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

# Використання штучного інтелекту в іграх

**Текстова частина до курсової роботи**
**за спеціальністю „Інженерія програмного забезпечення” - 121**

Керівник курсової роботи:

д.т.н., доцент, Глибовець А. М.

_____(підпис)

“_____” _____ 2021 р.

Виконав студент

Бойко Д. Р.

_____(підпис)

“_____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

доцент _____

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Бойко Данилу Романовичу факультету інформатики 3-го курсу

**ТЕМА:** Використання штучного інтелекту в іграх

**Вихідні дані:** Штучний інтелект для гри в Тетріс та отримання найкращого результату

**Зміст ТЧ до курсової роботи:**

Індивідуальне завдання

Вступ

Розділ 1: Історія застосування штучного інтелекту в іграх

Розділ 2: Способи використання штучного інтелекту в іграх

Розділ 3: Розробка штучного інтелекту для гри Тетріс

Висновки

Список використаної літератури

Дата видачі „___” _____ 2020 р.

Керівник _____ (підпис)

Завдання отримав _____ (підпис)

**Тема**: Розробка сервісу збереження та надання доступу до документів університету

**Календарний план виконання роботи:**

| № | Назва етапу | Термін виконання | Примітки |
|---|---|---|---|
| **1** | Отримання теми курсової роботи | 10.11.2020 | |
| **2** | Огляд технічної літератури за темою роботи | 30.11.2020 | |
| **3** | Встановлення необхідного програмного забезпечення | 02.12.2020 | |
| **5** | Написання основної частини курсової роботи | 12.01.2021 | |
| **6** | Перегляд змісту роботи з керівником | 14.05.2021 | |
| **7** | Внесення змін до курсової роботи відповідно до зауважень наукового керівника | 15.05.2021 | |
| **8** | Створення слайдів для доповіді та написання доповіді | 16.05.2021 | |
| **9** | Захист роботи | 24.05.2021 | |

Студент Бойко Д. Р.

Керівник Глибовець А. М.

“_____”_____

## Зміст

## Анотація

У процесі написання курсової роботи було розглянуто велика кількість ігор, які використовують штучний інтелект, з 1948 року по сьогодення. Також був розроблений штучний інтелект для гри в Тетріс на мові C++.

# Вступ

**Актуальність теми**: з кожним роком індустрія відеоігор стрімко зростає, а геймери стають все більш вибагливими, тому розробники прикладають великі зусилля на розробку та вдосконалення штучного інтелекту, щоб покращити ігровий досвід.

**Мета і завдання дослідження**. Дослідити та описати процес розвитку штучного інтелекту в сфері відеоігор.

**Об'єкт дослідження.** Процес розвитку штучного інтелекту в сфері відеоігор.

**Методи дослідження.** Дослідження випущених ігор, які використовують штучний інтелект.

**Джерела дослідження.** Власний досвід, відеоматеріали IGM (YouTube), інтернет-ресурс stopgame, інтернет-ресурс habr, інші інтернет-ресурси

# РОЗДІЛ 1 Історія застосування штучного інтелекту в іграх

## 1.1 Перші спроби використання штучного інтелекту в іграх

Алан Тюрінг та Девід Чамперноун у 1948 році розробили шахову програму під назвою Turochamp. Вона здатна зіграти цілу партію в шахи проти гравця-людини на низькому рівні гри, обчислюючи всі потенційні ходи та всі потенційні ходи гравця. Комп'ютер призначає певну кількість балів кожному стану гри та обирає наступний хід з найвищим середнім значенням [14].

На жаль гра не була завершена, оскільки їх алгоритм був надто складним, щоб його можна було запустити на комп'ютерах того часу, таких як Automatic Computing Engine [2]. Влітку 1952 року Тюрінг зіграв матч проти інформатика Аліка Гленні, використовуючи програму, Алан виконував програму поетапно [14].

Turochamp імітує гру в шахи проти гравця, приймаючи ходи гравця як вхідні дані та виводячи їх у відповідь. Алгоритм програми використовує евристику з двох ходів для визначення найкращого ходу: він обчислює всі потенційні ходи, які вона може зробити, потім усі відповіді потенційного гравця, які можна було б зробити по черзі, присвоює бальне значення кожному отриманому стану, тоді робить хід із найбільшим середнім отриманим балом. Бали визначаються на основі кількох критеріїв, таких як мобільність кожної фігури, безпека кожної фігури, загроза отримати мат, вартість фігури гравця, якщо вона взята, та кілька інших факторів. Різні ходи отримують різні бальні значення; наприклад, за взяття дами дається 10 очок, а пішака лише одне очко, за чек - очко або половина очка на основі макета дошки. На думку Чамперноуна, алгоритм в основному розроблений навколо рішення взяти фрагмент чи ні; за словами Тюрінга, отриманий в результаті геймплей створює гру в шахи низького

рівня, яку він вважав спів мірною з його само описаним середнім рівнем навичок у грі [14].

## 1.2 Перша комерційна гра з штучним інтелектом Pac-Man

Pac-Man - це аркадна гра в лабіринті, розроблена і випущена Namco в 1980 році. Оригінальний японський титул Puck Man був змінений на Pac-Man для міжнародних випусків як запобіжний засіб проти зіпсування аркадних машин шляхом зміни P на F. Гравець керує Pac-Man, який повинен з'їсти всі крапки всередині замкнутого лабіринту, уникаючи чотирьох кольорових привидів. Вживання великих миготливих крапок, які називаються «енергетизаторами», призводить до того, що привиди стають синіми, що дозволяє Pac-Man їсти їх за бонусні бали. Pac-Man мав широкий критичний та комерційний успіх, і він має стійкий комерційний та культурний спадок. Гра вважається важливою та впливовою, і її зазвичай зараховують до найвидатніших відеоігор усіх часів [10].

Екран гри займає собою лабіринт, коридори якого заповнені точками. Завдання гравця - керуючи Пакманом, з'їсти всі крапки в лабіринті, уникаючи зустрічі з привидами, які ганяються за героєм. На початку кожного рівня приведи перебувають в спеціальній прямокутній кімнаті в середині рівня, з якої вони з часом звільняються. Якщо привид доторкнеться до Пакмана, то його життя втрачається, приведи і Пакман повертаються на вихідну позицію, але при цьому прогрес зібраних точок зберігається. Після поїдання всіх точок починається новий рівень в тому ж лабіринті. З боків лабіринту знаходяться два входи в один тунель, при входженні в який Пакман і примари виходять з іншого боку лабіринту[10].

Всього в лабіринті знаходяться 240 маленьких точок і 4 великі, відомі як енерджайзери (англ. Energizer). За поживу маленької точки дається 10 очок, а за

поживу енерджайзера - 50. При з'їденні Пакманом енерджайзера на ранніх рівнях примари в лабіринті на короткий час входять в режим переляку, різко змінюють напрямок руху і перефарбовуються в синій колір. За цей час Пакман здатний з'їсти примар за допомогою зіткнення з ними. За поживу першого привида після отримання енерджайзера дається 200 очок. За поживу кожного наступного привиди при ефекті того ж енерджайзера дається в два рази більше: 400, 800 і 1600 відповідно. Однак з 19-го рівня примари перестають бути синіми і більше не можуть бути з'їдені[10].

До 21-го рівня швидкість Пакмана при звичайному русі вище швидкості привидів, але з 21-го примари в режимі погоні рухаються швидше головного героя. Поїдання точок уповільнює Пакмана приблизно на 10% від його швидкості, що дозволяє привидам на всіх рівнях наздогнати героя. Після поїдання енерджайзера, поки примари налякані, швидкість Пакмана збільшується, а рух привидів сповільнюється [10].

Привиди мають три різні запрограмовані моделі поведінки: переслідування, розсіювання і переляк. Для руху привидів в перших двох режимах гра використовує поділ ігрового екрану на квадрати. В режимах розсіювання і переслідування у привидів є певний квадрат, який вони намагаються досягти. При розсіюванні привиди прагнуть до квадратах, які знаходяться за межами лабіринту ближче кутів. Наприклад, червоний привид прагне до правого верхнього кута. У режимі переляку привиди не мають цільової точки і вибирають поворот в лабіринті за допомогою генератора псевдовипадкових чисел. Розрахунок цільової точки при переслідуванні винятковий для кожного привиду, що надає їм унікальне поведінку [10]:

Червоний привид Шедоу. У режимі переслідування використовує ту клітку, в якій знаходиться Пакман. Цей привид, на відміну від інших привидів,

збільшує свою швидкість переслідування щодо первісної двічі за рівень в залежності від кількості з'їдених точок. Якщо точок залишилося мало, то він змінює цільову клітку в режимі розсіювання на квадрат, в якому знаходиться Пакман, і так ганяється за героєм в двох режимах [10].

Рожевий привид Спіді. В якості цілі при переслідуванні використовує точку, що знаходиться на чотири клітини попереду Пакмана. Однак через помилки переповнення при русі Пакмана вгору використовує в якості цілі квадрат, що знаходиться на чотири клітини вгору і на чотири вліво від Пакмана [10].

Блакитний привид Башфул. Використовує найскладніший алгоритм переслідування: для визначення напрямку руху будується відрізок, один з кінців якого визначається власне положенням, а середина знаходиться на 2 клітини перед Пакманом. Другий кінець відрізка - шукана цільова точка. Отриману точку складно передбачити, тому Інки вважається найнебезпечнішим привидом. Через помилки переповнення, аналогічної в поведінці рожевого привиду, під час руху Пакмана вгору цільова клітина Інки це дві клітини вгору і дві вліво від Пакмана [10].

Помаранчевий привид Поки. Якщо він знаходиться далі 8 клітин від Пакмана, то він використовує в якості мети самого Пакмана. Якщо ж Пакман ближче 8 клітин, то привид прагне до лівого нижнього кута, як при розсіюванні [10].

