Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.941

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Математичне забезпечення для комп’ютерної гри CatchUp”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - 30.00.00.000

Студент

ІПЗ-41 _____ /Сергій СЕРГІЙЧУК/

Науковий керівник

д.т.н, с.н.с. _____/Геннадій ПОРЄВ/

Консультант

з питань нормоконтролю

фахівець _____ /Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., проф. _____/Олексій БИЧКОВ/

Київ 2021

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Завідувач кафедри
програмних систем і технологій
_____(Олексій БИЧКОВ)
„___”_____ 2021р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Сергійчуку Сергію Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи «Математичне забезпечення для комп’ютерної гри Catch-Up»
затверджена наказом вищого навчального закладу від „__”_____20__р. №_____
2. Строк здачі студентом закінченої роботи_____
3. Вихідні дані до роботи _____
_____
_____
_____
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
_____
_____
_____
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
_____
_____

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 13 жовтня 2020р.

Керівник _____ (Геннадій ПОРЄВ)

Завдання прийняв до виконання ____ _____ (Сергій СЕРГІЙЧУК)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Збір інформації	24.10.2020	
2	Вивчення варіантів реалізації та вибір варіанту для розробки	15.01.2021	
3	Аналіз аналогів	03.02.2021	
4	Проектування основної ідеї гри	09.02.2021	
5	Розробка графічної складової гри	12.02.2021	
6	Реалізація системи управління	05.04.2021	
7	Створення дизайну ігрової карти	30.04.2021	

8	Аналіз готового продукту та оформлення дипломної роботи	31.05.2021	

Студент – бакалавр _____(Сергій СЕРГІЙЧУК)

Керівник роботи _____(Геннадій ПОРЄВ)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 68 сторінок, 7 зображень, 1 таблиця, 10 додатків, 3 джерела

Тема: Математичне забезпечення для комп’ютерної гри CatchUp

Об'єкт дослідження: Процес розробки ігрового дизайну, а також шляхи застосування математичних методів для досягнення результатів у цьому напрямку.

Мета: збільшити ефективність розробки ігрового дизайну за допомогою математичних моделей.

Завдання:

- Реалізація управління переміщенням персонажів (симуляція їзди на ковзанах)
- Реалізація механізму передачі ролі доганяючого іншому гравцю
- Вирішення проблеми затяжних матчів
- Створення ігрових карт

Предмет дослідження: Підходи до розробки дизайну. Ті алгоритми та математичні моделі, що були розроблені в процесі проведення даного дослідження.

Методи розробки:

- математичне моделювання

- статистичні опитування
- збір даних з відкритих джерел.

Результати дослідження: Систематизація різних підходів до створення ігрового дизайну. Та практичний результат їх використання (гра)

Висновок: У ході дослідження були виконані поставлені задачі та досягнені поставлені цілі.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 68 страниц, 7 изображений, 1 таблица, 10 дополнений, 3 источника

Тема: Математическое обеспечение для компьютерной игры CatchUp

Объект исследования: Процесс разработки игрового дизайна, а также пути применения математических методов для достижения результатов в этом направлении.

Цель: повысить эффективность разработки игрового дизайна с помощью математических моделей.

Задача:

- Реализация управления перемещением персонажей (симуляция езды на коньках)
- Реализация механизма передачи роли догоняющего другому игроку
- Решение проблемы затяжных матчей
- Создание игровых карт

Предмет исследования: Подходы к разработке дизайна. Те алгоритмы и математические модели, разработанные в процессе проведения данного исследования.

Методы разработки:

- математическое моделирование

- статистические опросы
- сбор данных из открытых источников.

Результаты исследования: Систематизация различных подходов к созданию игрового дизайна. И практический результат их использования (игра)

Вывод: В ходе исследования были выполнены поставленные задачи и достигнутые поставленные цели.

SUMMARY

Final qualifying bachelor's thesis: 68 pages, 7 figures, 1 table, 10 appendices, 3 sources.

Topic: Mathematical software for computer game CatchUp

Object of research: The process of developing game design, as well as ways to apply mathematical methods to achieve results in this direction.

Objective: increase the efficiency of game design development with the help of mathematical models

Task:

- Implementation of character movement control (simulation of skating)
- Implementation of the mechanism of transferring the role of catching up with another player
- Solving the problem of long matches
- Creating game maps

Subject of research: Approaches to design development. Those algorithms and mathematical models that were developed in the course of this study.

Development methods:

- mathematical modeling
- statistical surveys

- open sources researh.

Research results: Systematization of different approaches to creating game design. And the practical result of their use (game)

Conclusion: In the course of the research the set tasks were fulfilled and the set goals were achieved.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

М — метр

С — секунда

М/С — метри за секунду

Ресурс — будь який файл який використовується у програмному проєкті. До ресурсів можна віднести: зображення, відео, скріпти коду, строки, файли стилів, 3d моделі, звукові файли, тощо.

Асет — те ж саме що й ресурс.

Ігровий рушій — програмний пакет та набір бібліотек, які полегшують розробку гри.

ВСТУП

Ігровий дизайн - це частина розробки комп’ютерних та настільних ігор яка полягає в проєктуванні та створенні елементів ігрового процесу, сюжету, персонажів, завдань та інших аспектів гри.

Актуальність даного дослідження є в тому, що на даний момент майже не існує наукових робіт по цій темі у відкритому доступі.

Дане дослідження, на тему створення математичного забезпечення для комп’ютерної гри покликане вирішити проблему відсутності робіт такого типу. А також створення приклад ігрового дизайну для комп'ютерної гри в жанрі "аркада".

В роботі досліджується процес розробки ігрового дизайну, а також шляхи застосування математичних методів для досягнення результатів у цьому напрямку.

В роботі були розглянуті різні підходи до розробки дизайну. А також використані алгоритми та математичні моделі, що були розроблені в процесі проведення даного дослідження.

При виконанні роботи було застосоване математичне моделювання а також статистичні опитування та збір даних з відкритих джерел.

Результатами дослідження стала систематизація різних підходів до створення ігрового дизайну. Та практичний результат їх використання у вигляді комп’ютерної гри.

РОЗДІЛ 1
АНАЛІЗ ПРОЦЕСУ РОЗРОБКИ ДИЗАЙНУ ДЛЯ КОМП'ЮТЕРНОЇ ГРИ.

1.1 Аналіз концепції ігрового дизайну

Ігровий дизайн, це напрямок в розробці комп'ютерних ігор, який полягає в проєктуванні елементів ігрового процесу. Як-от персонажів, завдань, деталей сюдету, інтерфейсу, музики, тощо. У своїй діяльності ігровий дизайнер повинен орієнтуватись на обрану цільову аудиторію, та приймати рішення згідно з досвідом інших розробників, які вже проєктували ігри для цієї категорії гравців. Наприклад, для такої категорії населення як групи друзів ігри повинні мати кооперативні елементи або елементи змагання. Також такі ігри повинні мати низький поріг входження та мати простий, привабливий візуальний дизайн.

До ігор такого типу можна віднести ігри жанру аркада. До них входять ігри, де гравець керує одним персонажем. Зазвичай, він бачить свого персонажа від третього обличчя. Ігрова сесія аркад часто розбита на матчі, які тривають від однієї хвилини до півтора годин.

1.2 Аналіз принципів розробки ігрового дизайну для ігор жанру "аркада"

Ігри жанру аркада, це При проєктуванні дизайну для ігор жанру "аркада", в загальному випадку, необхідно пройти такі етапи:

- проєктування основної ідеї гри
- проєктування візуального наповнення
- проєктування управління персонажем
- проєктування ігрової карти

1.3 Процес проєктування основної ідеї гри

Даний процес складається з таких етапів:

Робота з джерелами, дослідження вже існуючих прикладів готових ігор. На цьому етапі дизайнер переглядає ігри в жанрі, що його цікавить. А також в суміжних жанрах, якщо його ідея виходить за рамки існуючої системи жанрів. Результатом роботи на даному етапі є розуміння основних тенденцій в індустрії та натхнення, яке допомагає на наступних етапах проєктування.

Мозковий штурм. На цьому етапі дизайнер, використовуючи інформацію зібрану на попередньому етапі, розробляє декілька різних варіантів власного бачення гри. Після цього він обирає найкращий серед створених варіантів. Результатом цього етап є готова концепція гри. Після того як основа буде створена, дизайнер стає готовим переходити до проєктування деталей.

1.4 Процес проєктування візуального наповнення

Цей процес проходить через такі етапи:

Збір інформації. Аналогічно до попереднього етапу дизайнер переглядає різні концепти. Вони не обов’язково повинні відноситись до комп’ютерних ігор

того ж жанру, що й продукт що розробляється. Результатом роботи на цьому етапі є розуміння основних трендів в графічному дизайні комп'ютерних ігор.

Розробка концептів. Дизайнер розробляє велику кількість концептів для різних елементів ігрового простору: персонажів, ворогів, елементи ландшафту, текстури, декорації, елементи інтерфейсу користувача. В результаті, в дизайнера повинна бути множина різних концептів, з яких він може вибрати найкращі.

Структуризація концепту. Дизайнер старається відкинути всі не потрібні ідеї, та привести концепти до єдиного стилю. Також концепти змінюються в угоду обмежень комп'ютерної графіки. В результаті, виходить набір стилістично поєднаних концептів, які передаються художникам. Ті, в свою чергу, займаються виготовленням ресурсів для гри по наданим концептам.

1.5 Проєктування управління персонажем

У цьому процесі можна виділити такі етапи:

Дослідження існуючих систем управління персонажем. Часто, на цьому етапі дизайнер знаходить систему, яка його вдовольняє, і він приймає рішення реалізувати її у своєму проєкті. В такому разі процес закінчується вже на цьому етапі. Інакше, він може спробувати знайти цікаві механіки в реальних способах переміщення, та прийняти рішення зробити їх симуляцію.

Перенесення концепції реального переміщення в умови гри. На даному етапі дизайнер обмірковує вимоги створені обмеженнями ігрового рушія та ігрових контролерів. Та спрощує реальний спосіб пересування так, щоб

задовольнити цим вимогам. Результатом є готова концепція управління персонажем всередині гри.

