Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ГРИ З ВИКОРИСТАННЯМ КЕРУВАЛЬНИХ ПОВЕДІНОК

Текстова частина до кваліфікаційної роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Бучко О.А.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2024 р.

Виконав студент _____
____Близнюк І.О._____
(прізвище та ініціали)
“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Близнюк Іван Олегович факультету інформатики 4-го курсу

ТЕМА Розробка гри з використанням керувальних поведінок

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Анотація
Вступ
1 Теоретичні аспекти ігрового жанру пазл
2 Види «керувальних поведінок» та їх використання
3 Реалізація
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2024 р. Бучко О.А. _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка гри з використанням керувальних поведінок

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	20.09.2023	
2.	Аналіз матеріалів за темою	30.12.2023	
3.	Розробка та програмування алгоритму	01.03.2024	
4.	Написання текстової частини до курсової роботи	01.05.2024	
5.	Коригування виконаної роботи	08.05.2024	
6.	Створення слайдів для доповіді та написання доповіді.	13.05.2024	
7.	Остаточне оформлення роботи та слайдів	20.05.2024	
8.	Захист курсової роботи	27.05.2024	

Близнюк І. О. _____

Бучко О.А. _____

“_____”_____

Анотація

Робота присвячена дослідженню керувальних поведінок (Steering Behaviors) та їх використанню для створення гри в жанрі «пазл» з елементами стратегії в реальному часі. Гра має наступні функціональні можливості: віддавання наказів групі персонажів, комплексна поведінка персонажів, взаємодія персонажів з оточенням та система рівнів.

Вступ

Швидкий розвиток індустрії ігрової розробки зумовив зростання рівня комплексності кожної складової ігрових проектів, включаючи неігрових персонажів та їх поведінку. Одним з рішень для опрацювання такого складного аспекту є використання алгоритмів «керувальних поведінок», що значно спрощують опис сукупності дій цих персонажів. За мету роботи була поставлена розробка гри в жанрі «пазл» з використанням «керувалних поведінок» та їх дослідження.

Основна частина роботи складається з 3 розділів:

Перший розділ присвячено жанру «пазл», його особливостям, етапам розвитку та причинам вибору саме такого жанру для створення програмного продукту.

У другому розділі розглядаються основні види «керувальних поведінок» та їх особливості використання.

У третьому розділі описується реалізований програмний продукт, комп’ютерна гра, створена інструментами рушія Unity та мови C# з використанням «керувальних поведінок» для реалізації поведінки ігрових персонажів.

Основна частина

Розділ 1: Теоретичні аспекти ігрового жанру пазл

1.1 Особливості жанру

Пазл або головоломка[1] – це жанр відеоігор, в якому основною ціллю гравця є розв’язування певних головоломок, які можуть перевіряти різні види його ментальних навичок від логіки до словникового запасу. У гравця може бути невичерпний запас часу, або нескінченна кількість спроб для вирішення загадки, або може бути присутній ліміт часу. Прості головоломки можуть бути ускладнені шляхом додавання до них елементу реального часу (як, наприклад, Tetris). Хоча багато пригодницьких або екшен-ігор включають такі головоломки, як пошук недоступних об’єктів, справжня гра-головоломка зосереджена на розв’язанні головоломок як основної ігрової діяльності.

Зазвичай, замість того, щоб представляти випадковий набір головоломок для вирішення, ігри-головоломки пропонують серію пов’язаних головоломок, які є варіаціями на одну тему. Ця тема може включати розпізнавання образів, логіку або розуміння певних процесів. Ці ігри зазвичай мають простий набір правил, де гравці маніпулюють ігровими фігурами на сітці, в певній мережі чи іншому просторі для взаємодії. Гравці повинні знаходити підказки, щоб досягти певної умови перемоги, яка потім дозволить їм перейти на наступний рівень. Виконання кожної головоломки зазвичай призводить до складнішого завдання, хоча деякі ігри не виснажують гравця, пропонуючи простіші рівні між більш складними.

1.2 Етапи розвитку жанру

Загалом, відеоігри-пазли пішли від логічних головоломок та загадок, що існували протягом більшості історії людства. Тому не дивно, що з розвитком електроніки, саме головоломки стали одними з перших цілей для діджиталізації.

Одним з найперших представників жанру вважають гру «Heiankyo Alien»[2], розроблену у 1979 в університеті Токіо для PC-8001, у якій гравець повинен був готувати пастки для прибульців всередині лабіринту.

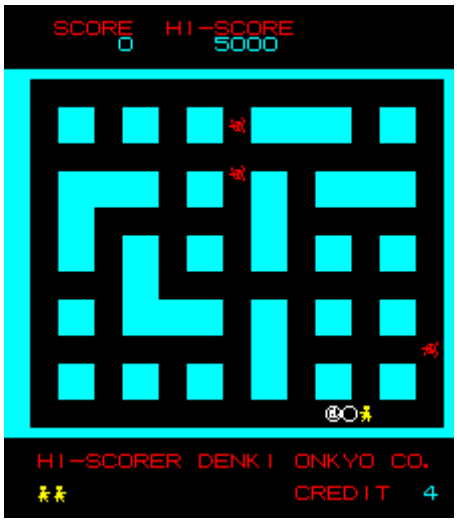

Рис. 1 Гра Heiankyo Alien

Першою грою, яка містила слово «пазл» у назві була Puzzle Panic[3] 1984 року, випущена для комп'ютера Atari 8-bit, де гравець керував лампочкою та повинен був обирати правильні двері, що позначені різними символами, аби переходити на нові рівні.

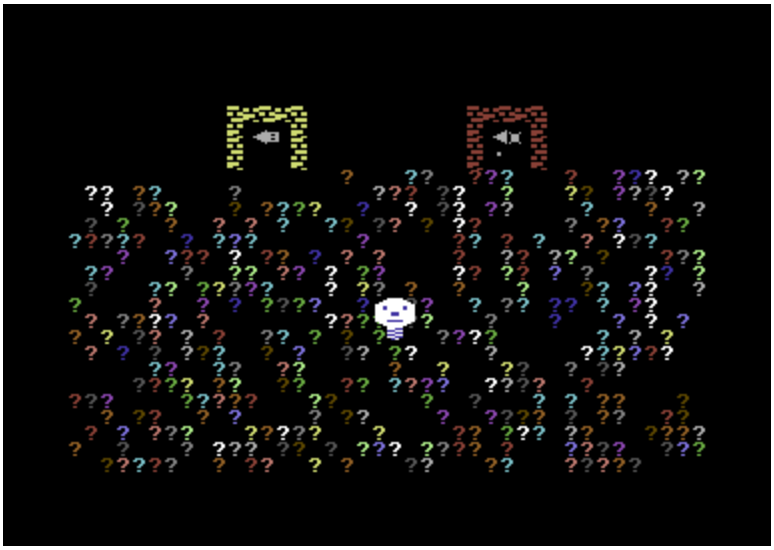

Рис. 2 Гра Puzzle Panic

Першу популярність жанру принесла всім відома гра Tetris[4] випущена у 1984 в радянському союзі, де гравець повинен був керувати падаючими тетраміно та складати з них ряди без пропусків.

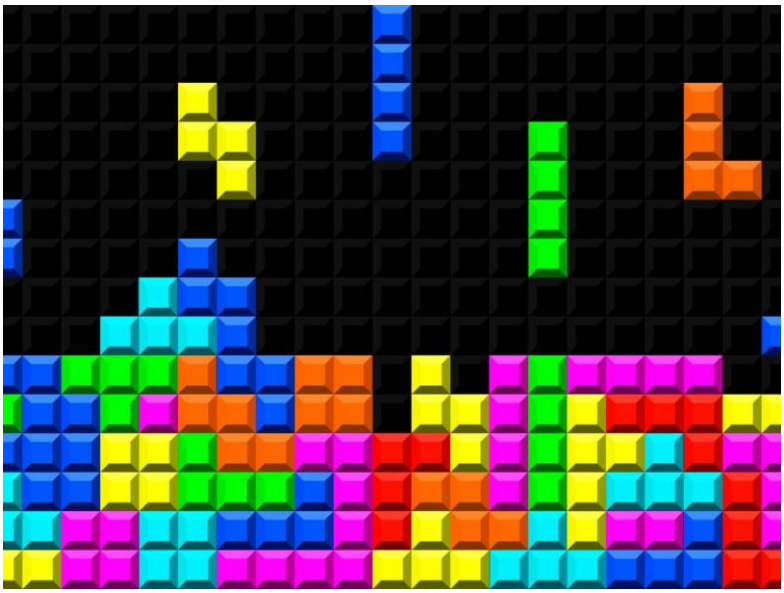

Рис. 3 Гра Tetris

Minesweeper[5] (або сапер) – це гра, що вийшла у 1995 разом з Windows 95, де гравець повинен був розмінувати мінне поле, спираючись на те, скільки мін знаходиться навколо кожної клітинки, та яка популяризувала

використання комп’ютерної миші у пазлах[6], адже до цього вони здебільшого керувались виключно з клавіатури.