Поки привиди перебувають в одному з режимів, змінити напрямок руху вони можуть тільки під час повороту в лабіринті, але в момент зміни одного режиму на інший привиди здатні змінити напрямок руху на протилежний. Виняток становить перехід від переляку до інших режимів. З закритої зони на початку рівня вони вибираються, перебуваючи в режимі розсіювання, пізніше

змінюючи його на переслідування. Потім протягом раунду привиди можуть почати розсіюватися ще три рази. При втраті Пакманом життя лічильник розсіювання скидається. Залежно від рівня час цього режиму може тривати від 1/60 до 7 секунд. Час переслідування до наступного розсіювання може тривати на різних рівнях від 30 до 1037 секунд. Коли всі чотири розсіювання закінчуються, примари переслідують Пакмана без зупинки. Якщо примари перелякані, таймер інших режимів призупиняється [10].

Єдиний привид, який на початку кожного рівня або після втрати Пакманом життя стартує відразу поза зоною привидів, - червоний. За ним в строгому порядку виходять рожевий, потім блакитний і останнім помаранчевий. Протягом перших трьох рівнів привиди виходять після поїдання Пакманом певної кількості точок. Наприклад, на першому рівні рожевий привид починає гру після 30 точок, а блакитний після 60-ти. Після втрати Пакманом життя цей лічильник скидається і запускається новий, але з меншою кількістю точок. З четвертого рівня і до кінця гри всі привиди відразу починають гру після старту рівня. Є також таймер, який дозволяє привидам покинути свою зону, якщо Пакман певний час не їсть точки. Наприклад, якщо на першому рівні Пакман, що не з'ївши 30 точок, застигне на місці, рожевий привид відправиться в погоню за ним через чотири секунди після цього. Кожна з'їдена точка скидає цей таймер до нуля [10].

## 1.3 Подальший розвиток штучного інтелекту у відеоіграх

### 1.3.1 Castle Wolfenstein

Пригодницький бойовик, виконаний в двовимірній графіці з видом зверху. Керований гравцем персонаж потрапляє в полон в тилу ворога, і його беруть в заручники в нацистську фортеця Wolfenstein. В очікуванні допиту головному герою вдається звільнитися і отримати в своє розпорядження пістолет з 10 набоями і три гранати. У цей момент починається гра, де протагоністу необхідно

відшукати ворожі секретні документи, в яких відображені плани противника, і далі вибратися з замку живим [3].

Ігровий світ являє собою процедурно-згенерований лабіринт з приблизно 60 кімнат замку, в якому знаходиться безліч нацистських охоронців і штурмовиків СС. Ворогів можна усувати за допомогою наявного зброї, а в подальшому обшукувати їх, знаходячи патрони, гранати або ключі. У деяких випадках охоронець здається - якщо «в упор» направити на нього пістолет, навіть якщо той без патронів. Ігровий процес побудований таким чином, що боєприпасів для знищення всіх ворогів не вистачає, і тому більш прийнятною стратегією для гравця стає використання можливості прокрастися повз охоронців. Це зробити важко, так як ворожі персонажі патрулюють кімнати замку і реагують на шум, який може створити головний герой. У замку є скрині, які, витративши якийсь ігровий час, можна відкривати за допомогою знайдених ключів, і в них можна знайти бронежилет, уніформу і секретні документи. Крім цих предметів в скринях можуть опинитися й інші - квашена капуста, ковбаса або шнапс, - їх головний герой може їсти, але далі на ігровий процес це не впливає. Уніформа дозволяє протагоністу проходити непоміченим повз нацистів, але проти штурмовиків СС вона залишається марною. Прискорити відкривання скрині можна вистріливши в нього, але це приверне увагу охоронців в кімнаті. Крім того, гравцеві потрібно остерігатися стрільби в скрині з боєприпасами (містять патрони і гранати), так як через це вони можуть вибухнути. В замку є замкнені двері, які можуть бути відкриті знайденим ключем або за допомогою пострілу з пістолета [3].

Для проходження гри протагоністу потрібно знайти вихід з лабіринту і тим самим втекти. Якщо до цього моменту гравець знаходить секретні документи, то після втечі йому підвищують звання, і це збільшує складність подальшої гри. Далі змінюється планування замку і гра починається заново. Всього в грі 8 рівнів

складності, які визначаються званням гравця, і до тих пір, поки не буде отримано нове звання, отримати доступ до більш складним рівнями не можна [3].

Загинути гравець може як від пострілу противника, так і від вибуху власної гранати. При цьому наслідки різні: в першому випадку стан замку зберігається (скрині і охоронці залишаються такими ж, і відбувається рестарт з початкової кімнати), а в другому гра повністю закінчується (лабіринт генерується заново) [3].

### 1.3.2 Thief: The Dark Project

Гра відбувається від першої особи в 3D-середовищі. Рівні в основному не містять сценаріїв. Неігрові персонажі (NPC) можуть або залишатися нерухомими, або ходити по маршруту патрулювання, і гравці мають свободу вибору. В залежності від обраної складності вороги будуть мати різну чутливість до свого оточення. Щоб допомогти гравцям залишатися прихованими, спеціальний лічильник на екрані хедс-ап (HUD) у формі дорогоцінного каміння допомагає вказувати видимість гравця. Щоб залишатись непочутими, гравці повинні бути обережними щодо того, наскільки багато вони створюють шуму, а також по яким поверхням вони рухаються; ходьба на м'яких поверхнях, таких як килими та трава, є кращою, оскільки кроки залишаються тихими, порівняно з ходьбою по металевій підлозі та керамічній плитці, яка створює багато шуму. Гравець може використовувати шум, щоб відволікти NPC, наприклад, кинути предмет, щоб заманити їх в інше місце [13].

### 1.3.3 Half-Life

Відеогра жанру науково-фантастичного шутера від першої особи, розроблена Valve Software і видана Sierra Studios 20 листопада 1998 року для ПК [7]. Вихідний код гри доступний у вільному доступі, тому є можливість дослідити штучний інтелект більш детально. Файли мають не дуже зрозумілі назви, а

реалізація класів C ++ розкидана по декількох файлах, з назв яких майже нічого не можна зрозуміти. У файлах schedule.h та schedule.cpp знаходиться дуже проста система, яка керується цілями. Вона складається з декількох рівнів завдань, які можна процедурно об'єднувати. Завдання - це короткі роздроблені поведінки, які мають конкретне призначення. Наприклад, більшість акторів Half-Life підтримує наступні завдання: TASK_WALK_PATH, TASK_CROUCH, TASK_STAND, TASK_GUARD, TASK_STEP_FORWARD, TASK_DODGE_RIGHT, TASK_FIND_COVER_FROM_ENEMY, TASK_EAT, TASK_STOP_MOVING, TASK_TURN_LEFT, TASK_REMEMBER. Вони визначаються як перерахування в файлі заголовка і реалізуються як методи C ++. Умови використовуються для вираження ситуації актора в світі. Оскільки логіка задана жорстко, умови можна виразити дуже компактно, як бітові поля, але в такому випадку умов може бути не більше 32. Наприклад, умовами є COND_NO_AMMO_LOADED, COND_SEE_HATE, COND_SEE_FEAR, COND_SEE_DISLIKE, COND_ENEMY_OCCLUDED, COND_ENEMY_TOOFAR, COND_HEAVY_DAMAGE, COND_CAN_MELEE_ATTACK2, COND_ENEMY_FACING_ME. План складається з серії завдань (з довільними параметрами) і враховує бітове поле умов, щоб визначити, коли план непридатний. Для зручності налагодження об'єкти планів мають імена. Цілі знаходяться на більш високому рівні і складаються з планів. Логіка цілі може при необхідності вибирати план на підставі неуспішного завдання і поточного контексту. Приклади цілей з Half-Life: GOAL_ATTACK_ENEMY, GOAL_MOVE, GOAL_TAKE_COVER, GOAL_MOVE_TARGET і GOAL_EAT [28].

На практиці всі ці плани і завдання пов'язані один з одним в структуру, схожу на скінчений автомат. На верхньому рівні для поновлення штучного інтелекту викликається функція в monsterstate.cpp:

```
void CBaseMonster :: RunAI (void);
```

Вона, в свою чергу, викликає перевизначені функції, що відповідають за перевірку чи можливо застосувати поточний план за допомогою MaintainSchedule () та вибір нових за допомогою GetSchedule (). Їх можна змінювати в залежності від потреб за допомогою породжених класів.

На нижньому рівні функції StartTask () і RunTask () реалізують логіку для кожного з ідентифікаторів завдань, визначених в конструкції enum. Вони реалізовані в класах, теж успадкованих з CBaseMonster. В результаті це багато в чому виглядає як скінчений автомат, реалізований як конструкція switch.

```
void CScientist :: RunTask( Task_t *pTask )
{
 switch ( pTask->iTask )
 {
 case TASK_RUN_PATH_SCARED:
  if ( MovementIsComplete() )
   TaskComplete();
  if ( RANDOM_LONG(0,31) < 8 )
   Scream();
  break;
 case TASK_MOVE_TO_TARGET_RANGE_SCARED:
  break;
 case TASK_HEAL:
  if ( m_fSequenceFinished )
  {
   TaskComplete();
  }
  else
```

```
        {
         if ( TargetDistance() > 90 )
           TaskComplete();
         pev->ideal_yaw = UTIL_VecToYaw();
         ChangeYaw( pev->yaw_speed );
        }
        break;

      default:
        CTalkMonster::RunTask( pTask );
        break;
      }
    }
```

При існуючій реалізації набагато простіше при необхідності використовувати логіку одного об'єкта в іншому [28].

Реалізація сенсорної системи. У стандартного monster.h та monster.cpp є код, що дає всім акторам зір, нюх і слух.

```
void CBaseMonster :: Look ( int iDistance );
```

Функція зору перевіряє різні прапори, такі як SF_MONSTER_PRISONER і SF_MONSTER_WAIT_TILL_SEEN. У рівнянні також враховуються такі параметри, як область видимості і кут огляду [28].

```
CSound* CBaseMonster :: PBestSound ( void );
```

Код слуху та нюху працює схожим чином, тільки використовує події звуку. Зберігається список об'єктів, які потребують уваги монстрів, а сенсорна система вибирає для фокусування кращий з них [28].