Розробка управління контролером. На даному етапі дизайнер шукає спосіб зв’язати дії які може робити персонаж із діями, які може робити гравець зі своїм контролером (геймпадом, клавіатурою чи мишею). Результатом є список пар клавіша-дія.

1.6 Проєктування ігрової карти

Даний процес складається з таких етапів:

Розробка плану місцевості. Даний етап проводиться з урахуванням характери руху персонажів, цілями які вони будуть переслідувати та візуальними елементами розробленими на попередніх етапах. Спершу дизайнер створює декілька варіантів приблизної схеми місцевості. Там він описує зв’язок між елементами карти, але не їх місцеположення.

Після чого обирає найкращу схему з доступних та збільшує її деталізацію. Після цього дизайнер може ітеративно збільшувати деталізацію плану, поки не отримає схему розташування кожного елементу гри. Цей етап може не відбуватись, якщо гра використовує процедурну генерацію ігрової карти. В такому разі дизайнер займається створенням зв’язків між різними об’єктами та правилами генерації. При такому підході дуже важлива співпраця програмістів та дизайнерів.

1.7 Висновок

Результатом дослідження процесів розробки дизайну комп’ютерних ігор, а конкретно ігор в жанрі аркада, є детальний опис всіх етапів через які проходить дизайнер або команда дизайнерів при проєктуванні ігрового продукту.

Таким чином розробка ігрового дизайну проходить через такі етапи:

- проєктування основної ідеї гри
- проєктування візуального наповнення
- проєктування управління персонажем
- проєктування ігрової карти

РОЗДІЛ 2
ПІДХОДИ ДО ПРОЄКТУВАННЯ ІГРОВОГО ДИЗАЙНУ

2.1 Обґрунтування вибору теми дослідження

Тема даного дослідження була обрана у зв’язку з тим, що не зважаючи на те, що ігрова індустрія – це одна з найприбутковіших галузей як мистецтва так і програмної інженерії, у відкритому доступі існує небагато наукових досліджень, що систематизують підхід до проєктування ігрового дизайну.

2.2 Підходи до проєктування візуального дизайну гри

При проєктуванні візуального дизайну можна розділити на два етапи: розробка загальної стилістики гри та розробка окремих ігрових ресурсів.

Для виконання першого етапу існують два підходи:

Копіювання реальних тематик, до цього можна віднести створення дизайну в стилі певної реальної культури (японська стилістика, середньовічна стилістика, давньогрецька стилістика). Або стилістику, створену іншими дизайнерами, які в свою чергу орієнтувались вже на готові елементи або на інших дизайнерів. До таких можна віднести стилістку ретро ігор, Sci-fi стилістику і так далі.

Поєднання різних елементів для створення нової стилістичної категорії. Цей підхід важчий для реалізації ніж попередній, проте він підвищує оригінальність продукту і, в свою чергу, зацікавленість користувачів.

А при проєктуванні конкретних ресурсів, які повинні відповідати стилістиці гри, можна скористатись трьома підходами:

Ручний – кожен ігровий ресурс проєктується вручну. Такий підхід потребує багато часу, тому кількість ресурсів часто знижується, а відповідно і різноманітність об’єктів у грі.

Автоматичний – об’єкти генеруються в процесі гри. При такому підході розробник займається не продумуванням дизайну конкретного ігрового предмета, а продумуванням дизайну цілої категорії предметів та дизайном їх генератора. Наприклад, якщо у грі є мечі, то при даному підході дизайнер не проєктує дизайн кожного окремого меча, а описує спільні характеристики всіх мечів а також межі цих характеристик, в яких меч буде коректно вписуватись в стилістику гри.

Комбінований – підхід, при якому генерація об’єктів використовується для полегшення роботи художника. Дизайнер обирає серед багатьох згенерований об’єктів найкращі. Та редагує їх при необхідності. До фінальної гри потрапляє сталий список ресурсів, який не змінюється в процесі гри.

2.3 Підходи до проєктування управління персонажем

При проєктуванні управління персонажем в аркадах можна використати такі підходи:

Повне копіювання дизайну системи управління з іншого ігрового продукту. Існують популярні види дизайну системи управління персонажами, такі системи є об’єктивно хорошими рішеннями. Вони являються стандартом в

індустрії і виконують свою утилітарну функцію. Їх використання не вважається плагіатом і не зустрічає негативу з боку користувачів.

Створення унікального дизайну управління персонажем. При такому підході дизайнер створює принципово ному систему управління. Її суть може бути як у новому підході до використання існуючих ігрових контролерів (клавіатури, комп'ютерні миші), так і потребувати принципово нових контролерів, які гравець повинен купувати разом із грою.

Такий підхід є ризикованим, так як в цьому разі гравці не знайомі з управлінням їм доведеться потратити певний час на його освоєння перш ніж вони зможуть повноцінно грати в гру. Також, при недостатніх дослідженнях запитів аудиторії може виявитись, що користувачам не подобається дизайн управління або він їм не зручний. В такому разі кількість гравців може упасти.

Створення дизайну управління на основі реальних систем. При такому підході розробник спостерігає за способами пересування різних реальних об'єктів (транспорту, тварин, людей) та намагається накласти особливості їх на ігрові реалії.

Часто, такий спосіб використовує загальновідомий підхід до прийому даних введених користувачем (положення комп'ютерної миші, клавіші WASD на клавіатурі) але по-новому підходить до їх обробки (політ або плавання замість ходьби, прискорення замість підпригування тощо).

2.4 Підходи до проєктування ігрового поля

Підходи до створення карти ігрового світу можна розділити на різні категорії по різним аспектам:

- За підходом до використання ресурсів:
    - Монолітний підхід. Розробка карти як єдиного об’єкта. Ігрове поле розробляється як єдиний об’єкт величезного розміру. Цей підхід є найменш ефективним і потребує багато зусиль. Але при правильному виконані цей підхід дає унікальні результати які підвищують культурну цінність фінального продукту.
    - Модульний підхід до розробки. В цьому випадку об’єкти, що розміщуються на карті ігрового світу розробляються окремо. Паралельно до цього процесу, створюється пуста заготовка ігрового ландшафту. Потім дизайнер розставляє попередньо створені об’єкти на ігровій карті. У великих командах за створення об’єктів та їх розстановку відповідають різні дизайнери. Даний підхід дозволяє розділити роботу між членами команди.
- За підходом до використання редакторів карт:
    - Без використання редакторів карт. Карта створюється в неспеціалізованому графічному редакторі. Можуть також використовуватись оцифровані об’єкти з реального світу чи відскановані малюнки намальовані за допомогою традиційних

інструментів. Це єдиний варіант що може бути використаний при розробці карти як єдиного об’єкта.

- З використанням спеціалізованих редакторів карт. В такому випадку розробникам необхідно спочатку створити власне редактор карт. І лише після цього дизайнери зможуть використати його щоб створити там карти ігрового світу. Такий підхід використовується разом з модульним підходом до проєктування ігрових об’єктів.

  Так як створення окремої програми потребує значних зусиль, цей підхід найкращим чином окупається якщо гра містить багато різних карт.

  Окрім того, редактор можна продавати разом з грою, тоді користувачі зможуть створювати власні карти що відкриває додатковий спосіб взаємодії з грою та збільшує час упродовж якого гра залишається актуальною.

- За підходом до використання процедурної генерації:
  - Ручне проєктування. Повністю без використання автоматизації процесу. Цей підхід може використовуватись як при модульному підході до проєктування так і при монолітному. Як з використанням спеціалізованого редактора карт, так і без нього. Цей підхід дозволяє отримати кращий результат при більших зусиллях. Проте, що більший розмір карти і більше на ній об’єктів то важчою стає задача по її проєктуванні.

- Процедурна генерація карт. Карти генеруються в процесі гри. Для використання цього підходу необхідно попередньо створити генератор карт. Плюсом даного підходу є те, що гру можна запускати безліч разів і завжди отримувати новий результат, це значно збільшує кількість контенту, який отримає гравець від гри. Головним мінусом є те, що при поганому дизайні генератора карта часто повторюється і у гравця виникає відчуття її штучності.
- Часткова генерація. Важливі місця карти створюються вручну, а проміжки між ними заповнюються процедурною генерацією. Особливо великий виграш можна отримати при використанні цього методу в іграх з великим відкритим світом по якому гравець може вільно пересуватись.

2.5 Висновок

У результаті дослідження різних підходів до проєктування ігрового дизайну вони були категоризовані. Нижче наведена класифікація підходів до розробки:

- Розробка візуального дизайну
  - Копіювання реальної стилістики
  - Створення нового стилю поєднуючи різні елементи дизайну
- Розробка ігрових ресурсів
  - Ручний підхід

- Повна процедурна генерація
- Попередня генерація з подальшою доробкою

- Проєктування системи управління
  - Копіювання існуючих ігрових систем управління персонажем
  - Створення унікальних систем управління
  - Розробка на основі системи руху реальних об’єктів
- Проєктування карти ігрового світу
  - Використання ресурсів
    - Монолітний підхід
    - Модульний підхід
  - Використання спеціалізованого редактора карт
    - З використанням редактора карт
    - Без використання редактора карт
  - Використання процедурної генерації
    - Ручне проєктування
    - Процедурна генерація
    - Гібридний підхід

РОЗДІЛ 3
РОЗРОБКА ДИЗАЙНУ ДЛЯ ГРИ CATCHUP

2.1 Вступ

CatchUp – це мульти-платформна комп’ютерна онлайн гра в жанрі аркада. Суть гри спроєктована за мотивами гри “World Chase Tag”. В даній імплементації правила гри дещо змінені.

У грі беруть участь два чи більше гравців. Вони керують персонажами які рухаються по ігровому полю. Один з гравців має роль того, хто доганяє. Його завдання – наздогнати будь-кого з інших гравців та доторкнутись до нього. Завдання інших гравців відповідно – не дати йому зробити цього. Коли гравець, що доганяє торкається до іншого гравця він передає йому свою роль. Якщо він не зможе зробити цього за встановлений час, то він вибуває з гри. А його роль передається випадковому гравцю, з тих що залишились. З кожною передачею ролі час, за який потрібно передати роль наздоганяючого зменшується на декілька секунд. Гра триває до останнього гравця що залишився. Він і оголошується переможцем ігрового матчу.