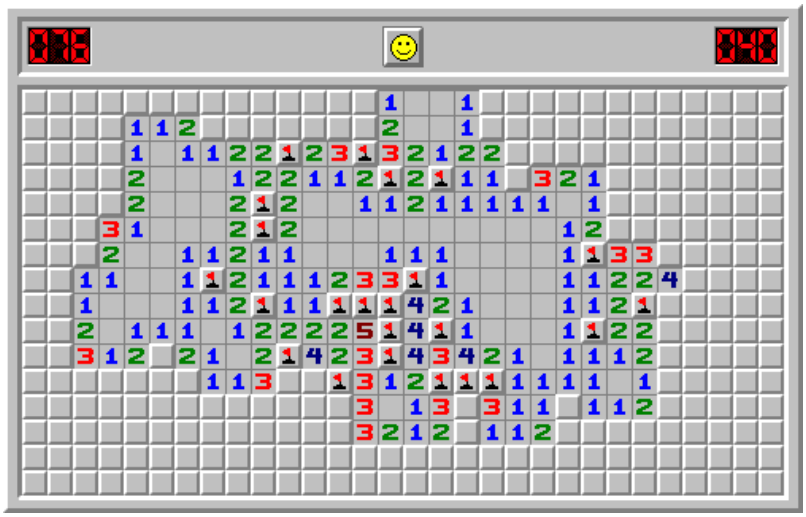

Рис. 4 Гра Minesweeper

В сучасному світі, «пазл» є дуже популярним жанром ігор для мобільних пристроїв, адже такі ігри зазвичай не вимагають великої кількісті системних ресурсів та здатні надовго зайняти гравця. Гарними прикладами є серія Angry Birds[7], або Candy Crush Saga[8], що є однією з найприбутковіших мобільних ігор всіх часів.

Рис. 5 Гра Angry Birds

Проте, існують і сучасні пазли для персональних комп’ютерів та консолей, такі проекти як серія Portal[9] або Entropy Centre[10] є успішними прикладами головоломок, що засновані на фізиці у тривимірному просторі.

1.3 Причина вибору жанру

Вибір саме жанру «пазл» для створення гри можна обґрунтувати кількома факторами:

- Сприятливе середовище для демонстрації Керувальних Поведінок: суть жанру «пазл» в розв’язанні гравцем певних логічних задач або головоломок з використанням наданих йому розробниками інструментів. В такому середовищі легко розкрити потенціал обраної теми, адже згідно до обраного концепту, гравець повинен вирішувати головоломки шляхом надавання наказів певним сутностям. Накази в свою чергу є варіантами реалізації різних поведінок. Таким чином, гравець вимушений постійно з ними взаємодіяти, що створює чудове підґрунтя для їх впровадження.
- Варіативність: Пазли зазвичай містять широкий спектр різноманітних задач та способів їх розв’язання. Наприклад, навігація в лабіринтах, уникання пасток, поведінка ворогів. Таким чином, створюються різноманітні ситуації, де може бути корисним впровадження Керувальних поведінок.
- Підвищення зацікавленості гравців: Ігри у жанрі «пазл» відносяться до інтелектуальних розваг. Таким чином, гравці можуть бути більш зацікавленні в ефективному використанні наданого функціоналу для розв’язання задач, а тобто у використанні та комбінуванні Керувальних Поведінок.

Розділ 2: Керувальні пооведінки (Steering Behaviours)

2.1 Загальний опис

Кервувальні поведінки (або Steering Behaviours)[11] спрямовані на те, щоб допомогти автономним персонажам рухатися природньо, використовуючи прості сили, що поєднуються, щоб створювати реалістичну імпровізаційну навігацію по навколишньому середовищу. Вони не ґрунтуються на складних стратегіях, що включають планування шляху чи глобальні розрахунки, а натомість використовують локальну інформацію, наприклад, сили, що впливають на сусідів. Це робить їх простими для розуміння та впровадження, але все ще здатними імітувати дуже складні моделі рухів.

2.2 Поведінка Seek (Пошук)

Пошук[12] є найпростішим видом поведінки, при якому сутність намагається досягнути певної цілі найкоротшим шляхом. Керуюча сила в такому випадку розраховується як різниця між поточною швидкістю та бажаною швидкістю, яка в свою чергу є просто напрямом від поточної позиції до позиції цілі.

Оскільки керуюча сила впливає саме на швидкість сутності, а не на її позицію напряму, шлях пошуку буде не прямою лінією, а буде залежати від початкової швидкості та напряму руху, та буде плавно прямувати до потрібного.

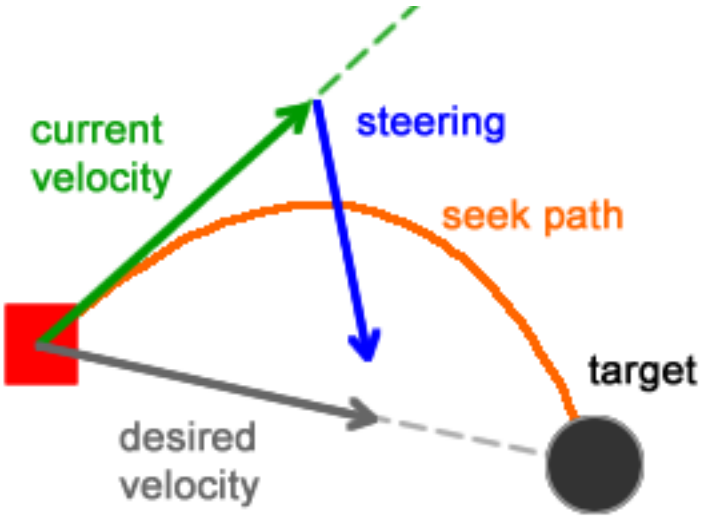

Рис. 6 Поведінка Seek[12]

Незважаючи на те, що пошук є найпростішою поведінкою, її можна вдосконалити шляхом доповнення поведінкою прибуття. В базовій реалізації, після досягнення цілі, сутність починає хитатися вперед-назад, адже керуюча сила все-ще спрямовує її до цілі і вона опиняється то з одного, то з іншого боку, що виглядає дуже неприродньо.

Поведінка прибуття (Arrival)[13] забороняє сутності рухатись крізь ціль, сповільнюючи тим більше, чим ближче вона до цілі. Для цього, обирається деякий радіус, що задає «зону сповільнення» навколо цілі. Якщо сутність знаходиться за зоною сповільнення, використовується звичайна поведінка Пошуку. Якщо ж персонаж знаходиться всередині зони сповільнення, його бажана швидкість множиться на коефіцієнт, що дорівнює відношенню відстані між сутністю та ціллю, до радіусу зони сповільнення. Таким чином, при наближенні до цілі, бажана швидкість буде прямувати до нуля.

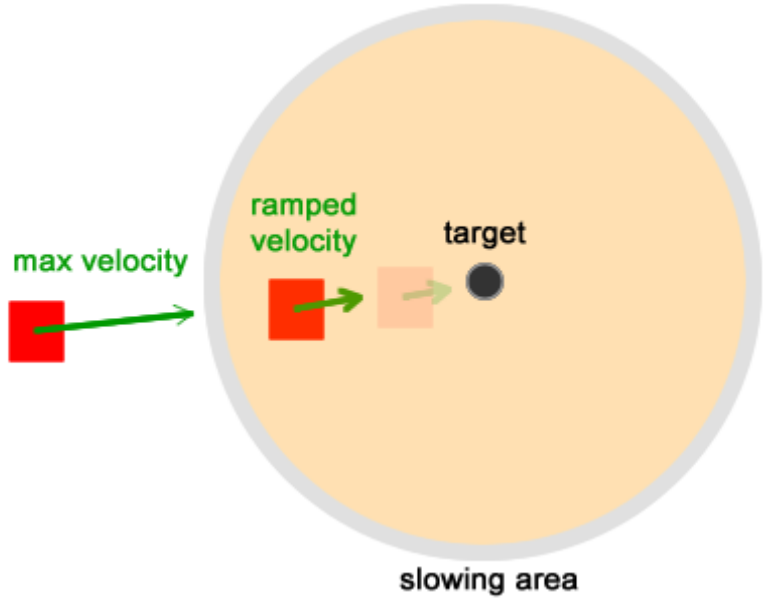

Рис. 7 Поведінка Arrival[13]

2.3 Поведінка Flee (Втеча)

Втеча[13] є дуже схожою на пошук, але при ній сутність намагається якомога швидше втекти від деякої цілі. Керуюча сила в такому випадку розраховується як різниця між поточною швидкістю та бажаною швидкістю, яка в свою чергу є оберненою до напряму від поточної позиції до позиції цілі.