### 1.3.4 F.E.A.R. First Encounter Assault Recon

Відеогра 2005 року, шутер від першої особи з елементами survival horror. Розроблена Monolith Productions, та видана Vivendi Universal Games і Warner Bros. Interactive Entertainment 17 жовтня 2005 року [6].

F.E.A.R. (далі гра) використовує ціле-орієнтоване планування дій (Goal-Oriented Action Planning, GOAP). Цей спосіб заснований на технології автоматизованого планування - процесі, при якому система намагається визначити послідовність дій для досягнення віддаленої мети. Щоб зробити це, потрібна модель, яка повідомляє всю актуальну інформацію про ігровий світ - факти. Сукупність фактів, що відносяться до одного моменту часу, формують стан світу [1].

Наприклад, уявімо зачинені двері. Щоб NPC міг відкрити їх, потрібна інформація про те, закрита вона чи ні. На даний момент стан цього світу – одні зачинені двері. Так як ця інформація доступна, можна запустити дію, яка дозволить відкрити їх. Кожна дія зазвичай розбивається на три частини: об'єкти, які беруть участь в дії (в даному випадку двері), передумови дії - перелік фактів про світ, і ефекти, які показують, як світ зміниться в результаті завершення дії. Щоб NPC відкрив двері, деякі передумови повинні бути істинними. Наприклад, двері були зачинені, а сам персонаж повинен перебувати в кімнаті з цими дверима. Результатом дії стануть відкриті двері [1].

Тепер припустимо, що NPC потрібно пройти з однієї кімнати в іншу, але між ними є зачинені двері. Тоді в системі планування має бути прописана умова, що персонаж не може просто так пройти через стіну, а переміщення відбувається через двері, які з'єднують кімнати. Тоді система створить план, в якому зазначить,

що персонаж повинен відкрити двері, щоб перейти в нове приміщення. Після виконання дії стан світу зміниться - тепер NPC буде знаходитися в другій кімнаті, а двері будуть відчинені. Для планування потрібно з'ясувати, що робити, і в якому порядку. Але проблема цього підходу полягає в тому, що для виконання послідовності дій потрібно враховувати величезну кількість додаткових факторів. Наприклад, в цьому прикладі послідовність буде працювати тільки в тому випадку, якщо NPC стоїть поруч з дверима. В іншому випадку доведеться враховувати розташування дверей і різних перешкод, які є на шляху. Додаткові умови значно ускладнюють застосування цього підходу на практиці [1].

Цей підхід (GOAP) використовує скінченні автомати (машина Тюрінга) - систему, в якій штучний інтелект існує в межах одного стану поведінки, а потім переходить в інший стан, якщо відбувається певна подія. В системі GOAP є всього лише три:

1. переміщення в певну точку в ігровому світі;
2. відтворення анімації;
3. взаємодія зі смарт-об'єктами - інтерактивними ігровими предметами.

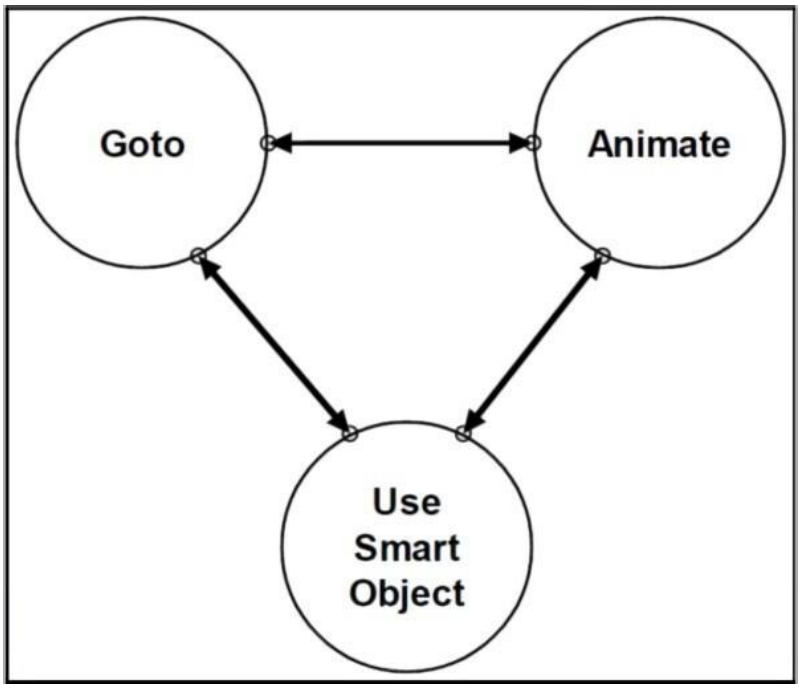



*Рисунок 1.1 [1]*

Goto - відтворення анімації руху (біг, ходьба, стрибок).

Animate - відтворення інших анімацій (стрільби, реакції на героя і так далі).

Use Smart Object - анімація взаємодії з предметами (відкрити двері, включити світло і так далі)

Якщо спростити роль противника в грі, то все, що він робить - це програє послідовність декількох анімацій в певних умовах. Коли ця послідовність працює правильно, то виникає ілюзія усвідомлених дій штучного інтелекту. Отже, кожне з цих трьох станів зводиться до відтворення анімації [1].

Таким чином, виходять не скінченні автомати, в яких кожна можлива поведінка вважається як окремий стан, а більш компактна система, яка більше спирається на дані. Система GOAP створює плани за допомогою алгоритму A-star (евристичний алгоритм пошуку, який використовується для пошуку найкоротшого шляху між двома вершинами графу з додатними вагами ребер), який дозволяє знайти найбільш оптимальну послідовність дій через порівнювання «вартості» варіантів. Потім дії переводяться в конкретні стани - передаються дані для переміщення, анімації та взаємодії зі смарт-об'єктами [1].

У F.E.A.R. у кожного персонажа, яким керує штучний інтелекет, є свої цілі. Система використовує ці цілі для планування дій в умовах бою. Без мети NPC нічого не будуть робити. Всього у противників є близько 70 цілей - система дозволяє змінювати їх пріоритет в залежності від умов бою. Наприклад, якщо солдат не знає про те, що гравець знаходиться поруч, то пріоритет мети «KillEnemy» буде дорівнює нулю. Відповідно, якщо при патрулюванні ворог помітить користувача, то пріоритети відразу ж зміняться. Щоб планувальник міг підлаштуватися під ситуацію, у нього повинен бути набір дій, з яких він буде вибирати. У грі є 120 дій, які включають в себе абсолютно все: від різних варіантів атаки до idle-анімації (анімація очікування ). При цьому не у всіх персонажів є доступ до всіх дій. Гейм дизайнери призначали конкретні дії кожному типу супротивників, завдяки чому розробники змогли визначати, наскільки універсальним вийде кожен персонаж при вирішенні своїх завдань [1].

Єдина велика проблема, що виникає при реалізації плану, полягає в тому, що умови можуть змінюватися прямо по ходу його виконання. Наприклад, якщо персонаж піде відчиняти двері, але його випередить хтось інший, то план виявиться недійсним. З цією проблемою стикається переважна більшість систем планування. Персонажі не можуть підлаштуватися під інших NPC. Жоден з ворожих персонажів  в F.E.A.R. не знає про існування своїх товаришів, а їх злагоджені спільні дії - це лише ілюзія, яка виникає, коли кожен з них виконує свою місію. Якщо якийсь персонаж порушує план іншого NPC, то повинен з'явитися новий план, що враховує оновлені умови, або змінити мету. Це досягається трьома різними способам:

- Коли з'являється чіткий план, система порівнює його зі станом світу, щоб упевнитися, що мета буде виконана [1].
- У кожної мети є функція ReplanRequired, яка постійно перевіряє, чи варто відмовитися від поточного плану і сформувати новий [1].

- Кожна дія повторно перевіряється під час виконання. Разом з цим проходить перевірка передумов і наслідків. Якщо з'являється невідповідність, то система заново формує план [1].

Коли дія спрацьовує правильно, воно повідомляє кінцевому автомату, в яке з трьох станів слід перейти, а також передає всі дані, необхідні для переходу в стан. Наприклад, якщо солдату потрібно перезарядити зброю, то стан переходить в «Animate», і він отримує всю необхідну інформацію для здійснення дії. Якщо план виконаний, то у NPC з'являється новий. Постійний перехід від одного плану до іншого підтримує темп гри, а штучний інтелект завжди знає, що потрібно робити в наступну мить. При цьому плани в F.E.A.R. досить короткі і складаються з однієї-двох дій. Лише зрідка вони збільшуються до трьох-чотирьох дій.

Згодом розробники з'ясували, що процес постійного перепланування іноді негативно впливає на продуктивність. Вся справа в тому, що на деяких рівнях є щури, які використовують таку ж систему цільових перевірок орієнтованого планування дій. Як правило, їх плани досить прості - вони пересуваються по світу і намагаються втекти від гравця, якщо він виявляється поблизу. Коли вони тікають, вони постійно будують нові плани. Щури продовжують це робити навіть тоді, коли користувач знаходиться на іншому краю карти, тому що вони не враховують, наскільки далеко втекли. Таким чином, гравець може зустріти пару щурів в перші секунди рівня, і навіть через 20 хвилин вони все ще будуть бігати, постійно створюючи нові плани.

### 1.3.5 The Elder Scrolls: Oblivion

Рольова відеогра для одного гравця, розроблена Bethesda Game Studios [12]. В грі використовується Radiant AI [25], технологія, яка дозволяє неігровим персонажам (NPC) приймати рішення і робить їх поведінку більш складним.

Цікавий факт, згодом розробникам Oblivion довелося практично відразу її пом'якшити, оскільки вона робила персонажів в грі занадто розумними. Ця технологія забезпечує неігрових персонажів процедурно-генерованими скриптами, щоб вони могли виконувати свої завдання в грі. Це означає, що неігрові персонажі в грі запрограмовані на те, щоб слідувати свого плану. Наприклад, в цій грі більшість неігрових персонажів запрограмовані, щоб їсти, спати і працювати в певний час доби, щоб зробити світ більш реальним і динамічним [31].