Візуальний стиль гри – гігантський айсберг, на якому знаходиться спортивний майданчик де й проходить ігровий матч. Персонажі якими керують гравці – це пінгвіни які рухаються по льоду на ковзанах.

Гра використовує полігональну 3д графіку.

3.1 Розробка основної ідеї гри

Спершу, була озвучена основна ідея проєкту: створити комп’ютерну гру для такої цільової аудиторії як групи товаришів, які люблять пограти в ігри разом. Дослідження найпопулярніших серед такої категорії геймерів ігор показало: це переважно онлайн ігри з простим дизайном та низьким порогом входу. Таким вимогам задовольняють ігри жанру аркада.

Далі, необхідно було визначити принципи ходу гри. Після перебору різних варіантів було прийняте рішення створити гру за мотивами чемпіонату World Chase Tag. Після цього були розглянуті різні пропозиції щодо реалізації механіки чемпіонату в комп’ютерній грі. Результатом роботи став опис механіки майбутньої гри.

3.2 Розробка візуальної частини

Проєктування візуального стилю проходило паралельно до проєктування ігрової механіки. Перед дизайнером стояли такі задачі:

- Розробити дизайн навколишнього середовища
- Розробити дизайн персонажів
- Розробити дизайн меню та інтерфейсу користувача

Спершу був обраний дизайн персонажів. Для цього був проведений аналіз дизайну персонажів в найпопулярніших аркадних онлайн іграх. Висновок: персонажі цих ігор мають просту, часто округлу форму та яскраві кольори

поверхні. На основі цих даних було прийняте рішення зробити персонажів гри пінгвінами. Пінгвіни мають гладку форму та забавну манеру ходьби.

Також потрібно виділити пінгвіна що має роль доганяючого для того, щоб гравці могли, з легкістю, виокремити його серед натовпу. Обране дизайнерське рішення: над головою доганяючого з’являється червоне кільце, що світиться.

При розробці дизайну середовища, в якому будуть знаходитись персонажі, дизайнери відштовхувались від уже визначеного дизайну персонажів. Після обмірковування, були запропоновані такі ідеї дизайну середовища:

- Дитячий майданчик в антарктиді
- Спортивний стадіон
- Каток на асберзі

Обраною стала остання пропозиція. Причиною вибору став той факт, що каток краще поєднується з антарктичною тематикою.

Для розробки дизайну окремих об’єктів був обраний модульний підхід. Був створений дизайн різних категорій об’єктів. Які після цього були змодельовані в 3д редакторі. До таких об’єктів відносяться: перешкоди на катку, трампліни, бордюри, огорожі, прапорці тощо.

Щодо дизайну меню, то після проведення дослідження ігрових меню з різних проєктів було прийняте рішення реалізувати його у вигляді тайлів. Екран монітора комп’ютера розділений на тайли. Кожен тайл – це велике зображення з написом поверх нього. Кожен тайл – це й кнопка, яка переносить користувача до наступної категорії меню. Між тайлами є прогалини через які видно задній фон.

Задній фон – це велике зображення з ефектом розмиття на ньому. Кожен тайл має два режими: чорно-білий та кольоровий. У звичайному своєму стані тайл чорно-білий, а при наведенні курсора на нього – стає кольоровим.

Зображення на тайлах складуються в єдину картину, що створює відчуття єдності композиції всього меню.

3.4 Розробка дизайну системи управління

У зв’язку з тим що в якості тематики гри була обрана гра на льодовому катку, дизайн системи управління персонажем повинен симулювати їзду на катку.

Для цього було проведене польове дослідження на катку. Результатом цього дослідження став аналіз принципу руху людини на ковзанах.

Ковзаняр може рухатись двома способами: ходити та їздити.

Ходьба – єдиний можливий спосіб переміщення на сухій поверхні. Коли ковзаняр ходить він пересувається з повільною швидкістю. Якщо він ходить по льоду то йому важче втримувати баланс.

Натомість їздити ковзаняр може лише на льоду. Для цього він періодично відштовхується від поверхні однією з ніг (ліва та права ноги чередуються). Коли не відштовхується, він рухається завдяки інерції, поступово втрачаючи швидкість. А коли відштовхується — навпаки, з кожним поштовхом його сила росте. Його максимальна швидкість обмежена, в першу чергу, силою його м’язів.

Для того щоб змінити напрямок руху він нахиляє корпус тіла у відповідну сторону. Тим самим зміщуючи центр ваги. Людина на ковзанах не може різко змінити напрямок свого руху чи загальмувати. Тому, вона повинна наперед розраховувати радіус свого повороту, для того щоб не зіткнутись з поверхнею.

Для гальмування ковзаняр ставить ноги під кутом до напрямку руху. Кут повинен бути не великим, інакше — ковзаняр може втратити рівновагу і упасти.

З цього можна зробити висновок про характерні особливості їзди на ковзанах:

- рух ривками
- поступовий розгін
- довгий гальмівний шлях
- великий радіус повороту
- повільна швидкість на сухій поверхні
- постійний ризик падіння

Отримавши цю інформацію дизайнер перейшов до проєктування симуляції їзди на ковзанах всередині комп’ютерної гри.

Запозичення сторонньої системи управління не підходить для виконання даної задачі, так як такі системи розраховані на симуляцію ходьби (рух з постійною швидкістю, миттєве прискорення та миттєва зупинка). Тому, було прийняте рішення створити свою систему управління персонажем з урахуванням особливостей руху на ковзанах.

Так, після аналітичної роботи була створена концепція такої системи. Її короткий опис:

Персонаж може рухатись в трьох режимах:

- Їзда на ковзанах
- Ходьба на ковзанах
- Рух в повітрі

У всіх трьох режимах за обробку руху персонажа відповідає вбудована система фізичної симуляції ігрового рушія.

В режимі їзди, коли гравець натискає на відповідну кнопку та дає команду до руху вперед, до персонажа прикладається сила раз в певний проміжок часу. Ця сила реалізує собою симуляцію поштовхів ногами ковзаняра. Його швидкість росте з кожним поштовхом проте не може перевищити деякого фіксованого значення.

До персонажа прикладаються горизонтальні сили не частіше ніж раз в певний заданий час, тому можливості гравця до управління дещо обмежені. Для того, щоб оволодіти вмінням їзди всередині гри, гравець повинен навчитись відчувати ритм, з яким відбуваються поштовхи, та знати мінімальний радіус, на який він може повертати.

Загалом, рух персонажа в цьому режимі можна описати такою моделлю:

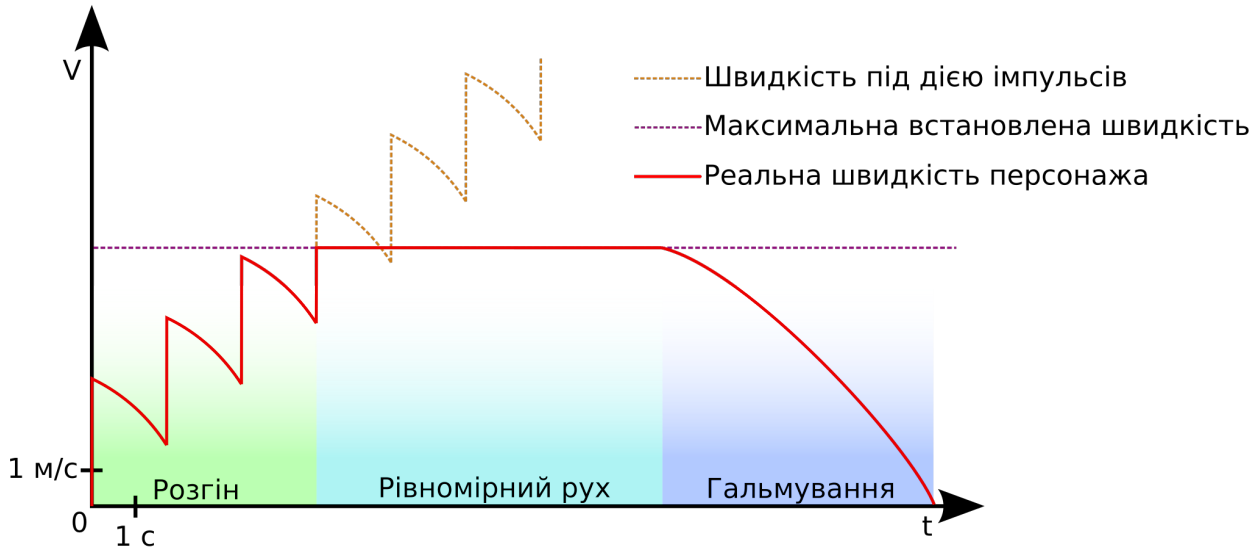


Рис. 3.1 Графік швидкості персонажа

Тут можна побачити, що персонаж в режимі їзди на ковзанах в процесі свого руху проходить через три етапи:

На етапі розгону швидкість персонажа різко збільшується з кожним ривком. А в період між ними — вона падає. Сила поштовхів та інтервал між ними відрегульовані так, що швидкість не падає до нуля і відповідно — постійно росте. Для того щоб запобігти цьому існує встановлений поріг швидкості.

Коли швидкість персонажа досягає цього значення він починає рухатись рівномірно. Тоді він знаходиться у фазі рівномірного руху. Це буде відбуватись до тих пір поки буде натиснута хоч одна клавіш руху.

А після цього, коли гравець відпустить клавіші руху — персонаж перейде до фази гальмування. І його швидкість буде падати аж до повної зупинки персонажа.

Для того, щоб гравець отримував відчуття розгону необхідно щоб максимальна швидкість була досягнута через 2-3 ривка. Для вирішення цієї проблеми була створена формула, за якою можна визначити силу та час між ривками, які дозволять отримати оптимальний результат.

$$\frac{S}{n} < a - p, p = D\,t\,f \quad (3.1)$$

Вище наведене рівняння описує ситуацію, в якій персонаж рухається ривками до досягнення максимальної швидкості.

S – максимальна швидкість руху

n – бажана кількість ривків до її досягнення

a – приріст швидкості при ривку

p - спад швидкості між ривками

D – сила тертя з поверхнею

t – необхідний час між ривками

f – частота роботи симуляції Unity. (50 Гц)

Використовуючи цю формулу дизайнер знайшов оптимальні значення сили поштовху, частоти поштовхів та максимальну можливу швидкість, якої може досягти персонаж.