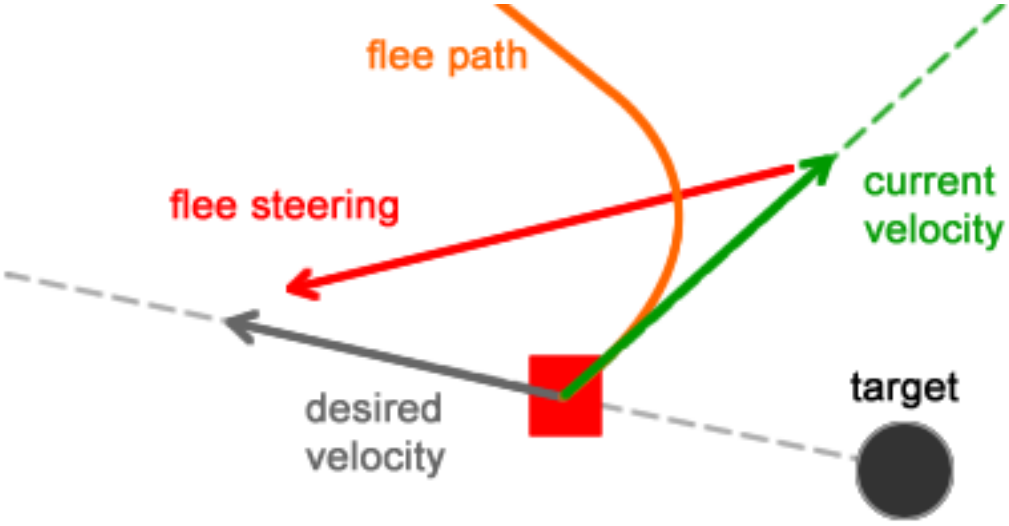

Рис. 8 Поведінка Flee[13]

2.4 Поведінка Pursuit (переслідування)

Загалом, переслідування описує процес слідування за деякою сутністю з ціллю її впіймати[14]. Варто зауважити, що саме спроба впіймати відрізняє його від поведінки пошуку або простого слідування, адже для цього достатньо повністю повторювати рухи цілі.

Для ефективного переслідування, сутність повинна намагатись передбачити, де буде знаходитись ціль у майбутньому. Відповідно, якщо таке передбачення можливе, можна відповідно скорегувати траєкторію свого руху, аби зробити свій маршрут більш оптимальним.

Для передбачення положення цілі у майбутньому, можна скористатися тим, що всі описані поведінки ґрунтуються на фізичній моделі, тому аби передбачити положення цілі в просторі через час T, достатньо порахувати наступне:

$$\bar{F} = \bar{C} + \bar{V} * T$$

Де $\bar{F}$ – положення цілі в майбутньому, $\bar{C}$ – поточне положення цілі, $\bar{V}$ – поточна швидкість цілі.

Ключовим для ефективного переслідування є правильний вибір значення параметру T, адже якщо воно буде завелике, переслідувач по суті буде намагатись впіймати привида, що знаходиться попереду цілі, а якщо ж воно буде замале, переслідувач буде просто прямувати за ціллю, не намагаючись вгадати її положення в майбутньому.

Не зважаючи на те, що такий підхід не враховує можливі зміни швидкості цілі, він все одно є ефективним з наступних причин:

1. Передбачити зміни швидкості цілі не завжди можливо (особливо, у випадку взаємодії з гравцем)

2. Навіть якщо зміну швидкості цілі можна передбачити, цього не варто робити, адже це зробить поведінку переслідувача неприродньою (переслідувач може почати рухатись зовсім нелогічно для стороннього наглядача) та прибере ситуації, коли ціль обманює переслідувача, різко змінюючи свій напрям.
3. Переслідування буде достатньо ефективним навіть при різкій зміні швидкості цілі, якщо напрям переслідування буде перераховуватись достатньо часто (в ідеалі, кожну одиницю часу) та якщо параметр T буде достатньо малим (зазвичай, долі секунди).

Для збільшення точності переслідування, параметр T можна обраховувати динамічно, як відношення відстані між переслідувачем та ціллю до максимальної швидкості переслідувача, таким чином переслідувач буде максимально точно передбачувати позицію цілі в майбутньому та уникне проблем з неправильним вибором T.

Таким чином, для реалізації поведінки достатньо порахувати значення параметру T, порахувати позицію цілі у майбутньому, використавши наведену формулу, та використати вже описану поведінку пошуку (Seek) для слідування у точку, де ціль буде знаходитись у майбутньому.

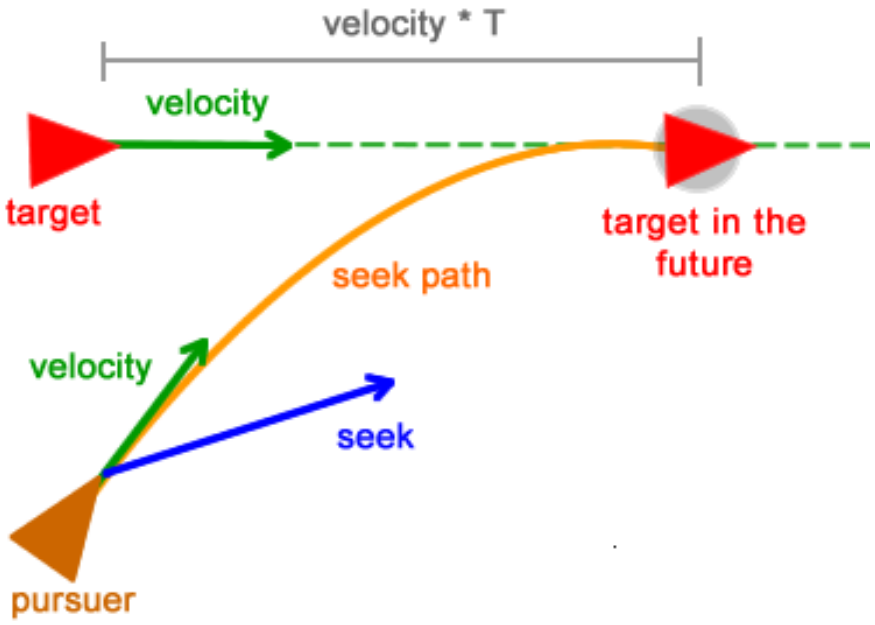

Рис. 9 Поведінка Pursuit[14]

2.5 Поведінка Evade (уникання)

Поведінка Evade[14] описує процес уникання переслідувача та є оберненою до поведінки Pursue. Тобто, використовує ту ж саму логіку реалізації, за винятком того, що втікач, розрахувавши позицію переслідувача, використовує поведінку Flee, аби максимально віддалятись від цієї позиції.

До уникання також можна застосувати динамічний обрахунок параметру T.

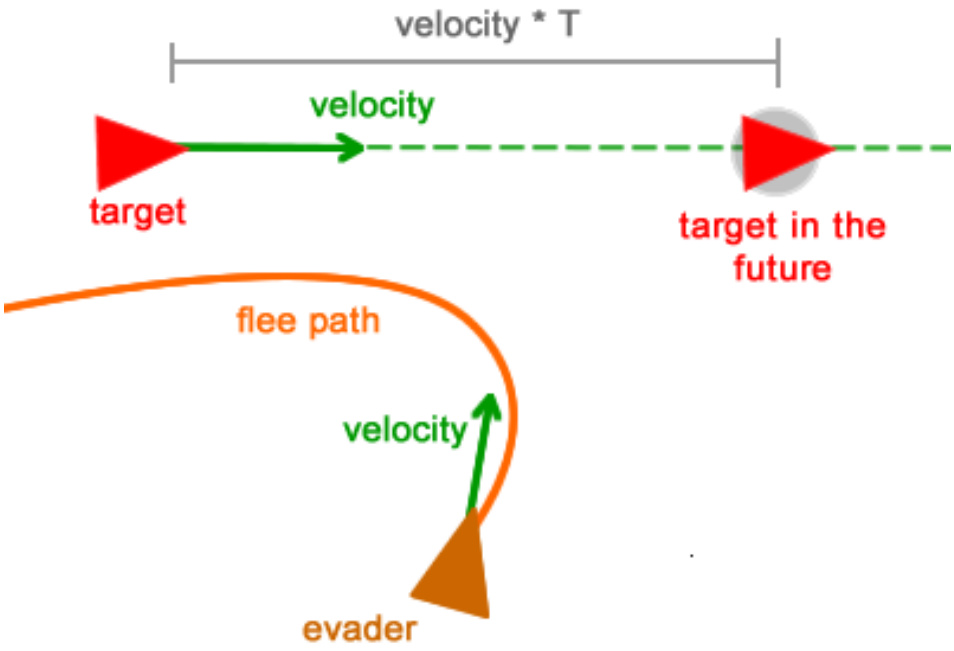

Рис. 10 Поведінка Evade[14]

2.6 Поведінка Collision Avoidance (уникання колізій)

Основна ідея уникання колізій[15] полягає в створенні керуючої сили для уникання перешкод кожного разу, коли сутність достатньо наближається до них. Навіть якщо у навколишньому середовищі присутні декілька перешкод, ця поведінка використовує одне з них для обрахування сили уникання. Аналізуються лише перешкоди, які стоять перед персонажем; для оцінки вибирається найближча, яка вважається найбільш загрозливою. В результаті персонаж здатний ухилятися від усіх перешкод, органічно переходячи від однієї до іншої.