Технологія надає неігрових персонажів свободу інтерпретувати ці команди і інструкції іншими способами, в залежності від того, як вони запрограмовані. Наприклад, в грі є невидимий атрибут «відповідальність», який диктує, наскільки дотримується законів той чи інший персонаж. Чим нижче цей показник, тим з більшою ймовірністю той чи інший персонаж буде ігнорувати злочини, що здійснюються гравцем. Крім того, неігрові персонажі з вкрай низьким рівнем відповідальності іноді будуть самі скоювати злочини, роблячи такі речі, як крадіжка їжі, якщо вони голодні і не мають запасів продуктів в своєму інвентарі. Також є наступні показники: агресивність, довіра та енергія. Проте існують певні шаблони поведінки, які не можна змінити [31].

Шаблон супроводу. NPC стежить за тим, щоб головний герой не відставав. Якщо відстає - NPC зупиняється, робить нетерплячі жести або повертається. Критична дистанція визначається двома змінними - одна для вулиці, інша для будинку [31].

Шаблон харчування. Щоб поїсти, NPC повинен мати при собі їжу. Якщо її немає, NPC намагається отримати в різні способи: купити, вкрасти, підібрати з землі або з трупа. Ті, у кого параметр відповідальності вище 30, вбивати за їжу будуть лише звірів. Якщо їжу знайти неможливо, персонаж не вмирає, а просто

починає ходити туди-сюди, час від часу перевіряючи, чи немає поблизу чогось, що можна з’їсти [31].

Сон. NPC йде в задану точку (зазвичай це його власна ліжко) і лягає спати. Якщо свого ліжка під рукою немає, NPC ляже в будь-яке доступне ліжко (а безвідповідальні можуть ще і в чуже). Якщо ліжка немає взагалі, персонаж почне безцільно блукати, час від часу намагаючись відшукати її [31].

Мета. NPC йде до мети до відстані, на якому він зможе її дістати, а потім активує. Якщо мета - звір або бандит, починається атака і включається бойовий штучний інтелект. Якщо мета - мирний NPC, то починається розмова. Якщо мета - об'єкт, він підбирається. Контейнер - відкривається. Двері - відкривається. Якщо мета - диванчик або стілець, NPC сяде на нього [31].

Втеча. NPC тікає від мети. Якщо мети немає, персонаж просто біжить, а потім почне зображати сильний переляк. Так він і буде стояти, поки не зголоднієє, не захоче спати і так далі [31].

Прогулянка. NPC гуляє в певному радіусі, час від часу намагаючись заговорити з тими, хто проходить поруч. Якщо поблизу є стілець, персонаж може сісти на нього [31].

Під час розробки гри, є можливість задати кожному NPC певний набір шаблонів та умов, при яких ці шаблони виконуються. Найпростіші персонажі мають три або чотири шаблони (їсти, гуляти, спати). Більше складні можуть виконувати певний розклад – йти на роботу, в бар, розмовляти та інше [31].

Основні недоліки цієї технології. Інтелект персонажів сильно обмежений, адже усі шаблони задані, їх не можна змінити чи створити нові. Розробник мав обмежити здібності технології, оскільки вона могла створювати перешкоди під час гри. Уявимо ситуацію, де звичайний торговець їжею попросив головного

героя принести 10 кілограм м'яса вовка (не знаю навіщо йому воно), коли головний герой повертається, щоб закінчити завдання, і знаходить тіло торговця. Адже в місті є голодні персонажі з низьким рівнем відповідальності, тому воно вбили та пограбували торговця, щоб отримати їжу. Виходить, що головний герой витратив час і не може отримати винагороду. Також враження гравців від такої ситуації будуть поганими, що шкодить репутації гри [31].

### 1.3.6 Left 4 Dead

Багатокористувацька відеогра в жанрах шутера від першої особи та survival horror, розроблена студією Turtle Rock, що належить Valve Corporation. Штучний інтелект в грі називається «Режисер» [8].

Штучний інтелект в даній грі є унікальним. Він слідкує за діями гравців та підлаштовую навколишній світ. «Режисер» розміщує ворогів в залежності від стану і розташування гравців, створюючи новий досвід при кожному проходженні. За допомогою візуальних ефектів, динамічної музики і характеру спілкування гравців «Режисер» створює певний настрій і напругу. Крім того, «Режисер» несе відповідальність за появу аптечок, комплектів покращень для зброї, боєприпасів і зброї, а також особливих заражених, наприклад відьми або танка [16].

Звичайних ворогів «Режисер» розміщає по карті таким чином, щоб їх не можливо було уникнути. Найбільша кількість буде на основному шляху. Спеціальні вороги (міні-боси) з’являються виключно в певних місцях. Кількість ворогів може змінюватися в залежності від успіхів гравців. Якщо гравці досвідчені, для них буде більша кількість. Для новачків – відповідно менше. Якщо гравці раз за разом не можуть пройти певний рівень, кількість ворогів буде зменшуватися [16].

Поява предметів. Для усіх предметів, які мають з’явитися на карті підготовлені спеціальні місця. Штучний інтелект не може змінювати їх розташування, однак може вирішувати, що саме має з’явитися. Для гравців з низьким рівнем здоров’я буде аптечка, інакше – зброя або щось інше. Є певні предмети, які з’являються незалежно від дії «Режисера» [16].

Фази. Під час гри «Режисер» проходить через ряд певних фаз, що залежать від стану тих, що вижили. Ця механіка додає в гру динамічність, дозволяючи гравцям перепочити або навпаки зустріти сильний опір. Досвідчені гравці можуть передбачати фази «Режисера» і діяти відповідно [16].

# РОЗДІЛ 2 Способи використання штучного інтелекту в іграх

## 2.1 Імітація суперника

Імітація суперника є основним використанням штучного інтелекту, адже кількість гравців в однокористувацьких іграх значно більша ніж в багатокористувацьких. Рівень графіки в останніх релізах досяг майже еталонного рівня, тому розробники ігор змагаються у рівні штучного інтелекту, щоб заволікти більшу кількість гравців, ніж у конкурентів. Основний жанр, який потребую гарного суперника для гравця, це шутери. З першого погляду в таких іграх не потрібно мати складний штучний інтелект. Усі ворожі NPC мають просто з’явитися, колі потрібно, стріляти, куди потрібно, та помирати від гравця. Що необхідно для гри обирають розробники. Такі ігри як Battlefield або Call of Duty хочуть створити ідеальні умови для гравця. Вороги не будуть квапитися, щоб перемогти гравця, адже це погано впливає на In-game-experience, тому в таких іграх штучний інтелект суперників майже відсутній.

### 2.1.1 Бажання вижити

Під час бойових дій кожний учасник в першу чергу має піклуватися про власне життя. Використання укриттів для власної безпеки, використання гранат, щоб змусити гравця покинути безпечну позицію, стріляти по-самолійські (вид стрільби із укриття, коли зброю висовують з-за кута та стріляють не дивлячись, малоефективний вид стрільби, проте безпечний). Це використовуються в іграх типу симулятора (Більшість ігор ділиться також на аркади та симулятори, аркадні ігри мають низький поріг входження, загальну низьку складність, орієнтовані на більш широку кількість гравців. Симулятори в свою чергу навпаки, більше реалізму, складніші умови та орієнтація на меншу аудиторію), такі ігри навпаки потребують як можна найкращих та розумних суперників. Прикладом є гра Escape from Tarkov розробників Battlestate Games [4]. В цій грі ворожі NPC ніколи

не підуть на гравця на пряму. Вони будуть ховатися, використовувати гранати та будь-яким чином захистити себе. В грі Metro Exodus [9] суперники здаються, коли більшість їх товаришів загинула. Таким чином вони визнають домінування гравця та намагаються зберегти своє життя [15].

Це все стосується людей, проте з тваринами трошки по-іншому. Тварини будуть нападати на людину тільки для того, щоб захистити свою територію чи через голод. Також тварини не ведуть бій до смертні, отримавши значні ушкодження, вони будуть тікати, щоб врятувати власне життя. Винятком є самки тварин, які захищають потомство. Вони будуть боротися до кінця життя. Є багато ігор, які містять велику фауну, для порівняння я обрав Fallout New Vegas (Obsidian Entertainment) [5] 2010 року та Red Dead Redemption 2 (Rockstar Studios) [11] 2019 року. Події серії ігор Fallout відбуваються на постапокаліптичному просторі навколо Лас-Вегаса. Усі тварини або загинули або мутували, майже усі тварини є агресивними та одразу атакують гравця, проте штучний інтелект майже відсутній. Вони ніколи не будуть використовувати переваги місцевості або тікати заради збереження життя. Вони намагаються підійти до гравця та атакувати його. В грі Red Dead Redemption 2 все набагато складніше. В грі реалізована унікальна механіка з полюванням на тварин. В ігровому світі є унікальні тварини, поведінка яких унікальна. Тварини майже ніколи не атакують гравця без вагомої причини та відступають при смертельній небезпеці. Поведінка різних груп тварин є різною. Більшість відступить від бою через смертельну небезпеку. Хижаки не вступають в бій наодинці, вони чекають зграю та атакують разом, при численних втратах, відступають [15].

### 2.1.2 Бажання перемогти

Не зважаючи хто атакує гравця, людина чи тварина, вони прагнуть отримати перемогу. В залежності від мотивації буде відрізнятися сила бажання та дії суперника.

У грі S.T.A.L.K.E.R. [32] та вже згаданій F.E.A.R, якщо гравець сидить в укритті, солдати ворога обов'язково будуть використовувати гранати та особливості локації, щоб спробувати обійти гравця з боку або з тилу. Також вороги кидають гранати не у гравця, а в останнє місце, де гравця бачили. Якщо в грі присутні особливі типи ворогів, вони будуть використовувати свої переваги для перемоги.