Режим ходьби був доданий до гри так як він дозволяє гравцям виходити із тісних приміщень та більш точно налаштовувати рух свого персонажа.

В режимі ходьби гравець має повний контроль над персонажем. Він миттєво досягає максимальної швидкості та миттєво зупиняється, якщо гравець не тримає клавіш руху. Щоб увійти в цей режим потрібно зажати клавішу Shift. Якщо її відпустити — то персонаж вийде з цього режиму і перейде в режим їзди на ковзанах. Перемикати режими можна лише за умови, що персонаж не знаходиться у повітрі.

Управління в даному режимі працює аналогічно управлінню інших ігор жанру аркада чи шутер від першої особи.

Щоб збалансувати переваги цього режиму, було прийняте рішення зменшити швидкість персонажа персонаж в режимі ходьби.

Важливим етапом розробки системи управління є зв’язування набору дій, які може виконати персонаж з набором дій, які може зробити гравець використовуючи клавіатуру і мишу.

Таблиця 3.1 Елементи управління в різних режимах руху

Дія гравця	Дія персонажа в режимі їзди	Дія персонажа в режимі ходьби
Рух миші	Поворот	
Клавіша W	Рух уперед	
Клавіша S	Загальмувати	Рух назад
Клавіші AD	Рух в боки	
Клавіша space	Підстрибнути	
Зажати Shift	Перейти в режим ходьби	
Відпустити Shift		Перейти в режим їзди

Також персонаж може перебувати в режимі польоту. Він знаходиться в ньому якщо він не розташований на поверхні землі, декорацій чи інших персонажів.

Існує декілька випадків в яких це може статись:

- Персонаж підстрибнув. Для цього до нього прикладається сила напрямлена прямо вгору. Завдяки тому, що персонаж знаходиться всередині фізичної симуляції, на нього постійно діє сила гравітації, яка тягне його вниз.
- Персонаж наступив на активну платформу. Дизайн гри передбачає наявність платформ, які підкидують персонажа вгору з великою силою. Механізм їх роботи аналогічний принципу дії підпригування.

- Персонаж зійшов з краю певного об’єкта, на якому він знаходився. Це може статись якщо гравець дав йому команду до руху, чи коли його штовхнув інший гравець.

В такому режимі рух персонажа повністю залежить від фізичної симуляції. А ручне управління відключається. Це потрібно для того, щоб гравець не міг випадково вилетіти за межі ігрового поля у разі, якщо він знаходиться надто високо у повітрі, та на нього не діють сили тертя з поверхнею.

Перебування персонажа в даному режимі продовжується до тих пір, допоки він знаходиться в повітрі. А коли він торкається поверхні землі, декорацій чи іншого персонажа, то переходить до одного з двох режимів: ходьба чи їзда на ковзанах. Якщо гравець в момент дотику тримає клавішу Shift нажатою — то персонаж переходить в режим ходьби на ковзанах. Інакше, якщо гравець не тримає цієї клавіші натиснутою, то персонаж переходить в режим їзди на ковзанах.

Також, було проведене порівняння даної системи управління персонажем з аналогічними системами управління що вже існують в інших проєктах. Так, в грі Fall Guys використовується управління яке має лише один режим який відповідає режиму ходьби на ковзанах з нашого проєкту. Тобто:

- управління клавішами WASD.
- Миттєвий розгін
- Миттєва зупинка

- Постійна швидкість

Даний підхід, хоч і чудово підходить для аркадних ігор, проте зовсім не підходить для симуляції їзди на ковзанах.

Рис. 3.2 Гра Fall Guys

Також, була розглянута гра Sky Roller Skate Stunt Games 2021 - Roller Skating. Ця гра — це симулятор їзди на ковзанах.

Суть управління персонажем у цій грі:

- Гравець може проводити пальцем вліво та вправо щоб заставити персонажа відхилитись від курсу.
- Персонаж самостійно рухається уперед з постійною швидкістю.
- Гра не передбачає зупинок персонажа. Його зупинка через зіткнення з перешкодою означає програш.

Підхід, використаний в розглянутій вище грі не підходить для багатокористувацької гри. А також для гри, цільова платформа якої —

персональний комп’ютер. Більшість персональних комп’ютерів мають клавіатуру. І їх користувачі очікують, що гра буде повноцінно використовувати наявні можливості машини, на якій вона запущена.

Рис. 3.3 Гра Sky Roller Skate Stunt Games 2021 - Roller Skating

3.4 Розробка ігрового балансу

При розробці дизайну ігрового процесу дизайнер зіткнувся з такими проблемами:

- При передачі ролі між персонажами, гравець, персонаж якого отримав роль доганяючого може миттєво передати роль назад попередньому гравцю.

Щоб запобігти цьому було прийняте рішення вимкнути рух персонажа, який отримав роль доганяючого на певний час, що дозволить іншим гравцям відійти від нього на безпечну відстань.

Дослідним шляхом було встановлено, що оптимальний час зупинки: 0.6 секунд. Для цього гра була багаторазово протестована з різними значеннями даного параметра та інших параметрів гри, таких як:

- Розмір ігрового поля
- Тривалість матчу
- Швидкість персонажів
- Кількість гравців
- Кількість перешкод на ігровому полі

Рис. 3.4 Момент перед передачею ролі

- Також важливу роль в балансі ігрового процесу займає тривалість гри. Для того, щоб гра не тривала занадто довго було прийняте рішення: кожного разу коли гравець передає свою роль доганяючого іншому гравцю, то термін, за який він повинен це зробити зменшується на декілька секунд.

  Нижче наведена формула за якою можна обрахувати оптимальне значення даного параметра:

$$dt = \frac{t0 - t1}{n}, n = p\,k \quad (3.2)$$

  dt — Значення на яке зменшується таймер

t0 — Час таймера на початку ігрового матчу.

t — найменший час таймера, за який, в середньому, гравець може наздогнати іншого гравця на ігровому полі.

n — бажана кількість передач ролі доганяючого, яка повинна відбутись в середньостатистичному матчі.

p — кількість гравців на початку матчу

k — Коефіцієнт тривалості гри

3.6 Розробка карти ігрового поля

Для розробки карти світу не використовувались спеціалізовані редактори карт. Це зв’язано з тим, що для цього необхідно створити сам редактор, а в умовах, коли гра містить лише одну або декілька карт це рішення є не вигідним. Був обраний значно простіший спосіб: створити 3d модель карти ігрового поля в 3d редакторі загального призначення. Там же були розроблені й моделі різних декорацій, які після цього були розташовані на ігровому полі. Розставляли їх всередині редактора ігрового рушія. Таким чином подальше редагування їх положення було спрощене.

Спочатку були створені об’єкти-перешкоди. Після цього вони були розставлені на карті. При їх розстановці потрібно було враховувати радіус кола, навколо якого рухається персонаж при повороті.

Розмір об’єктів на ігровому полі та оптимальна відстань між ними була визначена згідно з цією формулою.

Також, дослідним шляхом були визначені основні принципи, яких потрібно дотримуватись при проєктування ігрового поля для конкретної гри що розробляється:

- Перешкоди, що розкидані на ігровому полі повинні мати цілком округлу форму при виді згори, або їх кути повинні бути заокруглені якщо дивитись зверху.

  Це потрібно для того, щоб гравець легше обминав ці перешкоди.

- Розмір ігрового поля повинен мати такі розміри, щоб гравець міг об’їхати його не швидше ніж за 10 секунд.

  Це потрібно щоб гравці не відчували замкненості простору на ігровому полі.

- Відстань між елементами декорацій та перешкод, розкиданих на ігровому полі, повинна бути достатньою, щоб персонаж міг пройти між ними. Потрібно враховувати не лише розміри самого персонажа, а й мінімальний радіус його повороту.

  Так, була виведена формула для обрахунку даного радіусу:

$$R=\frac{0.5\,i\,v}{\cos(0.5\,\gamma i)} \quad (3.3)$$

  R – радіус;

  i – час між ривками;

  v – швидкість гравця;

  γ - швидкість обертання персонажа навколо своєї осі.

Отримана інформація дозволяє створювати ігрові карти, які не мають вузьких місць, де персонажі часто застрягають.

Для кращої ілюстрації практичної цінності вище наведеного дослідження можна навести приклад правильного, та не правильного дизайну частини ігрового поля:

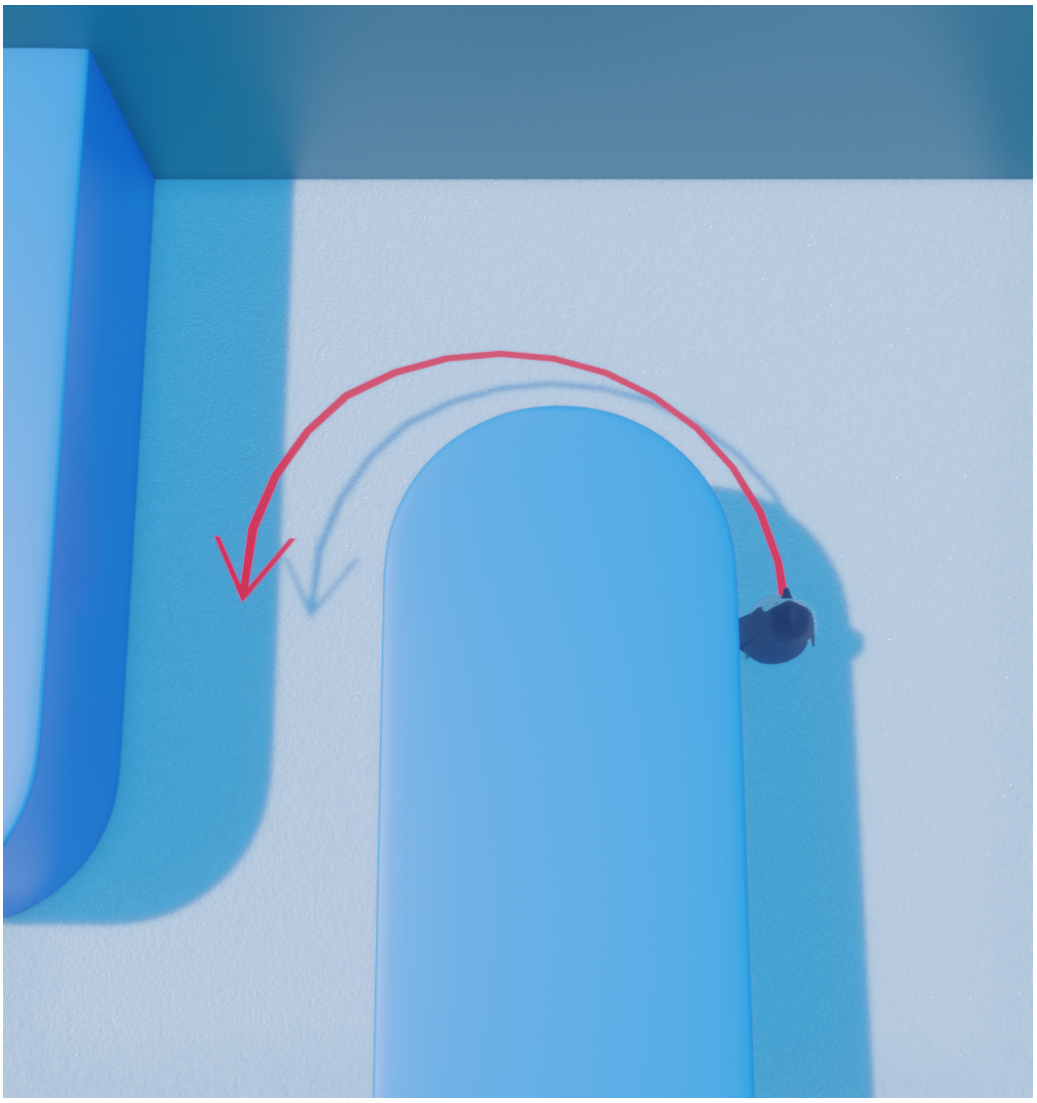


Рис. 3.5 Неправильний дизайн карти

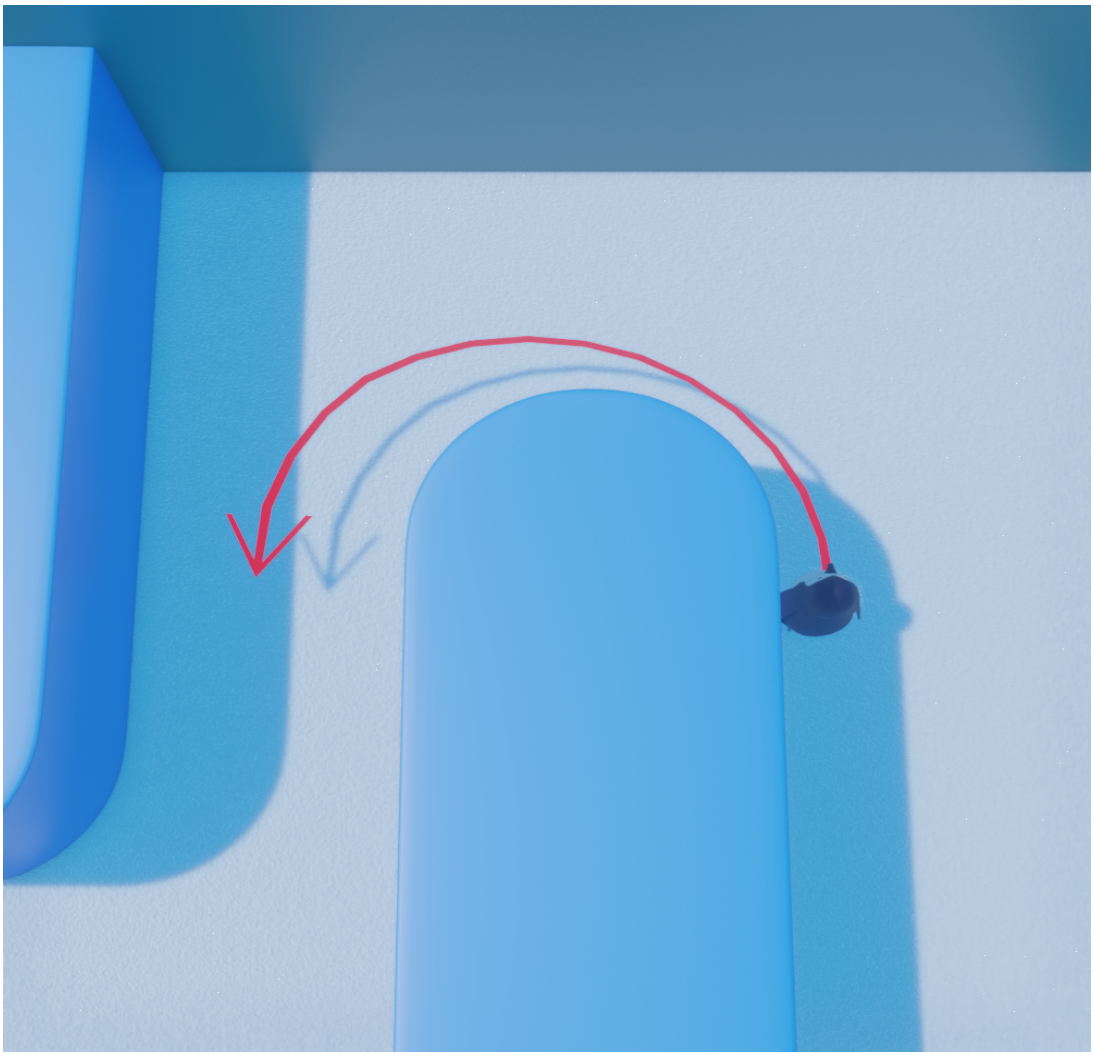


Рис. 3.6 Правильний дизайн карти

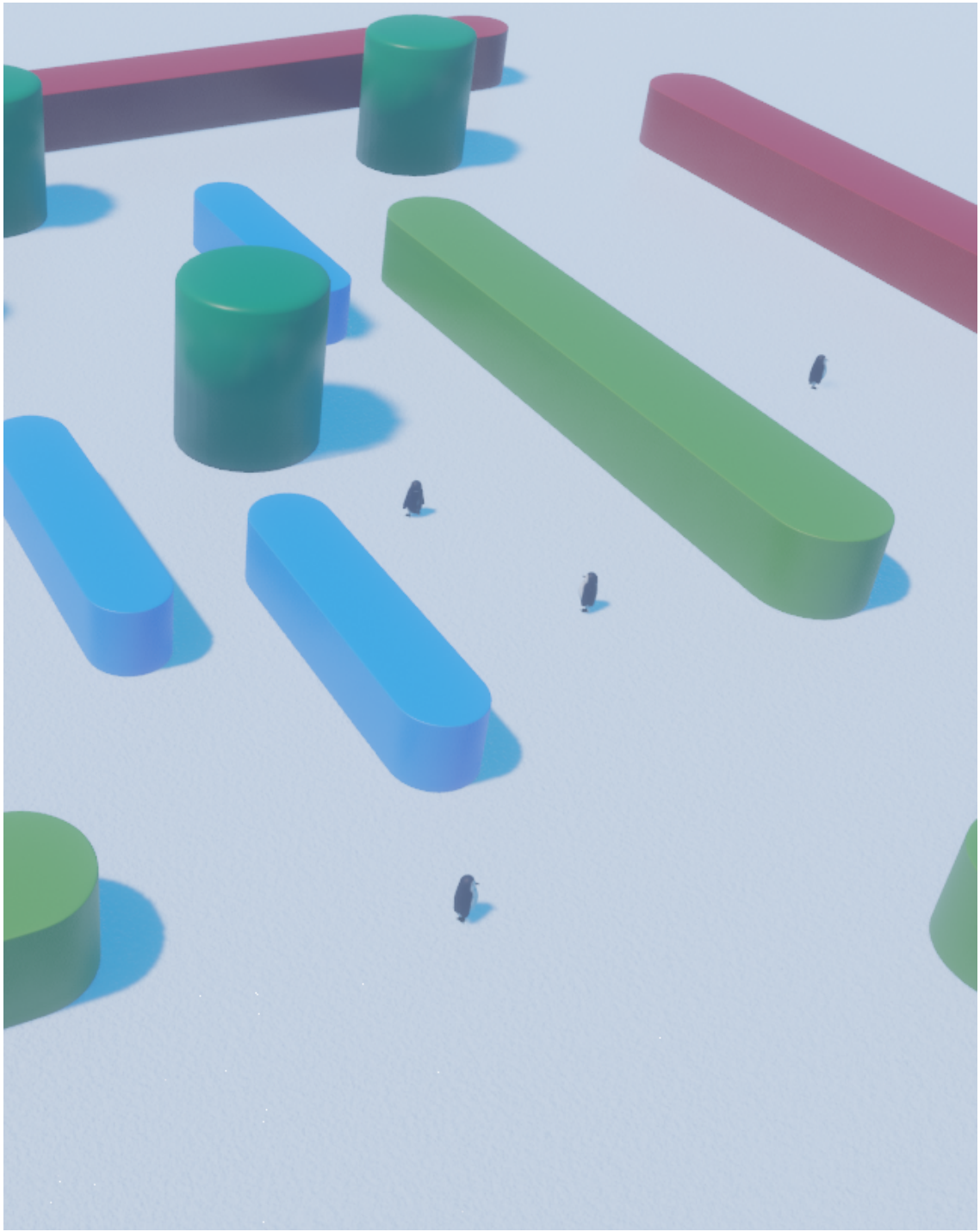


Рис. 3.7 Частина готової карти

3.7 Висновок

Результатом роботи по створенню дизайну для комп’ютерної гри Catch-Up є готовий план розробки, який може бути реалізований в програмному коді. При роботі були вирішені такі задачі:

- Проєктування системи руху персонажа
- Проєктування ігрової карти
- Проєктування глобальної стилістики гри
- Розробка дизайну конкретних об’єктів
- Розробка дизайну персонажа
- Розробка режимів руху персонажами
- Розробка моделі руху персонажа в режимі їзди на ковзанах
- Вирішення проблеми руху персонажа у повітрі
- Проєктування дизайну карти
- Вирішення проблеми розгону

Всі ці завдання були виконані а проблеми були вирішені. На основі описаних моделей та вказівок були створені реальні елементи ігрового продукту.