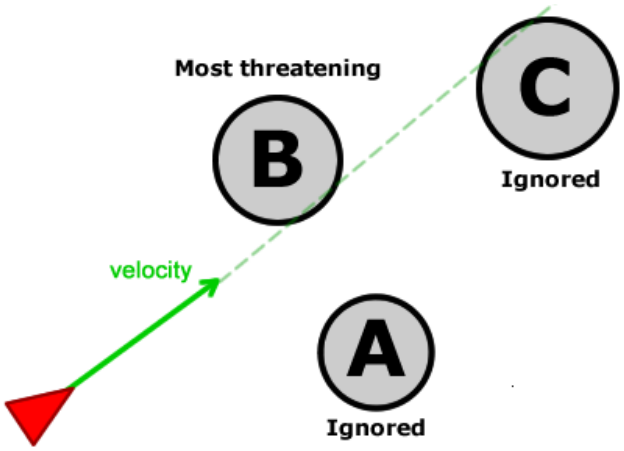

Рис. 11 Поведінка Collision Avoidance[15]. Вибір перешкоди для уникання

Варто зауважити, що поведінка уникання колізій не є алгоритмом пошуку шляху (pathfinding). Хоча вона може забезпечити рух сутностей в просторі, уникаючи перешкод, вона не дуже ефективно працює з перешкодами форм «L» або «T».

Для реалізації поведінки, спочатку обирається довжина поля зору, тобто величина, що описує, наскільки далеко сутність бачить перешкоди. Чим більша ця величина, тим раніше сутність почне ухилятись від перешкоди. Потім, відбувається пошук колізій на відстані довжини діапазону зору та у напрямку руху сутності, аби перевірити, чи є перешкоди, що заважають руху. Цього можна досягнути або математично обраховуючи відстань від точки кінця поля зору до центру перешкоди (для покращення ефективності поведінки, всі перешкоди вважаються круглими або сферичними), або скористатися киданням променів (ray casting), що полягає в створенні променю у відповідній точці з напрямом та довжиною і перевіркою, з якими об’єктами перетнувся промінь. Перевага кидання променів у простоті використання та розповсюдженості (ray casting є стандартним функціоналом більшості ігрових рушіїв).

Наступний крок – розрахунок сили уникання. Ця сила має відштовхувати сутність від перешкоди (що представлена колом або сферою), дозволяючи ухилитись від неї. Її можна розрахувати за такою формулою:

$$\bar{F} = \bar{P} - \bar{A} - \bar{O}$$

Де $\bar{A}$ – вектор положення кінця поля зору сутності, $\bar{O}$ – центр перешкоди, $\bar{P}$ – положення сутності в просторі. Для більш стабільної поведінки, вектор сили уникання варто потім нормалізувати та помножити на скалярну величину, що задає модуль сили уникання перешкод.

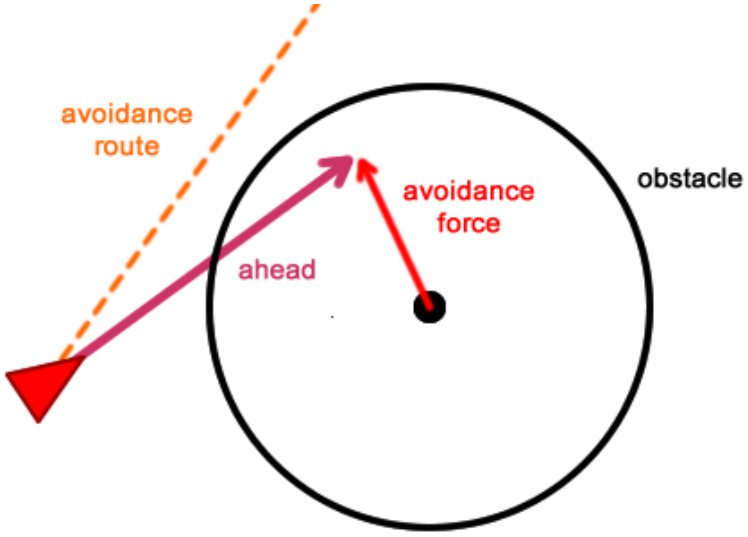

Рис. 12 Поведінка Collision Avoidance[15]. Розрахунок сили уникання

Проте, такий підхід має один недолік, а саме – якщо сутність знаходиться біля перешкоди та починає розвертатись, може відбутись колізія і сутність почне відштовхувати від перешкоди, хоча вона навіть не рухалась в її напрямі. Вирішити цю проблему можна шляхом додаткового множення довжини поля зору на коефіцієнт, рівний відношенню модулю поточної швидкості до модулю максимальної швидкості, що може досягати сутність.

2.7 Поведінка Path Following (слідування шляху)

Слідування шляху[16] можна реалізувати декількома шляхами, наприклад, створення шляху з набору ліній, по яких сутності рухаються як по рейках. Проте, така точність не завжди потрібна, тому сутність може рухатись по шляху з ліній, але використовувати їх як орієнтир, а не жорсткі рейки.

Для реалізації потрібно задати шлях як набір вершин, поєднаних прямими лініями (криві краще апроксимізувати до ламаних ліній, адже це сильно полегшує обрахунки, а відповідно дає виграш по часу виконання алгоритму).

Для навігації між вершинами, можна використати вже описану поведінку Seek, розглядаючи вершини як цілі пошуку. Сутність рухається до поточної вершини, доки не досягне її, після чого наступна вершина зі шляху стає поточною. Оскільки всі поведінки обраховуються кожну одиницю часу, зміна поточної вершини проходить непомітно.

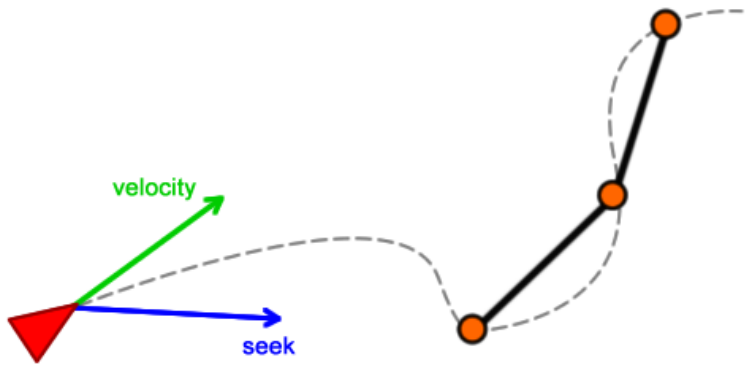

Рис. 13 Поведінка Path Following[16]

Проте, при такій реалізації, сутність має доторкнутись до поточної точки шляху для переходу до наступної і як наслідок, може рухатись дещо неприродньо, наприклад кружляти навколо вершини, доки не доторкнеться її.

У природі, рух зазвичай підкоряється принципу найменших зусиль, тобто людина не завжди тримається середини коридору, а наближається до стіни перед поворотом, щоб трохи скоротити шлях. Така модель поведінки може бути відтворена шляхом додавання радіусу до шляху. Радіус застосовується до вершин, і його можна розглядати як «ширину» шляху. Він контролюватиме, на яку відстань сутність може відійти від вершин на шляху. Таким чином, сутність переходитиме до наступної вершини шляху не тоді, коли досягне її, а тоді, коли вона опиниться на відповідній (або меншій) відстані від неї.

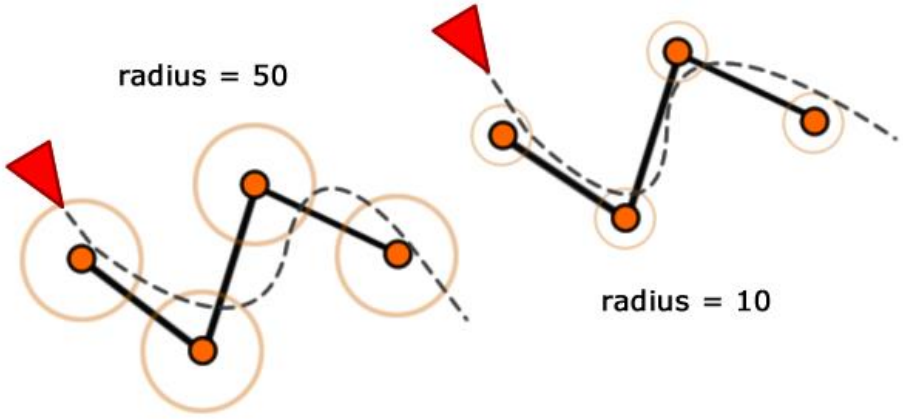

Рис. 14 Path Following з радіусом шляху[16]

2.8 Поведінка Leader Following (слідування за лідером)

Поведінка слідування за лідером[17] полягає в слідуванні сутності (або їх групи) за іншою сутністю, яку призначено лідером. Її реалізація полягає в комбінуванні інших вже реалізованих поведінок. Найпростіший підхід – використати поведінки пошуку або переслідування для створення ефекту слідування, проти вони не дають достатньо природніх результатів. При використанні поведінки пошуку, сутність прямує до цілі та зрештою починає займати те ж саме місце, що і ціль, а при використанні переслідування, сутність намагається впіймати ціль, а не слідує позаду неї.