Проте в деяких іграх ворожі персонажі, які використовують специфічну зброю, застосовують її як звичайну. В грі Fallout 4 [17] є вороги, які використовують гранатомети як особисту зброю. Якщо гравець підійде до такого суперника впритул, він все одно буду її використовувати. Бажання перемогти пересилює здоровий глузд, тому ворог все одно вистрелить та завдасть собі смерті [15].

В усіх іграх, де персонажі використовують вогнепальну зброю, заміна основної зброї на другорядну, зазвичай пістолет, є швидшою ніж перезарядка основної, тому це є більш ефективно. Проте в більшості ігор суперники ніколи не використовують цей спосіб [15].

В серії ігор The Last of Us [18], де кількість набоїв для зброю дуже маленька, вороги будуть здаватися лише для того, щоб гравець підійшов як можна ближче, далі вони спробують використати зброю для ближнього бою, коли гравець втратить пильність [15].

### 2.1.3 Взаємодія між собою та з оточенням

Взаємодія з оточенням є найслабшим місцем штучного інтелекту, який керує ворожими персонажами. Більшість ворогів тільки можуть використовувати гранати та аптечки. Замінити зброю на більш потужну, обшукати тіло для знаходження корисних речей та підібрати набої з землі – усі ці дії суперники ніколи не виконують. Можливі дві причини таких недоліків в цьому аспекті штучного інтелекту: складно навчити штучний інтелект таким діям або підвищиться загальна складність гри і буде важче затримати гравця в грі [15].

Загалом ворожі персонажі не мають необхідності спілкуватися між собою, адже усю інформацію та команди вони отримують від спільного джерела. Всі фрази, якими вони обмінюються, необхідні лише для кращої атмосфери. Також в більшості ігор відсутня взаємодія між персонажами ворогів, хоча спільними діями завжди можна досягти кращого результати. Досить мала кількість штучного інтелекту в іграх мають такі можливості. В уже згаданій раніше F.E.A.R. суперники постійно спілкуються між собою, підказують один одному де знаходиться гравець та старші по званню віддають накази. Це також допомагає гравцю. Він знає приблизну кількість ворогів, завдяки ворожим наказам може завчасно протидіяти ним. Наприклад покинути позицію, коли лунає команда «Граната». Суперники в даній грі діють разом, лише іноді окремий суперник спробую самотужки атакувати гравця. Схожі дії виконують також суперники в грі S.T.A.L.K.E.R. Обмінюються фразами, дають накази та діють командно. В обох іграх суперниками є звичайні люди, наступний приклад – Mass Effect: Andromeda [19], де представники інших космічних рас [15].

В грі Mass Effect: Andromeda [19] присутні важко озброєні суперники та суперники, які використовують спеціальний щит. Під час бою такі суперники атакують гравця першими та змушують його змінити позицію або відступити. Це

дає змогу іншим солдатам ворога зайняти кращі позиції або перезарядити зброю. Також в грі присутні офіцери, які мають різні здібності. Вони не вступають у бій, а знаходять за спинами солдатів та допомагають їм. Зцілення, зробити солдата невидимим, поліпшення зброю – це лише декілька прикладів з усього арсеналу можливостей офіцерів [15].

### 2.1.4 Стелс ігри

Поведінка суперників відрізняються в залежності від жанру гри. В іграх жанру стелс особливі вимогу до штучного інтелекту, який керує суперниками, оскільки гравець, як правило, не вступає у відкритий бій. На жаль ігрова індустрія останні роки розвиваються у розважальному напрямку. Це означає, що будь-хто, незалежно від досвіду, повинен мати можливість грати в задоволення. Тобто ігровий процес з кожним роком стає все більш легким, щоб у гравця не було складнощів з проходженням певного рівня або він не отримував негативні емоції під час чергової поразки. Тому розробники додають нові механіки в ігри та адаптують штучний інтелект. Серед нових механік можна відокремити індикатор тривоги, коли гравцю повідомляють, що певний суперник його помітив або може помітити, також гравець бачить, чи шукають його або пошук закінчено та можна діяти далі, «Орлине око», яке вперше використали в серії ігор Assassin's Creed [20], воно дозволяє бачити противників через перешкоди, потім ця механіка з’явилась в інших іграх, але під іншою назвою, «стелсянка» - це придуманий синдром, яким хворіють майже усі суперники в стелс іграх, противники постійно кашляють, свистять, топчуться на місці, бормочать, задля того, щоб гравець міг без проблем відслідковувати противників по локації, індикатор освітленості, який вказую гравцю чи можуть його побачити [15].

Важливою складовою таких ігор є реакція на оточення. В серії ігор Thief ворог, який помічає гравця або труп, одразу кличе на допомогу та починає

шукати загрозу. Нейтральні персонажі будуть або тікати або кликати стражу. Проте основний недолік більшості таких ігор це те, що вороги через певний час перестають шукати гравця та повністю заспокоються. Вони будуть переступати через труп, ніби нічого не сталося. Так само вороги не реагують на пастки, які розставлені на видних місцях, ігнорують відкриті двері, які були зачиненими або навпаки. Деякі вороги пересуваються в темноті без жодного джерела світла, хоча очевидно, що вони нічого не помітять. В грі Metro вороги використовують ліхтарик на голові, але це використовуються для позначення поля зору ворога, відповідно гравець також може це використовувати [15].

### 2.2 Імітація другого гравця

Велика кількість ігор потребують двох людей для гри. Для того, щоб тільки один гравець міг грати, розробники створюють режими, де комп'ютер керує другим «гравцем». Основний гравець може обирати рівень складності в залежності від власного досвіду та навичок. В ігри, які створені тільки для одного гравця, додають умовного суперника, який робить гру цікавішою та змушую гравця грати краще. Розробнику досить створити алгоритм, який буде грати у їхню гру та налаштувати його для різної складності.

### 2.3 Імітація «живого світу»

В іграх рівня ААА [21] (умовна підмножина відеоігор, створюваних й розповсюджуваних середніми й великими видавництвами, що зазвичай мають більше коштів на розробку й рекламу) штучний інтелект також керую майже усіма персонажами, які наповнюють місто чи іншу локація. Він відповідає за правильну реакцію персонажів на дії гравця, найкращий приклад – це реакція поліція у грі Grand Theft Auto V [22]. Поліція використовую систему укриттів, кажуть один одному про позицію гравця, допомагають сховатися пораненому товаришу, викликають підкріплення в разі необхідності та інше. Також був

випадок, коли патрульна машина поліція мала розбиті передні ліхтарі. Вночі важко їхати по не освітленій дорозі, проте напарник водія використовував власний ліхтар для освітлення. Це все відбувалося виключно під керуванням комп’ютера. Як повідомляє розробник (Rockstar North), вони витратили дуже значну кількість часу на розробку штучного інтелекту та не знаю його повні можливості, адже про певні можливості вони дізнавалися вже від гравців. Дії інших персонажів також відображають виконану роботу над штучним інтелектом гри. Вони чекають автобус, ходять на роботу, влаштовують вечірки. Якщо гравець зупиниться біля іншого персонажа, він спробую поговорити з ним. Якщо довго йти за персонажем, через певний час він почне бігти, адже подумаю, що його переслідують. Якщо довго йти за жіночим персонажем, вона подзвонить в поліцію.

Даний вид штучного інтелекту досить повільно розвивається, такий висновок можна зробити через гру Cyberpunk: 2077 (CD Project Red) [23]. Гра має чудові візуальну та сюжетну частини. Проте більшість користувачів скаржиться на жахливий штучний інтелект, який керую жителями міста, деякі називають його найгіршим штучним інтелектом серед сучасних ігор. Перелік основних недоліків [24]:

- Відсутній денний та нічний цикли життя для персонажів, вони просто зникають та з’являються за межами зору гравця.
- Якщо персонаж виконую певну дію, він буде виконувати її поки гравець не почне рухатись.
- Персонажі, які не стосуються жодного завдання, мають тільки одну репліку діалогу.
- Якщо гравець атакую одного персонажа, усі інші персонажі в певному радіусі просто присідають на землю (однакова анімація для всіх), навіть якщо вони не бачили та не чули джерело небезпеки.

- Усі персонажі, що рухаються за кермом, мають чіткий маршрут руху. Якщо цим маршрутом чомусь не можна рухатись (гравець заблокував маршрут своїм авто), персонажі просто зупиняться назавжди, доки бар'єр не буде прибрано.
- Персонажі в автомобілі не покинуть його при небезпеці.
- Персонаж ніяк не зреагує, якщо переїде гравця на своєму авто.
- Якщо гравець силою відбере у персонажа авто, потерпілий не спробує повернути його чи викликати поліцію.
- Персонажі ніколи не будуть атакувати гравця або захищатися.
- Коли поліція розшукую гравця через скоєний злочин, поліцейські з'являються прямо в полі зору гравця, тобто «з повітря». Якщо гравець намагається втекти, вони будуть з'являтися тільки позаду руху або попереду.
- Поліція не реагує на незначні порушення, наприклад на ДТП.
- Поліція не здатна керувати автомобілями та переслідувати гравця, якщо він намагається втекти.
- Якщо гравець знаходиться далеко від міста та атакує іншого персонажа, то поліція все одно про це дізнається та біля гравця з'явиться патрульний без автомобіля.
- Якщо гравець допоміг персонажу, він ніколи не подякують.
- Якщо персонаж має унікальний діалог, який необхідний для завдання, то такий персонаж залишиться на завжди на місці, де цей діалог почався та буду його постійно повторювати, коли гравець повернеться.

Розробники відслідковують свої помилки та працюють над поліпшення штучного інтелекту.

Ігрові рушії, які дозволять гравцям власноруч розробляти модифікації для гри, допомагають в цій проблемі. Так для ігор The Elder Scrolls: Oblivion, The

Elder Scrolls: Skyrim, Fallout 3 та Fallout: New Vegas розроблена велика кількість модифікацій, які покращують певні аспекти гри. Штучний інтелект персонажів не став винятком, тому кожний бажаючий може завантажити певну модифікацію та використовувати її. Розробники таких ігор надають спеціальні можливості для ентузіастів, які розробляють модифікації, та допомагають їм. Розробникам це також вигідно, адже таким чином гра продовжую вдосконалюватись та не втрачати аудиторію.