ВИСНОВКИ

Усі поставлені завдання даної наукової роботи була виконані а її мета досягнена. Створений детальний опис процесу через який проходить розробка дизайну для будь-якої комп’ютерної гри та конкретно ігор жанру “аркада”. А також описані приклади проблем та завдань з якими можуть стикатись ігрові дизайнери. Та наведені приклади вирішення даних проблем.

Також були виконані такі завдання як:

- Створення симуляції їзди на ковзанах. Створення математичної моделі такого виду руху.
- Створення ігрового дизайну. Опис таких елементів ігрового процесу як передача ролі від одного персонажа до іншого а також визначення оптимальної тривалості ігрового матчу і знаходження оптимальних параметрів ігрового процесу, які дозволяють досягти бажаної тривалості ігрового матчу.
- Опис основних принципів за якими повинна створюватись карта ігрового поля. Та, користуючись цими принципами створений приклад поля гри.

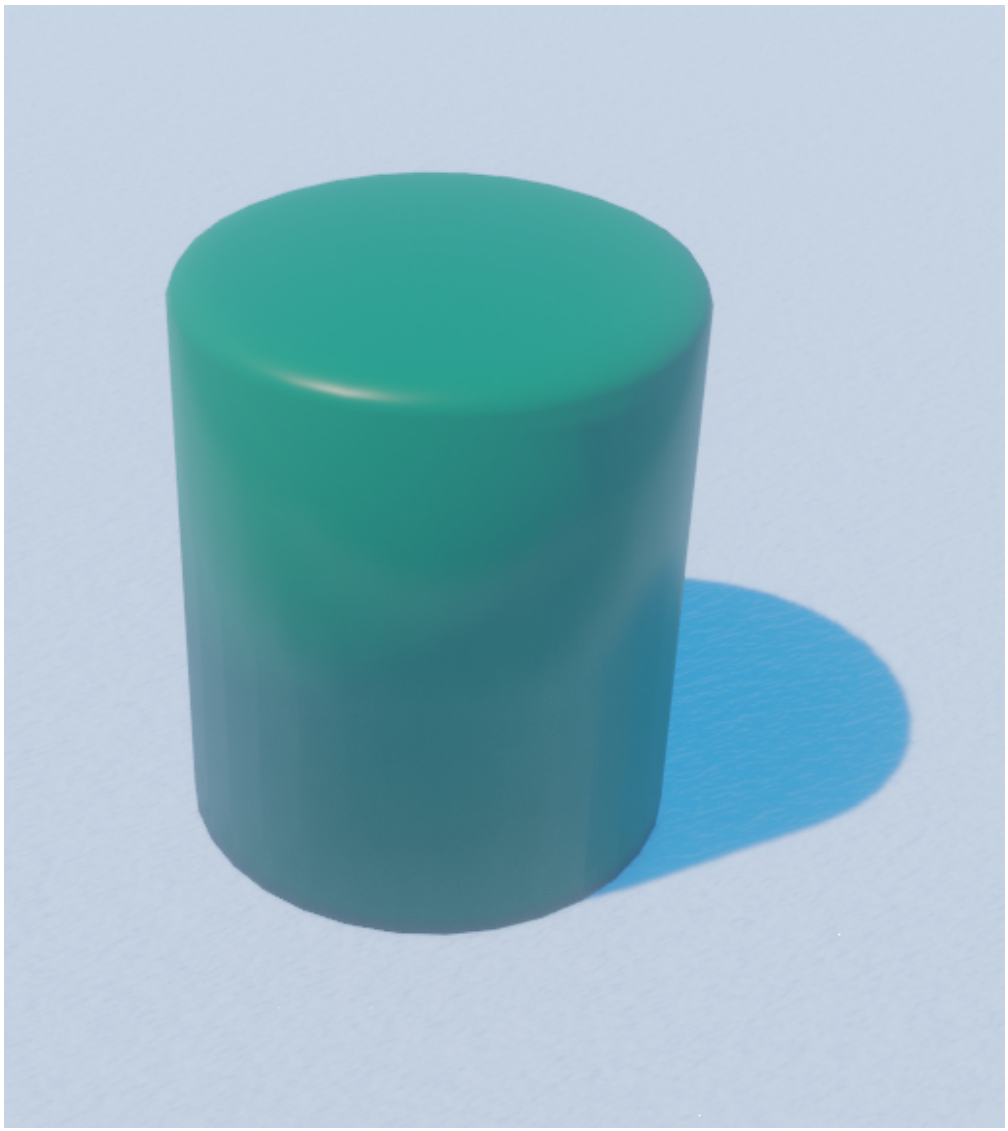
На основі інформації отриманої в даній роботі створена комп’ютерна гра CatchUp. Гра готова вийти на ринок. Елементи механік та дизайну цієї гри можуть бути використані іншими ігровими дизайнерами всередині їх власних проєктів.

ЛІТЕРАТУРА

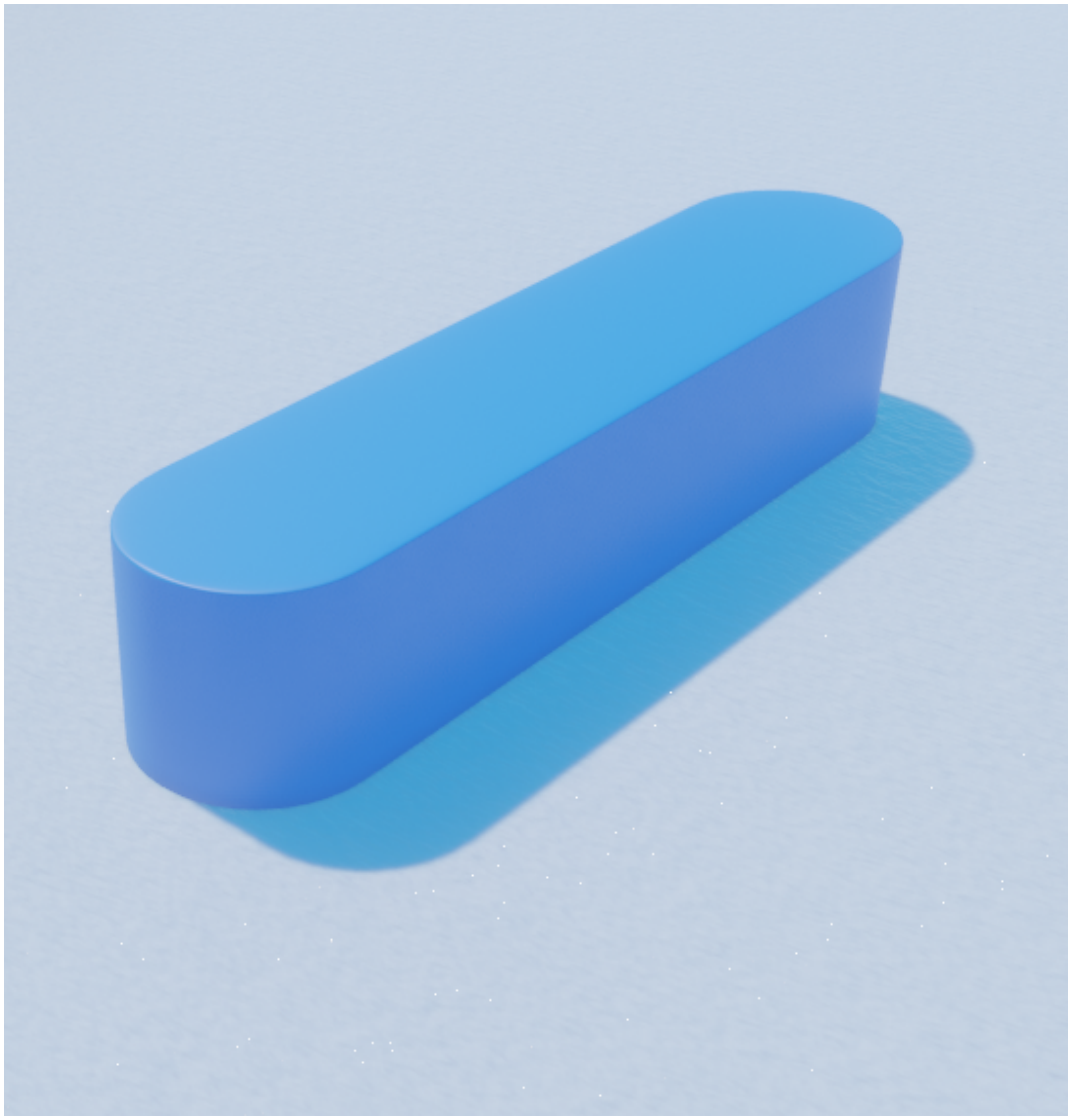
1. Jason Schreier - Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made
2. Gregory Trefry - Casual Game Design Designing Play for the Gamer in ALL of Us
3. Baur, Wolfgang. Complete Kobold Guide to Game Design. Open Design LLC 2012. ISBN 978-1936781065
4. Burgun, Keith. Game Design Theory: A New Philosophy for Understanding Games. Publisher: A K Peters/CRC Press 2012. ISBN 978-1466554207