Суть поведінки слідування за лідером полягає в тому, щоб знаходитись достатньо близько до лідера, але бути трохи позаду нього. Сутність повинна швидко прямувати до лідера, якщо знаходиться далеко від нього, та сповільнюватись, коли відстань мала. Такого ефекту можна досягти, поєднуючи три інші поведінки:

1. Пошук (з доповненням у вигляді поведінки прибуття) – сутність має рухатись до лідеру та сповільнятись, підходячи достатньо близько.
2. Уникання – якщо сутність знаходиться на шляху лідера, вона має звільнити шлях.
3. Розділення (допоміжна поведінка, що дозволяє групі сутностей триматись на певній відстані одна від одної) - щоб уникнути скупчення, коли кілька сутностей слідують за одним лідером.

Для реалізації поведінки спочатку обирається точка, що знаходиться позаду лідера та до якої прямуватимуть сутності. Координати цієї точки можна обрахувати за такою формулою:

$$\bar{B} = \frac{-\bar{V}}{|\bar{V}|} \cdot d + \bar{P}$$

Де $\bar{V}$ – поточна швидкість лідера (що нормалізується), d – скалярна величина, що характеризує, наскільки далеко позаду від лідера мають триматись сутності та $\bar{P}$ – поточна позиція лідера. А коли точка позаду лідера обрахована, можна використати поведінку пошуку (з використанням поведінки прибуття), аби сутність прямувала до неї.

Проте, якщо сутності будуть знаходитись занадто близько одна до одної, результат може виглядати неприродньо, оскільки на сутності будуть впливати схожі сили та вони будуть рухатись схожим чином, формуючи велику

«краплю». Зробити цю поведінку більш природньою може використання допоміжної поведінки Separation (розділення).

Розділення[18] – це поведінка, яка змушує сутність віддалятись від її сусідів. Сила розділення розраховується за такою формулою:

$$\bar{F} = \frac{1}{n}\sum_{i=1}^{n}(\bar{P} - \bar{P}_i)$$

Де $\bar{P}$ – позиція поточної сутності, $\bar{P}_i$ – позиція інших сутностей, відстань між якими та поточною сутністю менша за деякий поріг, n – кількість інших сутностей.

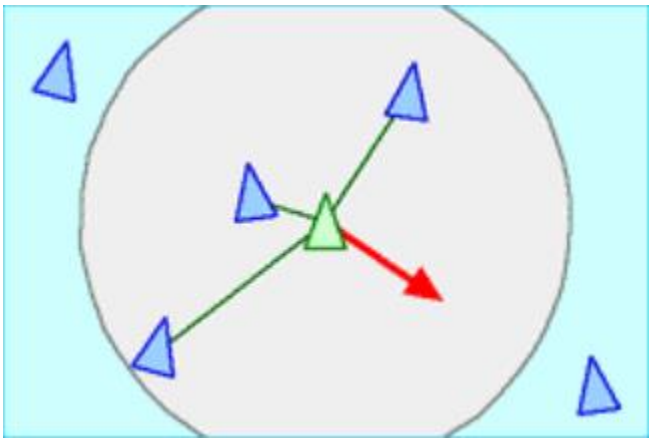

Рис. 15 Поведінка Separation[18]

Ще одним неприроднім моментом може стати ситуація, коли сутність опиниться на шляху лідера. Оскільки сутності мають прямувати за лідером, немає сенсу дозволяти їм залишатися попереду нього. Тому якщо така ситуація виникає, сутність повинна посунутись аби звільнити шлях. Такої поведінки можна досягнути з допомогою використання поведінки уникання.

Для перевірки, чи знаходиться сутність на шляху лідера, спочатку обирається точка попереду, позиція якої розраховується за такою формулою (схожою на спосіб розрахунку точки позаду лідера, але швидкість береться з іншим знаком):

$$\bar{B} = \frac{\bar{V}}{|\bar{V}|} \cdot d + \bar{P}$$

Де $\bar{V}$ – поточна швидкість лідера (що нормалізується), d – скалярна величина, що характеризує наскільки далеко позаду від лідера мають триматись сутності, $\bar{P}$ – поточна позиція лідера. Відповідно, якщо сутність знаходиться від $\bar{B}$ на відстані, меншій за певну величину (довжина поля зору лідера), сутність починає уникати лідера, використовуючи поведінку уникання.

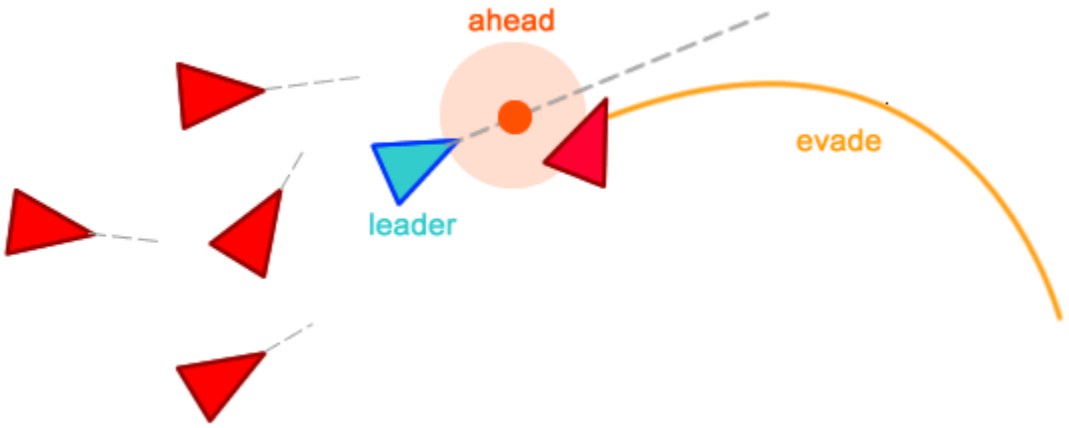

Рис. 16 Поведінка Leader Following[17]. Уникання лідера

Розділ 3: Реалізація

3.1 Ігровий рушій Unity

Unity – це кросплатформний ігровий рушій, який широко використовується великими ігровими студіями та є особливо популярним серед незалежних розробників та невеликих студій[19], через свою зручність у використанні, широкий набір інструментарію та високу якість документації.

Незважаючи на те, що основа рушія створена мовою C++, для розробки на ньому використовується саме C#, адже він є простішим у використанні, зокрема завдяки вбудованому збирачу сміття[20].

Рис. 17 Unity Editor

Весь ігровий процес та інтерактивність, розроблені в Unity, будуються на трьох основних будівельних блоках: ігрові об’єкти (GameObjects), Компоненти (Components) та змінні (Variables).

Будь-який об’єкт у грі є GameObject[21]: персонажі, джерела світла, спеціальні ефекти або реквізит.

GameObjects не можуть нічого робити самостійно. Щоб справді стати частиною гри, потрібно надати GameObject певні властивості та можливості, шляхом додавання до них компонентів.

Компоненти[22] визначають і контролюють поведінку GameObjects, до яких вони приєднані. Простим прикладом може бути створення джерела світла, яке передбачає прикріплення компонента Light до GameObject. Або додавання компонента RigidBody до об’єкта, щоб він почав підкорюватись законам фізики як справжнє тверде тіло.

Компоненти можуть мати деяку кількість редагованих властивостей або змінних, які можна налаштувати прямо в графічному редакторі Unity та/або за допомогою коду. Наприклад, для компоненту Light такими властивостями є дальність, колір та інтенсивність світла.

Вбудовані компоненти Unity надають широкий функціонал, проте зазвичай для створення програмного продукту потрібно вийти за рамки того, що вони можуть надати. Для цього можна використовувати скрипти (Scripts)[23] для реалізації власної ігрової логіки та поведінки, які потім можна додавати як компоненти до GameObjects. Кожен скрипт зв’язується з внутрішньою логікою рушія і є класом, який походить від вбудованого класу під назвою MonoBehaviour[24].

Оскільки ігрові об'єкти часто використовуються повторно, в Unity існує система Prefabs[25], яка дозволяє створювати, налаштовувати та зберігати GameObject разом із усіма його компонентами, значеннями властивостей і дочірніми ігровими об’єктами як багаторазовий ресурс. Ресурс Prefab діє як шаблон, за допомогою якого можна створювати нові екземпляри Prefab на сцені.