## 2.4 Напарники

Система напарників присутня в обмеженій кількості ігор. Це вже згадані раніше The Elder Scrolls: Oblivion, The Elder Scrolls: Skyrim, Fallout 3, Fallout: New Vegas та Fallout 4. Гравець має можливість обрати, хто буду допомагати йому чи взагалі відмовитися від допомоги. Задачі, які виконую напарник досі прості. Переважно більшу частину часу він просто слідує за гравцем, коли гравець вступає у бій, напарник також починає бій, коли гравець ховається, напарник також присідає та рухається повільно. Напарнику можна віддати базові команди, відпустити його, обмінятися речами чи просто поспілкуватися. Як можна здогадатись, штучний інтелект напарників також не є ідеальним. Так наприклад в приміщені чи в печері можна застряти, адже напарник може застряти в дверях або в іншому вузькому проході. Загалом напарники не потребують складного штучного інтелекту, в вище перелічених іграх напарники не можуть загинути, отже в них відсутня тактика ведення бою. Також присутні різні модифікації для ігор, щоб покращити поведінку напарників.

# РОЗДІЛ 3 Розробка штучного інтелекту для гри Тетріс

## 3.1 Вибір мови програмування

Для більшості штучних інтелектів використовують мову Python через велику кількість готових бібліотек, які полегшують процес створення штучного інтелекту. Я обрав саме мову C++, яка дозволяє більш ефективно використовувати пам’ять та є однієї з найшвидших мов.

## 3.2 Опис правил гри та вибір стратегії

Гра Тетріс була розроблена в 1984 році. Ігрове поле – це прямокутник з шириною 10 клітинок та висотою 20 клітинок. Іноді його називають стаканом. Під час гри зверху з’являється випадкова фігура – тетраміно. Гравець може обертати фігуру та рухати вліво та вправо. Сама фігура поступово опускається вниз, доки не опиниться на іншій фігурі або на дні ігрового поля. Гравець має можливість опускати фігуру вниз, коли він вже обрав, куди має потрапити фігура. Якщо заповнюється горизонтальний ряд з 10 клітинок, тоді цей ряд знищується, гравець отримую бали, а поле вище даної лінії опускається на одну клітинку. Обмежень не має, тобто можна знищити будь-яку кількість ліній одночасно. Гра закінчується, коли нова фігура не може з’явитися. Задача гравця – це знищити як можна більше ліній [26].

Кількість фігур обмежена, їх 7. Вони не маю назв, проте з часом їх почали позначати великими літерами англійського алфавіту.

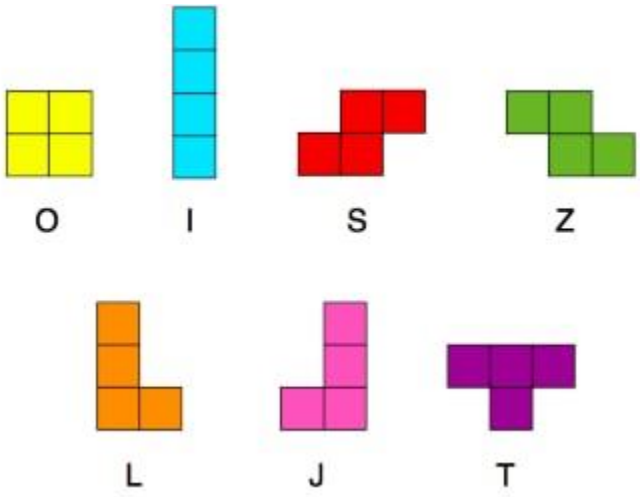



*Рисунок 3.2[29]*

Кожна фігура також має свою вісь оберту, це зроблено через те, що деякі фігури, як I, не можуть обертатись навколо центра. Адже вони мають займати відповідні клітинки та не можуть розташовуватись не в межах клітинок.

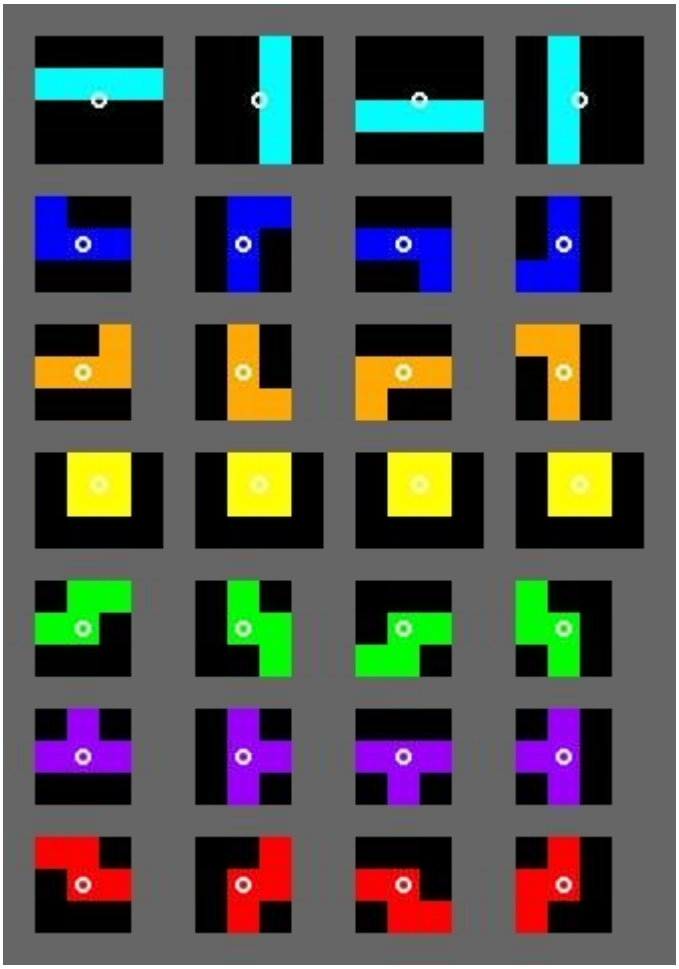


*Рисунок 3.3[30]*

### 3.2.1 Вибір наступної фігури

Тетріс має довгу історію, за її час алгоритм, який обирає наступну фігуру змінювався. В оригінальній грі наступна фігура обиралась випадково, незалежно від раніше обраних фігур. Це призводить до двох явищ. До повені та засухи. «Повінь» – це випадання декілька разів однакової фігури або схожих (схожі фігури: L та J, S та Z). «Засуха» – це ситуація, коли необхідна фігура не з’являється. Оригінальна гра не використовувала жодного алгоритму, лише випадковий вибір серед усіх фігур[27].

Гра Тетріс від Nintendo мала власний алгоритм для вибору наступної фігури, але досить простий. Попередню фігуру запам’ятовували, далі обрана

фігура порівнюється з попередньою, якщо вони однакові, вибирають знову, але тільки один раз. Ймовірність випадання двох однакових фігур зменшуються в 7 раз, тобто з 1/7 до 1/49. Проте цей алгоритм ніяк не бориться з «засухами» [27].

Наступна спроба була у The Grand Master. Їх алгоритм також був не складним, лише вдосконаленим алгоритмом від Nintendo. Тепер запам’ятовують 4 фігури, а спроб, щоб обрати фігуру, якої немає в історії фігур, є 4. Цей алгоритм також не є ідеальним, проте значно краще ніж попередній. Він гарно бориться з «повенями», проте не є сильно ефективним проти «засухи» [27].

Наступні два алгоритми схожі та використовуються в сучасних версіях гри Тетріса. Вони, в свою чергу, використовують колекцію типу Bag. Перший алгоритм «кладе» в мішок усі фігури, далі витягує випадкову фігуру. Коли мішок виявляється пустим, туди знову поміщають усі фігури. Таким чином усі фігури точно з’являться, а однакова фігура може з’явитися два рази підряд, проте далі 6 фігур точно будуть відрізнятися. Проте все одно є ймовірність випадання двох схожих фігур підряд чотири рази, вона дорівнює 0.22 відсотка. Другий алгоритм також використовує колекцію Bag, проте в мішок потрапляють одразу 35 фігур (5 фігур кожного типу). Коли фігуру достають з мішка, туди одразу кладуть нову фігуру, яку доставали найдавніше. Тобто кількість фігур, якої давно не було, в мішку буде збільшуватись, що збільшує ймовірність отримати цю фігуру з мішка. Таким чином цей алгоритм є ефективним, він дає фігури, яких не було до цього, та навпаки не дає фігури, які були перед цим[27].

Задача гравця – це знищити як можна більше ліній. В світі існую безліч стратегій, проте головне питання – це як оцінити хід. Який хід є гарним, який хід є поганим. Створюючи штучний інтелект було обрано 4 основних критерія.

### 3.2.2 Загальна висота фігури

Загальна висота фігури – це перший та досить очевидний критерій. Чим вище фігура буде розташована, тим гірше. Наступні рисунки (Рисунок 3.4 та Рисунок 3.5) гарно ілюструють чому висота фігури погано впливає на результат. Отже чим нижче фігура буде розташована, тим краще.