ДОДАТКИ

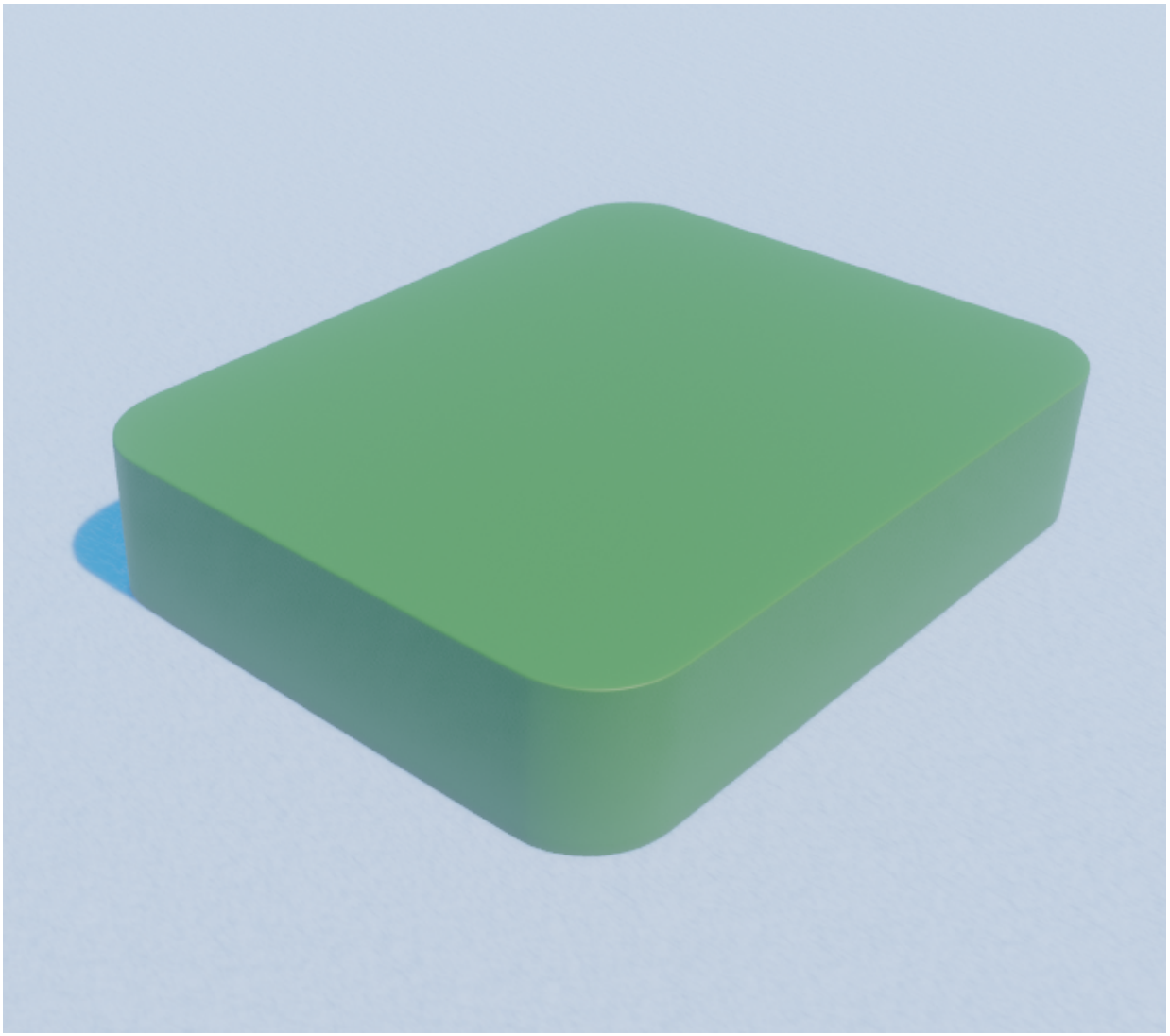
Додаток А



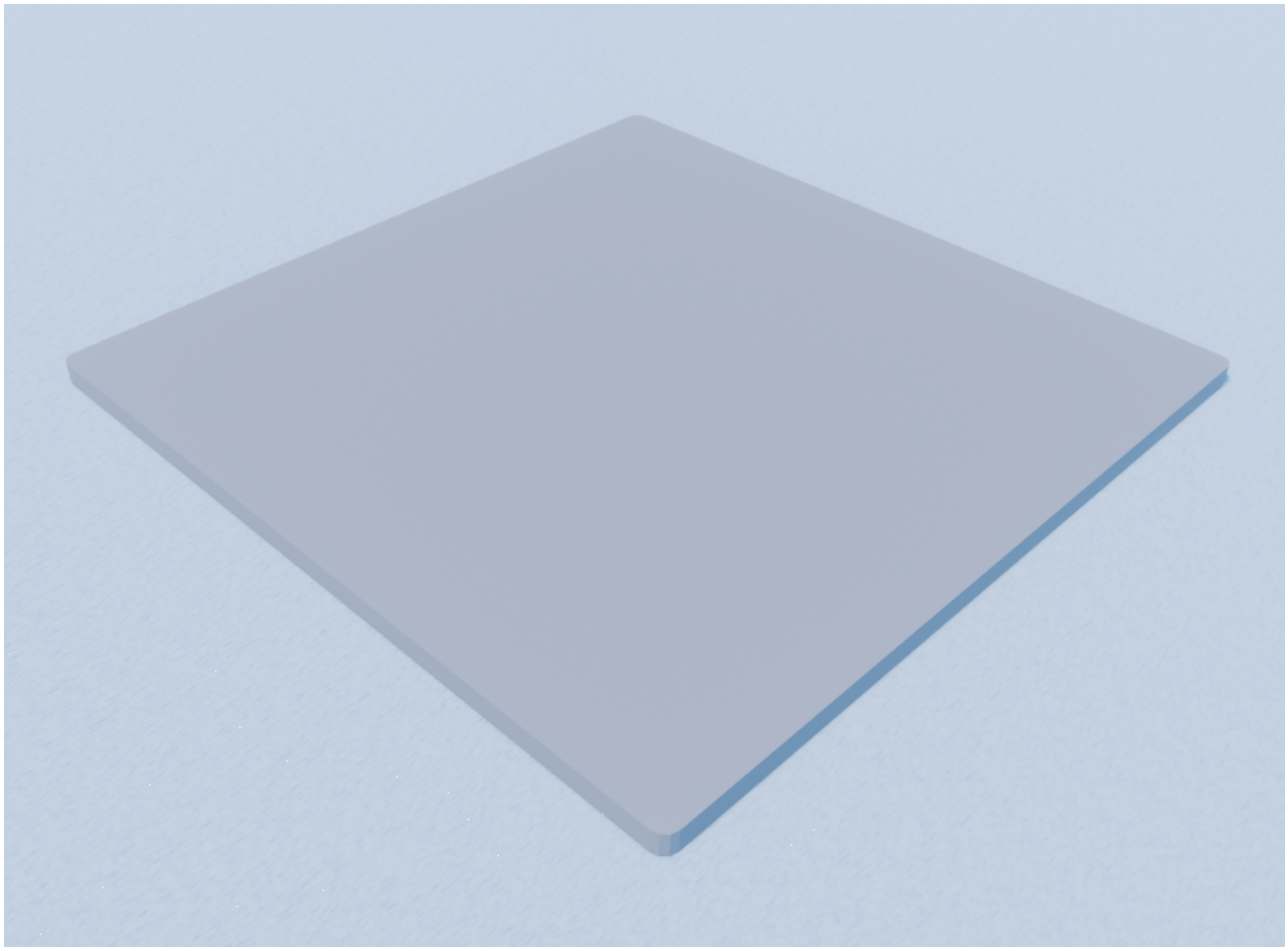
Додаток Б



Додаток В



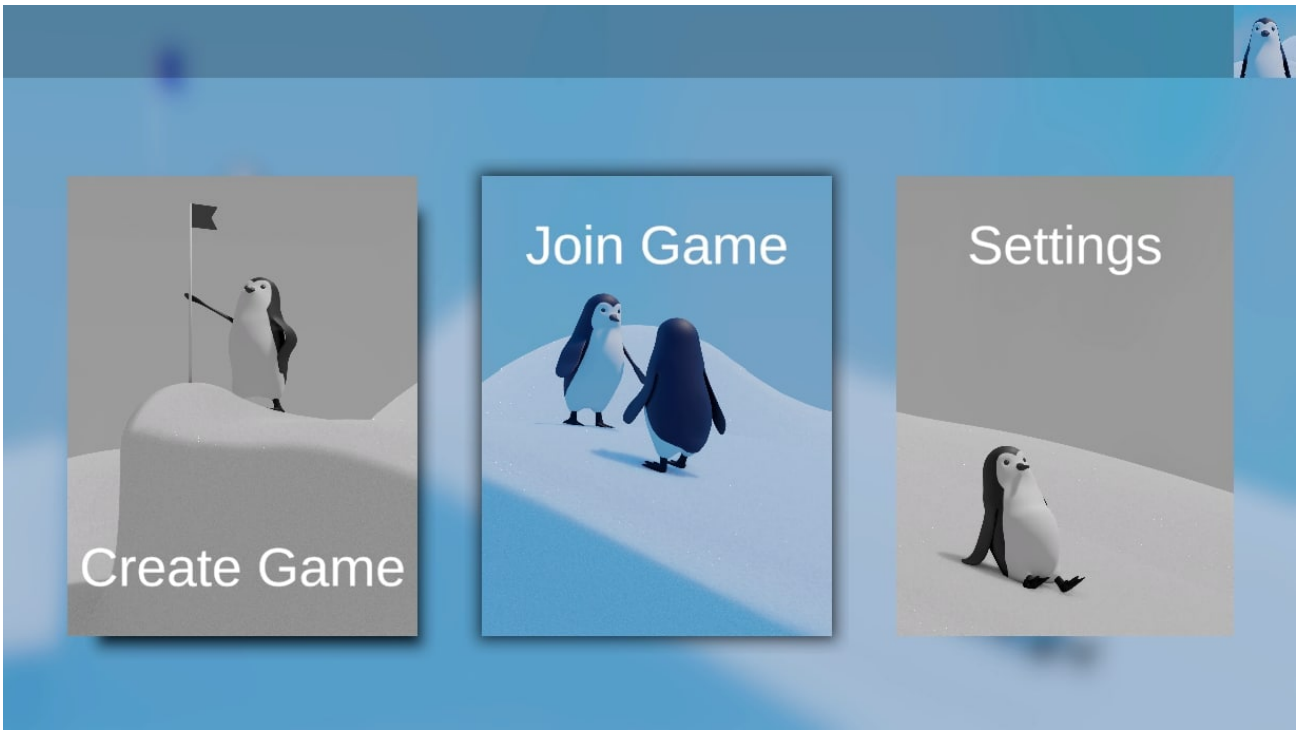
Додаток Г



Додаток Д

Додаток Е

Додаток Ж



Додаток 3

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

[System.Obsolete]
public class PlayerController : NetworkBehaviour
{
    [SerializeField] private float speed = 5.0f;
    [SerializeField] private float speedMax = 10.0f;
    [SerializeField] private float jumpForce =
200.0f;
    [SerializeField] private LayerMask layerMask;
    [SerializeField] private Transform target;