3.2 Реалізація керувальних поведінок. Клас Steering Manager

Оскільки однією з основних переваг керувальних поведінок є можливість їх одночасного використання та комбінування, для їх реалізації було прийняте рішення створити один спільний скрипт Steering Manager, який буде застосовувати поведінки до об’єкту, до якого він прикріплений.

Steering Manager тісно співпрацює з іншим компонентом Unity, а саме RigidBody[26], який реалізує поведінку твердих тіл, що стає дуже в нагоді враховуючи, що керувальні поведінки ґрунтуються на фізичній моделі.

Як вже раніше було зазначено, кожний вид поведінки створює відповідну керуючу силу, яка має направити сутність у правильному напрямку. Сили, отримані від різних поведінок можна додавати, оскільки по своїй суті вони є двовимірними векторами, комбінуючи таким чином ці поведінки. Таким чином, Steering Manager має набір методів, кожен з яких є реалізацією певної поведінки. Поведінка застосовується у певний кадр (дискретний момент часу) якщо в цей кадр було викликано метод, відповідальний за цю поведінку. Відповідно, кожен викликаний метод додає власну керуючу силу до загально, накопичуючи її таким чином. Вкінці кадру, результуюча керуюча сила застосовується до сутності.

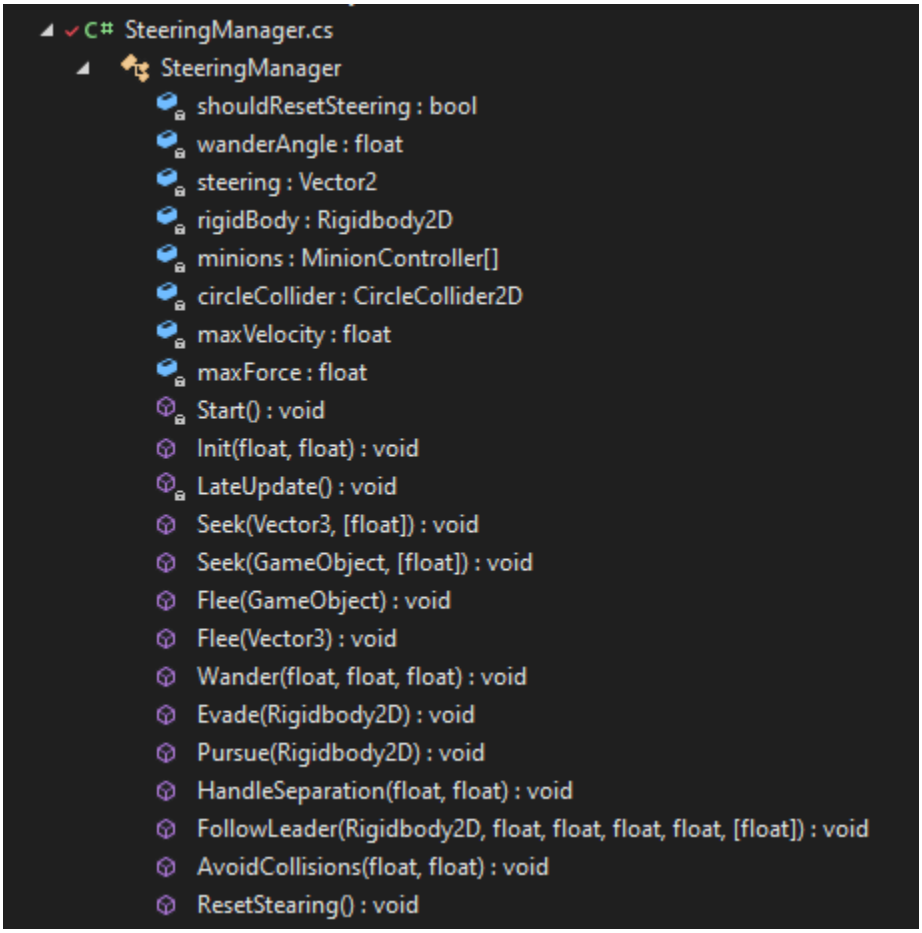

Рис. 18 Структура класу Steering Manager

Серед основних полів класу можна виокремити наступні:

- Steering – значення поточної керуючої сили
- rigidBody – посилання на компонент, що відповідає за реалізацію фізики твердого тіла
- maxVelocity – максимальна швидкість, до якої може розігнатись сутність
- maxForce – максимальне допустиме значення довжини вектору керуючої сили

Серед методів класу можна виокремити наступні:

- Start та Init – методи, потрібні для початкової ініціалізації

- Seek – реалізація поведінки пошуку з вбудованою поведінкою прибуття (з двома варіантами вхідних параметрів для випадків, коли шукаємо точку в просторі та пошуку іншої сутності)
- Flee – реалізація поведінки втечі (з двома варіантами вхідних параметрів для випадків, коли втеча відбувається від точки в просторі та коли вона відбувається від іншої сутності)
- Pursue – реалізація поведінки переслідування, де в якості вхідного параметру надається вказівник на RigidBody цілі, адже для виконання поведінки потрібна її поточна швидкість.
- Evade – реалізація поведінки уникання, де в якості вхідного параметру надається вказівник на RigidBody цілі, адже для виконання поведінки потрібна її поточна швидкість.
- HandleSeparation – реалізація допоміжної поведінки розділення (що використовується у реалізації поведінки слідування за лідером)
- FollowLeader – реалізація поведінки слідування за лідером, в якості вхідного параметру надаються вказівник на RigidBody лідера та декілька чисельних параметрів для більш тонкого налаштування поведінки.
- AvoidCollisions – реалізація поведінки уникання колізій, як параметри приймає довжину поля зору сутності та максимальну довжину вектору керуючої сили.
- LateUpdate[27] – службовий метод Unity, що потрібен для оновлення стану об’єктів та викликається кожен кадр, але після того, як відпрацюють всі методи Update. В даному класі він відповідальний за застосування обрахованої керуючої сили до об’єкту (через взаємодію з компонентом RigidBody). Використання саме LateUpdate зумовлене тим,

що застосування керуючої сили до об’єкту має відбуватись після застосування всіх потрібних поведінок в даному кадрі.

3.3 Опис програмного продукту

3.2.1 Загальна ідея гри та її основні механіки

В результаті виконання роботи маємо програмний продукт, що представляє собою гру, розроблену в жанрі «пазл» під назвою The Controller. Гравець бере на себе роль штучного інтелекту, який має керувати групою роботів-міньйонів, віддаючи їм накази різних видів і має довести певну їх кількість до фінішу аби пройти кожен рівень.

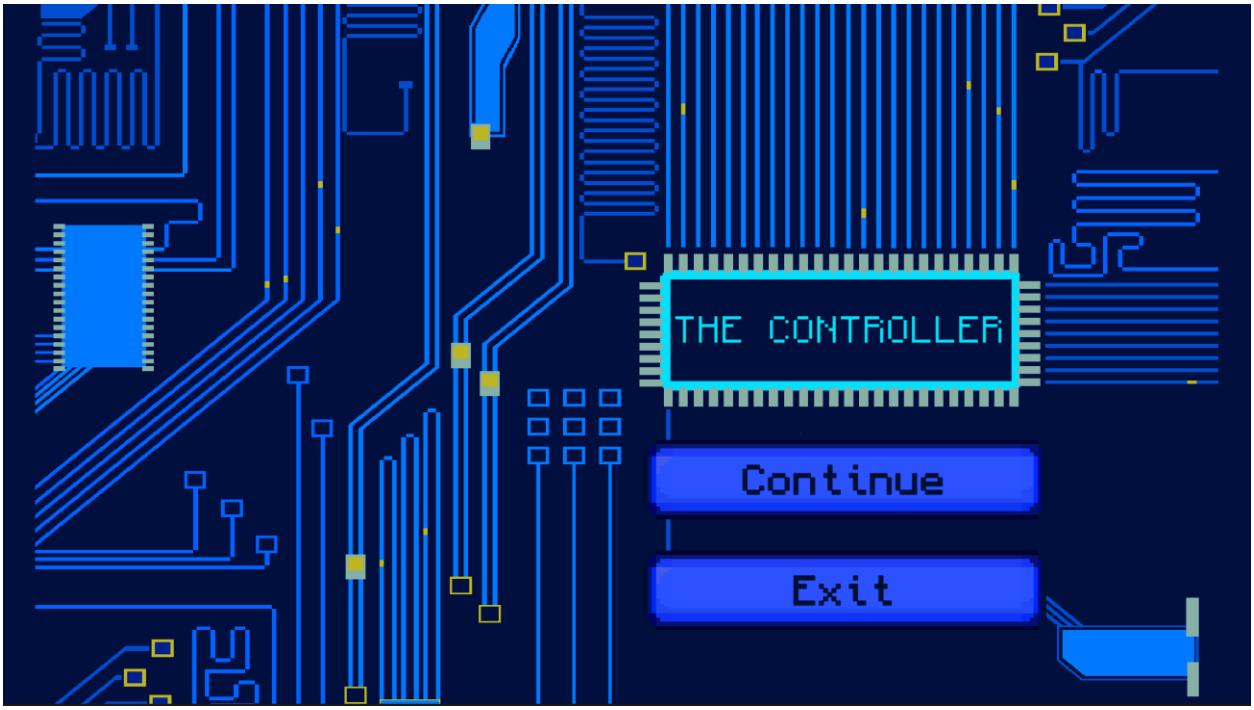

Рис. 19 Головне меню гри

Серед основних механік можна виділити такі:

- Віддавання наказів міньйонам – на кожному рівні у гравця є певна кількість наказів кожного типу, які він може віддавати своїм міньйонам.