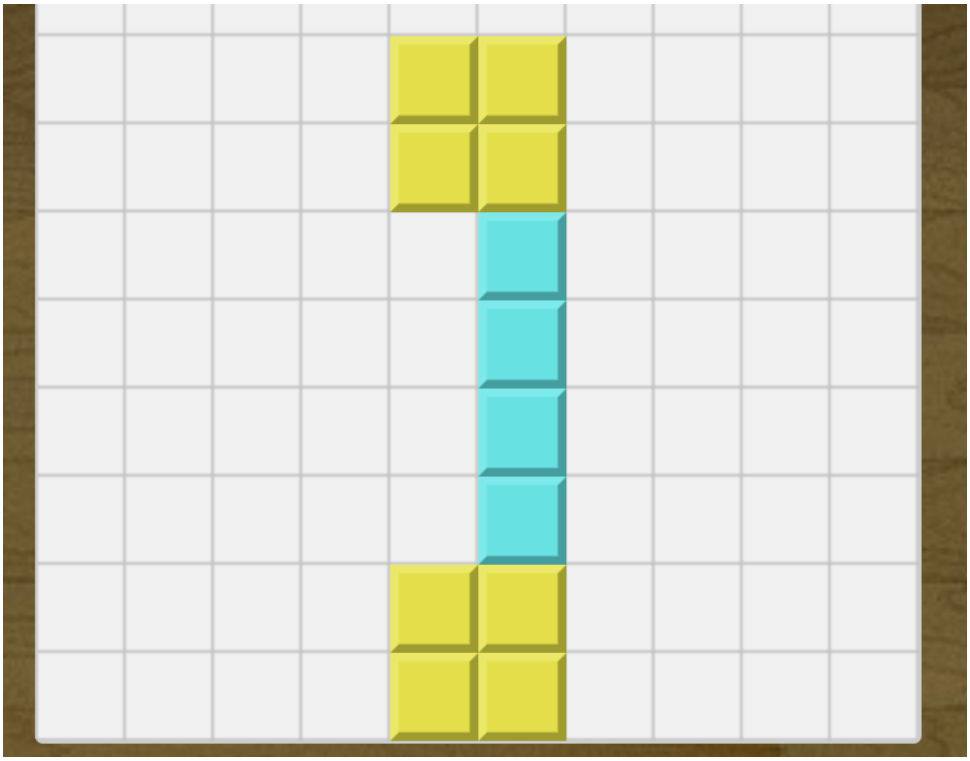


*Рисунок 3.4*

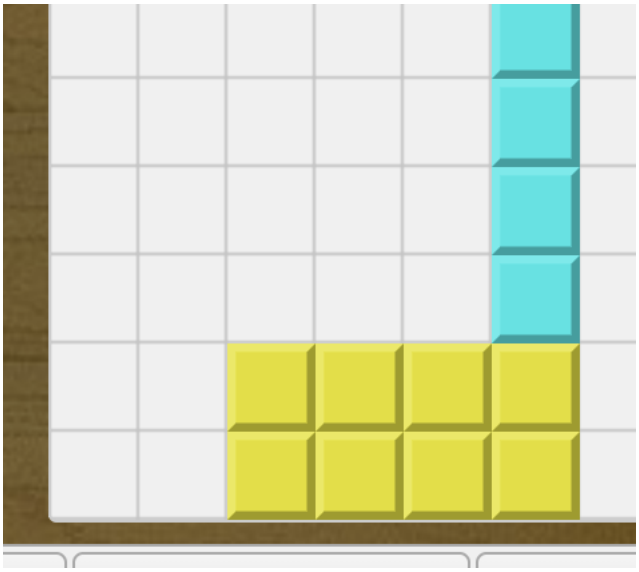


*Рисунок 3.5*

### 3.2.3 Заблоковані клітинки

Заблоковані клітинки – це найважливіший критерій на мою думку. Це такі клітинки, які заблоковані зверху, тобто більшість фігур не має до них доступу. Це означає, що необхідно зруйнувати усі лінії зверху таких клітинок, що б мати до них доступ. Таким чином це зменшую ігрове поле.

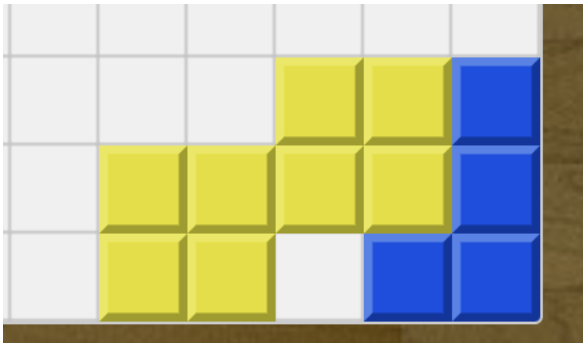


*Рисунок 3.6*

### 3.2.4 Пусті стовпчики більші за 2 клітинки

Досить розповсюджена стратегія гри полягає в тому, щоб залишати крайній стовпчик поля вільним, далі просто очікувати фігуру I (фігура, яка містить чотири

клітинки в ряд, найдовша фігура), щоб знищити певну кількість рядків та грати далі. Також є фігури L та J, які можуть закрити стовпчики з двох кубиків. Отже стовпчик більше двох кубиків – це погано. Основний недолік такої гри – це випадковість. Адже наступна фігура випадкова, отже необхідної фігури I довгий час може не бути. Як я вже описав способи обрати наступну фігуру в розділі 3.2.1, необхідна фігура рано чи пізно з’явиться.

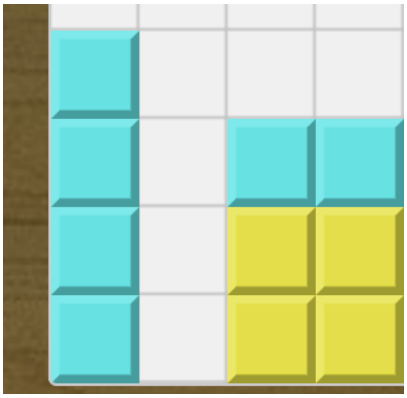


*Рисунок 3.7*

### 3.2.5 Знищення ліній

Останній критерій для оцінки ходу. Якщо хід, знищує лінії, це позитивно впливає на гру. З’являється додаткове місце та зараховуються бали. Це і є основна ціль гри, отже такі ходи мають бути винагородженні. Залишилось тільки оцінити ці критерії та визначити, які важливіші за інші.

## 3.3 Класи та структура програми

### 3.3.1 Shape

Клас Shape реалізує фігури в грі, структура зображена в Додатку А. Абстрактний клас Shape має необхідні методи для взаємодії (оберт фігури, стан фігури, який відповідає за правильний оберт, хід вправо та хід вліво, хід вниз). Кожна конкретна фігура унаслідує цей абстрактний клас та має унікальну

реалізацію оберту фігури та власний конструктор, який задає початкові координати кожної фігури.

### 3.3.2 Game

Клас Game відповідає за сам процес гри, структура зображена в додатку Б. Він містить поле, поточну фігуру та набір методів для гри. Поле зберігається в масиві типу bool (bool займає тільки один байт). Фігура зберігається через указник типу Shape. Таким чином використовується поліморфізм. Методи в класі дозволяють керувати фігурою (хід вправо, хід вліво, оберт фігури), є методи, які дозволяють перевірити, чи можливо зробити хід, обрати нову фігуру, вивести поле гри. В класі є поле, яке відповідає за розмір поля гри, поле, яке відповідає за рахунок гри. Для зручності також було додане поле, яке є указником на значення масиву координатів фігури, таким чином можна змінювати координати фігури, через поле в класі Game.

### 3.3.3 AIGame

Клас AIGame унаслідую клас Game, структура зображена в додатку В. Цей клас унаслідує клас Game. В клас AIGame додані методи необхідні для роботи з штучним інтелектом. Загалом усі вони необхідні для прорахунку наступного кроку.

### 3.3.4 AI

Клас AI відповідає за обрання наступного найкращого кроку, структура зображена в додатку Г. Поля зберігають усю необхідну інформацію, методи надають можливість розрахунку необхідних показників для обрання наступного кроку.

### 3.3.5 Array та Sequence

В програмі використовуються два допоміжні класи Array та Sequence. Вони використовуються як колекції.

## 3.4 Опис алгоритму

Спочатку алгоритм має тільки поле гри та поточну фігуру. Далі необхідно отримати всі можливі варіанти розміщення поточної фігури на полі гри. Для цього визначається ширина та висота фігури, для того, що б уникнути ситуації, коли фігура буде розташована за межами полю. Далі визначається крайня точка фігури по правій та нижній сторонам. Це необхідно для правильного розташування фігури на полі. Далі починається процес коли фігуру підставляють на поле гри. Початкова координата це нижня права точка поля. Фігура вставляється на поле, далі нове поле з фігурою перевіряється на валідність. Валідність розташування фігури передбачає дві перевірки – перевірка, щоб не було колізії з фігурою та полем (тобто фігура не має накладатись на інші клітинки поля), та перевірка чи спирається фігура на кубик чи на нижню частину ігрового поля. Якщо поле з фігурою підходить, тоді позиція, куди підставляється фігура зміщається на одну клітинку вліво, а саме поле з фігурою зберігається, якщо поле не підходить, тоді фігура підставляється на одну клітинку вище, поки поле пройде валідацію або висота поля не дозволить розмістити фігуру. Таким чином фігура підставляється на поле в усіх можливих позиціях та обертах.

Далі необхідно обрати розташування фігури, яке має найкращий результат. Як описується в попередньому розділі, є 4 критерія за якими кожне розташування фігури отримає власну оцінку. Чим оцінка нижче, тим краще розташування фігури. Для кожного розташування фігури на полі розраховується висота, кількість закритих клітинок, кількість пустих стовпчиків з висотою більше ніж два кубика та кількість заповнених ліній. Далі кожне значення множиться на

відповідний коефіцієнт критерія та додається, окрім кількості повних ліній. Це значення віднімається від суми, таким чином, чим нижче оцінка розташування фігури на полі, тим краще. Розташування фігури з найменшою оцінкою є переможцем. Далі обирається нова фігура та алгоритм повторює свою роботу. Гра закінчується, коли нова фігура не може з’явитись, тобто на координатах нової фігури вже розташовується кубик.

Значення коефіцієнтів для кожного з критерія є невідомими. В процесі тестування штучного алгоритму було встановлено, що оптимальними значеннями є:

- Для закритих полів – 75.
- Для висоти – 45.
- Для пустих стовпчиків – 5.
- Для заповнених ліній – 90.

За таких значеннях штучний інтелект знищую за гру приблизно 1000 ліній. Такі значення є оптимальними, проте існують і інші значення, які можливо дадуть кращий результат.