    [SerializeField] private float waitTime = 0.5f;

    private bool isMoveTime = true;



    private Rigidbody rb;
```

```
public bool isGrounded;
private bool isJumpKeyDown;

private void Awake()
{
    Cursor.lockState = CursorLockMode.Locked;
    rb = gameObject.GetComponent<Rigidbody>();
}

private void Start()
{
    if (isLocalPlayer)
    {
        target.gameObject.SetActive(true);
    }
}

private void Update()
{
    if (!isLocalPlayer) return;

    MouseControll();
```

```
        if (Input.GetButtonDown("Jump"))
            isJumpKeyDown = true;
    }

    private void FixedUpdate()
    {
        if (!isLocalPlayer) return;

        isGrounded =
Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.down), 0.1f,
layerMask);
        if (isGrounded)
            Move();

        Jump();
    }
    private void Move()
    {

        if (rb.velocity.magnitude ≥ speedMax)
            return;
```

```
            Vector3 direction = new
Vector3(Input.GetAxis("Horizontal"), 0,
Input.GetAxis("Vertical")).normalized;
            Debug.Log(direction);


            if (direction.magnitude == 1 && isMoveTime)
            {
                rb.AddRelativeForce(direction * speed,
ForceMode.Impulse);
                isMoveTime = false;


                StartCoroutine(WaitMoveTime());
            }


            #region
            /*
    Vector3 b = new Vector3(Input.GetAxis("Horizontal"),
0, Input.GetAxis("Vertical"));
    if (b.magnitude > 1)
        b = b.normalized;


    if(Input.GetAxis("Vertical") < 0)
```

```
            b.x /=2;
        b = transform.TransformDirection(b) * speed;

        var velocityChange = b - rb.velocity;
        velocityChange.y /= 2;

        rb.AddForce(b*speed, ForceMode.VelocityChange);
        */

                /* Move Velocity
                b = transform.TransformDirection(b) * speed;
                rb.velocity = b + Vector3.up *
rb.velocity.y;
                */
                #endregion
            }
            private IEnumerator WaitMoveTime()
            {
                yield return new WaitForSeconds(waitTime);
                isMoveTime = true;
                yield return null;
            }
```

```
private void Jump()
{
    if (isJumpKeyDown)
    {
        if (isGrounded)
        {
            rb.AddForce(jumpForce * Vector3.up);
        }
        isJumpKeyDown = false;
    }
}


private void MouseControll()
{
    float mouseX = Input.GetAxis("Mouse X");
    float mouseY = Input.GetAxis("Mouse Y");

    const int max = 80;
    const int min = -40;
    var angle = target.rotation.eulerAngles.x;
```

```
            if (angle > 180)
                angle -= 360;

            var tooLow = angle < min && mouseY < 0;
            var tooHigh = angle > max && mouseY > 0;
            var normal = angle ⩾ min && angle ⩽ max;

            transform.Rotate(Vector3.up * mouseX);
            if (tooLow || tooHigh || normal)
                target.Rotate(-Vector3.right * mouseY);
        }

    }
```

Додаток И

```
    using UnityEngine;
    using UnityEngine.Networking;
    using Random = System.Random;

    public class AnimationSubcontroller :
NetworkBehaviour
    {
```

```
        private int isFlying =
Animator.StringToHash("IsFlying");
        private int isMoving =
Animator.StringToHash("IsMoving");
        private int isJumping =
Animator.StringToHash("IsJumping");

        private int idle1 =
Animator.StringToHash("idle1");
        private int idle2 =
Animator.StringToHash("idle2");

        private int speedX =
Animator.StringToHash("SpeedX");
        private int speedY =
Animator.StringToHash("SpeedY");



        private float restMode =
Animator.StringToHash("RestMode");

        private PlayerController player;
        private Animator animator;
        private NetworkAnimator netAnimator;
```

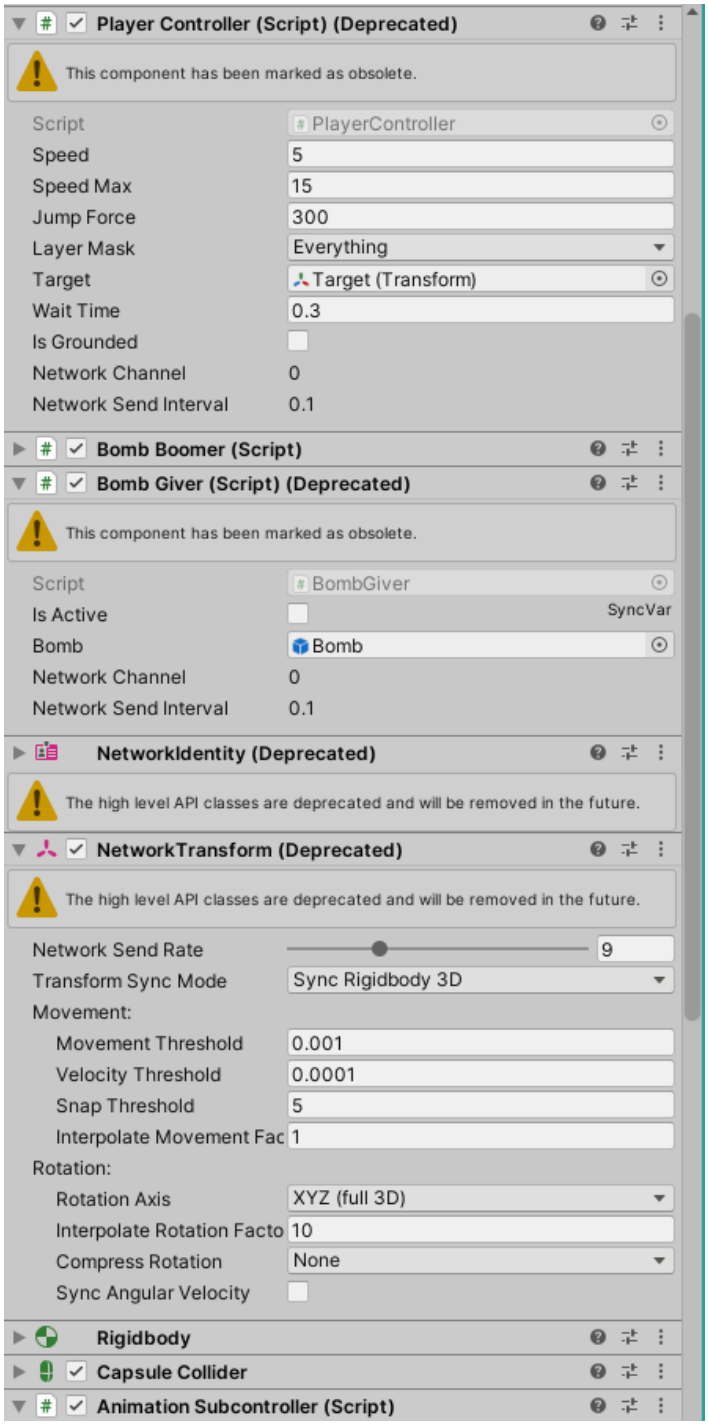
```
        private Random rn;

        private void Awake()
        {
            netAnimator =
GetComponent<NetworkAnimator>();
            netAnimator.SetParameterAutoSend(0, true);
            netAnimator.SetParameterAutoSend(1, true);
            netAnimator.SetParameterAutoSend(2, true);
            netAnimator.SetParameterAutoSend(3, true);
            netAnimator.SetParameterAutoSend(4, true);
            netAnimator.SetParameterAutoSend(5, true);



            animator =
GetComponent<NetworkAnimator>().animator;
            rn = new Random();

            player = GetComponent<PlayerController>();
        }
```

```
        private void Update()
        {
            if (isLocalPlayer)
            {
                Resting();


                Walking();



                Jumping();
                Flying();
            }
        }
        private void Resting()
        {
            var stateInfo =
animator.GetCurrentAnimatorStateInfo(0);
            if (stateInfo.IsName("Idle0"))
            {
                switch (rn.Next(2))
                {
```

```
                    case 0:
                        animator.SetTrigger(idle1);
                        animator.ResetTrigger(idle2);
                        break;

                    case 1:
                        animator.SetTrigger(idle2);
                        animator.ResetTrigger(idle1);
                        break;
                }
            }
        }
        private void Walking()
        {
            if (player.isGrounded)
            {
                animator.SetFloat(speedX,
Input.GetAxis("Horizontal"));
                animator.SetFloat(speedY,
Input.GetAxis("Vertical"));
            }
            else
```

```
            {
                animator.SetFloat(speedX, 0);
                animator.SetFloat(speedY, 0);
            }

        }

        private void Jumping()
        {
            if (Input.GetButtonDown("Jump") &&
player.isGrounded)
            {
                netAnimator.SetTrigger(isJumping);
                animator.SetTrigger(isJumping);
            }
            else
                animator.ResetTrigger(isJumping);
        }
        private void Flying() ⇒
animator.SetBool(isFlying, !player.isGrounded);
    }
```

Додаток К


Player Controller (Script) (Deprecated)
This component has been marked as obsolete.
Script
PlayerController
Speed
5
Speed Max
15
Jump Force
300
Layer Mask
Everything
Target
Target (Transform)
Wait Time
0.3
Is Grounded
Network Channel
0
Network Send Interval
0.1
Bomb Boomer (Script)
Bomb Giver (Script) (Deprecated)
This component has been marked as obsolete.
Script
BombGiver
Is Active
SyncVar
Bomb
Bomb
Network Channel
0
Network Send Interval
0.1
NetworkIdentity (Deprecated)
The high level API classes are deprecated and will be removed in the future.
NetworkTransform (Deprecated)
The high level API classes are deprecated and will be removed in the future.
Network Send Rate
9
Transform Sync Mode
Sync Rigidbody 3D
Movement:
Movement Threshold
0.001
Velocity Threshold
0.0001
Snap Threshold
5
Interpolate Movement Fac
1
Rotation:
Rotation Axis
XYZ (full 3D)
Interpolate Rotation Facto
10
Compress Rotation
None
Sync Angular Velocity
Rigidbody
Capsule Collider
Animation Subcontroller (Script)