. Al-Bastami, B. G., & Naser, S. S. A. (2017). Design and Development of an Intelligent Tutoring System for C# Language.

38. Yamacli, S. (2017). Beginner's Guide to C# Programming: A Practical Approach in Visual Studio.

39. Daradkeh, Y.I.; Tvoroshenko, I. Application of an Improved Formal Model of the Hybrid Development of Ontologies in Complex Information Systems. Appl. Sci. 2020, 10, 6777.

40. Kobylin, O. A., Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). THE APPLICATION OF NON-PARAMETRIC STATISTICS METHODS IN IMAGE CLASSIFIERS BASED ON STRUCTURAL DESCRIPTION COMPONENTS. Telecommunications and Radio Engineering, 79(10).

41. Алмакаєва А.Є. Метод автоматичної генерації рівнів комп’ютерної гри на основі дисперсії випадкової величини. Радіоелектроніка та молодь у XXI столітті: тези доповідей 24-го Міжнародного молодіжного форуму (Харків, 7–9 квітня 2020 р.). Харків: ХНУРЕ, 2020. Т. 7. С. 10-11.

42. Tvoroshenko I., and Almakaieva A. (2020) Application of procedural generation of game content using software algorithms, Abstracts of IV International Scientific and Practical Conference «Integration of scientific bases into practice» (October 12-16, 2020). Stockholm, Sweden, pp. 449-455. DOI: 10.46299/ISG.2020.IV.