Кожний наказ можна віддати як одному міньйону, так і цілій групі. Деякі накази можуть виконуватись одночасно (наприклад, прямувати у певну точку та тікати від ворога), а деякі послідовно, один за одним (декілька наказів «слідувати у певну точку» будуть виконуватись послідовно, тобто міньйон спочатку прийде в точку згідно першому наказу, а потім почне рухатись до точки другого наказу). Візуально, накази представлені у вигляді прапорів, які гравець може розміщувати на ігровому полі, або «чіпляти» їх до певних сутностей (наприклад, призначення лідеру можна застосувати лише на міньйона).

- Виділення груп міньйонів – накази можна віддавати не всім одразу, а певній групі міньйонів. Для зручності, групу можна обрати шляхом виділення певної області за допомогою курсору (за аналогією до стратегій в реальному часі або файлових менеджерів). Вже існуючу групу можна розширити, шляхом натиснення спеціальної клавіші (SHIFT) та виділення міньйонів, що слід додати до поточної групи. Якщо ж у групу потрапив зайвий міньйон, його можна прибрати з неї, шляхом затиснення тієї ж спеціальної клавіші та повторного його виділення.
- Інтерактивні об’єкти – на рівнях присутні деякі об’єкти з якими міньйонам необхідно взаємодіяти для розв’язання головоломок. Вони поділяються на два види – активатори та активовані об’єкти. Активатори – це об’єкти, взаємодіючи з якими, міньйони можуть активувати інші об’єкти (наприклад, натискні плити, на які може встати група міньйонів, аби натиснути її своєю вагою). Активовані об’єкти – це об’єкти, які можуть змінювати свій стан, відповідно до під’єднаного до них активатора (наприклад, лазер, який може вимикатись якщо натиснута під’єднана до нього натискна плита).

- Фази планування та виконання – ігровий процес поділений на дві стадії. В першій стадії, планування, гра фактично стоїть на паузі (міньйони та вороги не рухаються) та гравець може продумати стратегію проходження рівня, вільно віддаючи або відміняючи накази. Коли гравець закінчив планування, він може перейти до другої стадії. У другій стадії, виконання, міньйони починають виконувати надані накази для проходження рівня. Гравець все-ще може віддавати нові накази «на ходу», якщо вони в нього залишились, проте вже не може відміняти їх. Якщо обрана тактика виявилась неефективною, гравець може повернутись назад на стадію планування, тоді стан рівня повернеться до початкового і гравець зможе змінити план дій.
- Вороги – іноді на шляху міньйонів присутні вороги, які патрулюють певну територію, а якщо помічають міньйона, починають його переслідувати аби вбити, після чого повертаються до патрулювання.
- Поділ на рівні – гра складається з послідовності рівнів, кожен з яких представляє собою певну головоломку, яку повинен розв'язати гравець, надавши міньйонам правильну послідовність наказів. Для завершення рівня, відповідна кількість міньйонів має опинитись достатньо близько до супутникової тарілки, що є тригером кінця рівня.

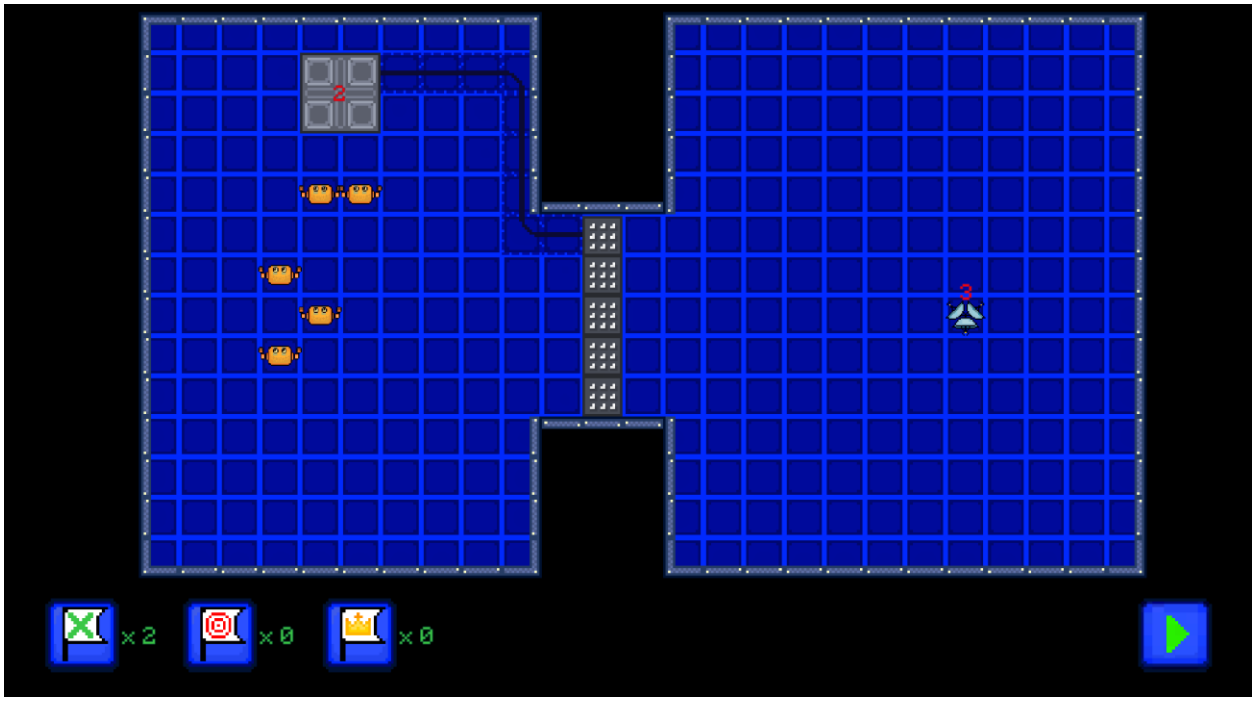

Рис. 20 Скріншот ігрового рівня

3.2.2 Ієрархія сцени

Рис. 21 Ієрархія сцени на прикладі одного рівня

Оскільки гра складається з послідовності рівнів, що представляють собою окремі сцени в Unity, було прийняте рішення будувати їх типовим чином, аби розробка нових рівнів та підтримка старих вимагала мінімальних зусиль. Типовий рівень містить такі об’єкти:

- App – батьківський об’єкт для всіх систем, що відповідає за їх ініціалізацію та виконує реалізацію шаблону проектування Dependency Injection[28].
- Cameras – пустий об’єкт, що містить камеру та все, що потрібно для її роботи.

- World – об’єкт, що є батьківським для всі об’єктів, що існують в ігровому світі. Містить компонент Objects Container, який зберігає посилання на важливі об’єкти на сцені (наприклад, ціль для камери, UI компоненти, тощо).

Серед важливих об’єктів ігрового світу, тобто таких, що є дочірніми для World можна виділити такі:

- Grid – сітка, на якій побудована геометрія рівню (підлога, стіни, тощо).
- Camera target – ціль, за якою слідує камера (центр її поля зору)
- Canvas – графічний компонент, що використовується для відображення користувацького інтерфейсу (кнопки, меню паузи, тощо).
- Flag Preview – об’єкт, що використовується для відображення, куди поставиться флаг для віддавання наказу та чи підходяще це місце для нього.
- Minion – об’єкти міньйонів (роботів, яким віддає накази гравець), містять у собі візуальне представлення та відповідні компоненти (Collider, RigidBody, Steering Manager, тощо). Міньйонів може бути різна кількість в залежності від потреб конкретного рівня.
- Level Ending Trigger – тригер кінця рівня, до якого має наблизитись відповідна кількість міньйонів, аби рівень був успішно завершений.
- World Objects – пустий батьківський компонент для більш зручного структурування об’єктів на рівні.

3.2.3 Опис основних систем та компонентів

Деякі компоненти є відповідальними за загальну логіку гри, а не за окремий об’єкт, такі компоненти названо системами, та вони винесені на об’єкти, дочірні до App, який відповідає за їх ініціалізацію.

Список основних систем:

- Input System – система, відповідальна за зчитування натискань клавіш, рухів та натиснень клавіш миші та передавання їх іншим системам, що повинні на них реагувати.
- Unit Selection System – система, відповідальна за виділення міньйонів для подальшого віддання їм наказів, позначення міньйонів як виділених та відображення рамки для виділення міньйонів на екрані.
- Flag System – система, що відповідальна за створення та розміщення прапорів, що символізують накази, відповідно до дій гравця, та додавання наказів обраним міньйонам.
- Game State System – система, що відповідальна за керування та переключення між ігровими станами (стадії планування та виконання наказів).

Найважливішими та найскладнішими компонентами в грі, які власне і використовують Steering Manager є контролери міньйонів та ворогів, тому розглянемо їх більш детально.

Контролер міньйонів описує логіку поведінки відповідно до відданих наказів та стану навколишнього середовища та основна логіка його роботи зосереджена в методі Update, який викликається рушієм кожен кадр. Алгоритм оновлення стану міньйона виглядає наступним чином:

1. Якщо міньйон знаходиться в неактивному стані (гра знаходиться в стані планування), оновлення завершується.

2. Застосовується поведінка уникання колізій (нічого не робить, якщо немає перешкод в полі зору)
3. Якщо є цілі для слідування (шлях ще не завершений):
    a. Застосовується поведінка пошуку поточної цілі.
    b. Якщо міньйон достатньо наблизився до цілі, обирається наступна ціль слідування із списку-шляху
4. Якщо є статичні цілі для уникання, відносно кожного з них застосовується поведінка втечі.
5. Якщо є вороги для уникання, відносно кожного з них застосовується поведінка уникання.
6. Якщо є призначений міньйон-лідер, відносно нього застосовується поведінка слідування за лідером.

Вороги є дещо простішими за міньйонів, адже вони лише переключаються між патрулюванням шляху та переслідуванням міньйонів та їх алгоритм оновлення виглядає так:

1. Якщо ворог знаходиться в неактивному стані (гра знаходиться в стані планування), оновлення завершується.
2. Якщо є ціль для переслідування:
    a. Застосовується поведінка переслідування цілі.
    b. Якщо ціль переслідування знаходиться ближче, ніж дистанція атаки, починається анімація атаки.
    c. Оновлення завершується.
3. Якщо в полі зору є цілі, які можна переслідувати, одна з них. обирається як ціль до переслідування та оновлення завершується.
4. Якщо є шлях для патрулювання:
    a. Застосовується поведінка пошуку до поточної цілі пошуку.

b. Якщо ворог достатньо наблизився до цілі пошуку, обирається наступна ціль пушку із списку-шляху.
c. Якщо дійшли до кінця шляху, шлях починається заново, або розвертається навпаки, відповідно до налаштувань ворога.

Висновки

У процесі створення програмного продукту було досліджено ігровий жанр «пазл», види керувальних поведінок та способів їх реалізації з використанням ігрового рушія Unity.

В результаті було створено гру в жанрі «пазл» під назвою «The Controller» з використанням Unity та мови C#. У грі було реалізовано такі механіки, як віддавання наказів групам міньйонів, система рівнів, інтерактивне оточення, різні фази гри та вороги. Було використано керувальні поведінки для реалізації поведінки міньйонів та ворогів, що дало змогу зробити її природньою, тонко налаштовувати та комбінувати різні види поведінок.

Реалізація природньої поведінки ігрових персонажів є невід'ємною частиною процесу створення сучасних ігрових продуктів, а керувальні поведінки (Steering Behaviors) показали гарні результати як в плані органічності та зручності використання, так і в плані швидкодії. Вибір жанру «пазл» також створив сприятливі умови для впровадження поведінок, адже він мотивує гравця ефективно та вдумливо використовувати надані розробниками інструменти та дозволяє створити спектр різноманітних сценаріїв які гравцю треба подолати за допомогою ігрових персонажів.

Список літератури

1. History of puzzle games. *GameSpot*.
URL: https://web.archive.org/web/20100204081152/http://www.gamespot.com/features/vgs/universal/puzzle_hs/
2. Heiankyo Alien – Hardcore Gaming 101. *Hardcore Gaming 101*.
URL: http://www.hardcoregaming101.net/heiankyo-alien/
3. 8-Bit Product Reviews: Gyruss, Spelunker, Football, Keystone Kapers, Puzzle Panic, Super Mailer Plus, Music Construction Set, Mask Of The Sun, One On One, Frogger II: Threedeep, Beach-Head. *Classic Computer Magazine Archive*.
URL: https://www.atarimagazines.com/v3n8/productreviews.html
4. The Editors of Encyclopaedia Britannica. Tetris | Puzzle Game, Block Shapes, Soviet Origin. Encyclopedia Britannica. 2008.
URL: https://www.britannica.com/topic/Tetris
5. Cobbett R. The most successful game ever: a history of Minesweeper. *TechRadar*.
URL: https://www.techradar.com/news/gaming/the-most-successful-game-ever-a-history-of-minesweeper-596504
6. 2. Fulton J., Fulton S. The Essential Guide to Flash Games. Berkeley, CA : Apress, 2010.
URL: https://doi.org/10.1007/978-1-4302-2615-4
7. Gaudiosi J. Rovio Execs Explain What Angry Birds Toons Channel Opens Up To Its 1.7 Billion Gamers. *Forbes*.
URL: https://www.forbes.com/sites/johngaudiosi/2013/03/11/rovio-execs-explain-what-angry-birds-toons-channel-opens-up-to-its-1-7-billion-gamers/?sh=12e4fa5857f2

8. Rousseau J. King's Candy Crush Saga hits $20bn in lifetime revenue. *GamesIndustry.biz*.
URL: https://www.gamesindustry.biz/kings-candy-crush-saga-hits-20bn-in-lifetime-revenue
9. Gamasutra - Features - Games Demystified: Portal. *Wayback Machine*.
URL: https://web.archive.org/web/20100807083401/http://www.gamasutra.com/view/feature/3770/games_demystified_portal.php
10. LeBlanc W. The Entropy Centre Review - Reverse-Engineered Ingenuity - Game Informer. *Game Informer*.
URL: https://www.gameinformer.com/review/the-entropy-centre/reverse-engineered-ingenuity
11. Reynolds, C. W. (1999) Steering Behaviors For Autonomous Characters, in the proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California.
URL: https://www.red3d.com/cwr/steer/gdc99/
12. Bevilacqua F. Understanding Steering Behaviors: Seek | Envato Tuts+. *Code Envato Tuts+*.
URL: https://code.tutsplus.com/understanding-steering-behaviors-seek--gamedev-849t
13. Bevilacqua F. Understanding Steering Behaviors: Flee and Arrival | Envato Tuts+. *Code Envato Tuts+*.
URL: https://code.tutsplus.com/understanding-steering-behaviors-flee-and-arrival--gamedev-1303t
14. Bevilacqua F. Understanding Steering Behaviors: Pursuit and Evade | Envato Tuts+. *Code Envato Tuts+*.

URL: https://code.tutsplus.com/understanding-steering-behaviors-pursuit-and-evade--gamedev-2946t

15. Bevilacqua F. Understanding Steering Behaviors: Collision Avoidance | Envato Tuts+. *Code Envato Tuts+*.

URL: https://code.tutsplus.com/understanding-steering-behaviors-collision-avoidance--gamedev-7777t

16. Bevilacqua F. Understanding Steering Behaviors: Path Following | Envato Tuts+. *Code Envato Tuts+*.

URL: https://code.tutsplus.com/understanding-steering-behaviors-path-following--gamedev-8769t

17. Bevilacqua F. Understanding Steering Behaviors: Leader Following | Envato Tuts+. *Game Development Envato Tuts+*.

URL: https://gamedevelopment.tutsplus.com/understanding-steering-behaviors-leader-following--gamedev-10810t?_ga=2.46153376.687836658.1715179838-803940246.1710100825

18. Pemmaraju V. 3 Simple Rules of Flocking Behaviors: Alignment, Cohesion, and Separation | Envato Tuts+. *Code Envato Tuts+*.

URL: https://code.tutsplus.com/3-simple-rules-of-flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444t

19. Dealessandri M., Calvin A. What is the best game engine: is Unity right for you?. *GamesIndustry.biz*.

URL: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

20. Scripting in Unity for experienced C# & C++ programmers. *Unity*.

URL: https://unity.com/how-to/programming-unity

21. Unity - Scripting API: GameObject. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/GameObject.html

22. Unity - Manual: Introduction to components. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/Manual/Components.html

23. Unity - Manual: Creating and Using Scripts. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

24. Unity - Scripting API: MonoBehaviour. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

25. Unity - Manual: Prefabs. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/Manual/Prefabs.html

26. Unity - Scripting API: Rigidbody. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/Rigidbody.html

27. Unity - Scripting API: MonoBehaviour.LateUpdate(). *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html

28. Dependency Injection is Loose Coupling. *ploeh blog*.
URL: https://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/