Використавши сучасний алгоритм, який описаний в розділі 3.2.1 та обирає наступну фігуру, штучний інтелект зміг досягти значно кращого результату, а саме 17000 ліній, що перевищую рекорд людини більше ніж в 3 рази.

## Висновок

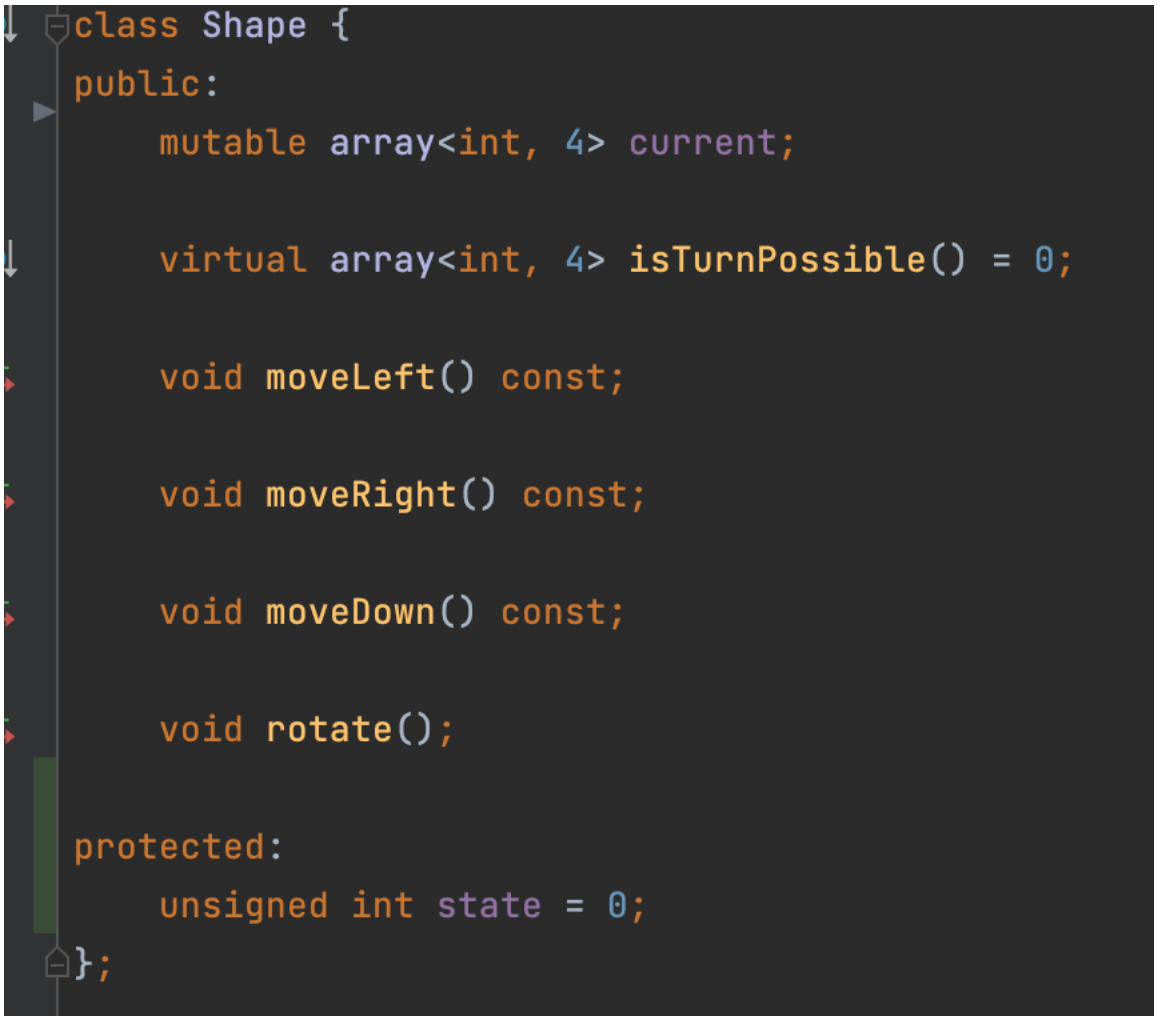
В результаті дослідження виявлено, що розвиток штучного інтелекту в відеоіграх значно повільніший ніж інші складові відеоігор. Більшість користувачів негативно оцінюють можливості штучних інтелектів сучасних відеоігор. Проте вони все одно розвиваються, хоча не значними темпами.

В результаті розробки штучного інтелекту для гри Тетріс, був отриманий алгоритм, який зможу перемогти більшість гравців в цій грі. Проте він також потребую покращень, а саме яким чином визначити найкращий хід. Додавання нових критеріїв допоможе отримати кращий результат.

Посилання на репозиторій з власним проектом - https://github.com/danilboiko1302/AI-Tetris

## Додаток А

Структура класу Shape



```
class Shape {
public:
    mutable array<int, 4> current;

    virtual array<int, 4> isTurnPossible() = 0;

    void moveLeft() const;

    void moveRight() const;

    void moveDown() const;

    void rotate();

protected:
    unsigned int state = 0;
};
```

## Додаток Б

Структура класу Game

```
    void destroyLine(int line);

    bool checkMoveDown();

    void removeShape();

    void seeShape();

    bool addShape();

    static constexpr unsigned int size = 20;

    unsigned long long score = 0;

    Shape *current{};

protected:
    array<bool, (size * 10)> ar;

    array<int, 4> *currentArr{};
};
```

## Додаток В

Структура класу AIGame

```
class AIGame : public Game {
public:
    void play() override;

    array<bool, Game::size * 10> getBoard();

    array<int, 4> *getShape();

    Shape *getShapeClass();

    void setBoard(array<bool, Game::size * 10> newAr);

    void destroyLine();

    array<bool, Game::size * 10> nextMoveDown();

};
```

# Додаток Г

Структура класу AI

```
class AI {
public:
    AI();

    explicit AI(int, int, int, int, int, int);

    AI &operator=(const AI &);

    unsigned long long start();

private:
    int sonsAmount;
    int hole;
    int height;
    int moreThan3Holes;
    int step;
    int destroy;
    int score = 0;

    unsigned long long sonsPlay();

    static Sequence<array<bool, Game::size * 10>> &allBoards(Shape &, array<bool, Game::size * 10>);

    static bool checkBoard(array<int, 4>, array<bool, Game::size * 10>);

    static int width(array<int, 4>);
```

```
    static int heightShape(array<int, 4>);
    static int scoreBoard(array<bool, Game::size * 10>, int, int, int, int);
    static int countDestroy(array<bool, Game::size * 10>);
    static int countHoles(array<bool, Game::size * 10>);
    static int countHeight(array<bool, Game::size * 10>);
    static int countColumns(array<bool, Game::size * 10>);
    static void seeBoard(array<bool, Game::size * 10>);
    static bool turn(AIGame &);
    static void goToLeftSide(AIGame &);
    void createSons();
    void show() const;
    AI(const AI &);
    AI(int, int, int, int);
    Sequence<AI> *sons = new Sequence<AI>;
};
```

## Список використаної літератури:


| | |
|---|---|
| Електронні ресурси | 1. Building the AI of F.E.A.R. with Goal Oriented Action Planning, AI and Games. AI and Games. URL: https://www.aiandgames.com/2020/05/06/ai-101-goap-fear/.<br>2. Automatic Computing Engine - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Automatic_Computing_Engine.<br>3. Castle Wolfenstein - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Castle_Wolfenstein.<br>4. Escape from Tarkov. URL: https://www.escapefromtarkov.com/?lang=en.<br>5. Fallout: New Vegas. URL: https://fallout.bethesda.net/en/games/fallout-new-vegas.<br>6. F.E.A.R. (video game) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/F.E.A.R._(video_game).<br>7. Half-Life (video game) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Half-Life_(video_game).<br>8. Left 4 Dead - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Left_4_Dead.<br>9. Metro Exodus. URL: https://www.metrothegame.com/en-us/.<br>10. Pac-Man - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Pac-Man.<br>11. Red Dead Redemption 2. URL: https://www.rockstargames.com/reddeadredemption2/.<br>12. The Elder Scrolls IV: Oblivion. URL: https://elderscrolls.bethesda.net/en/oblivion/. |

13. Thief: The Dark Project - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Thief:_The_Dark_Project.

14. Turochamp - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Turochamp.

15. Gennadium. Искусственный интеллект в играх, или «Тупой и еще тупее» / Компьютерные и мобильные игры / iXBT Live. iXBT Live. URL: https://www.ixbt.com/live/games/iskusstvennyy-intellekt-v-igrah-ili-tupoy-i-esche-tupee.html.

16. The Director. Left 4 Dead Wiki. URL: https://left4dead.fandom.com/wiki/The_Director.

17. Fallout 4. URL: https://fallout.bethesda.net/ru/games/fallout-4.

18. The Last of Us - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/The_Last_of_Us

19. Mass Effect: Andromeda. URL: https://www.ea.com/games/mass-effect/mass-effect-andromeda/gameplay

20. Assassin's Creed - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Assassin's_Creed

21. AAA (video game industry) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/AAA_(video_game_industry)

22. Grand Theft Auto V. URL: https://www.rockstargames.com/ru/games/V

23. Cyberpunk 2077. URL: https://www.cyberpunk.net/us/en/

24. r/cyberpunkgame - The AI of Cyberpunk 2077: An In-depth Look at the Worst AI in Modern Gaming. reddit. URL: https://www.reddit.com/r/cyberpunkgame/comments/kbk4ap/the_ai_of_cyberpunk_2077_an_indepth_look_at_the/

25. Radiant AI - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Radiant_AI

| | |
|---|---|
| | 26. Тетріс – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Тетріс<br>27. PatientZero. История алгоритмов рандомизации «Тетриса». Все публикации подряд / Хабр. URL: https://habr.com/ru/post/466579/<br>28. PatientZero. Искусственный интеллект Half-Life SDK: ретроспектива. Все публикации подряд / Хабр. URL: https://habr.com/ru/post/331438/<br>29. How is Tetris related to math? - Quora. Quora - Quora, wo man Wissen miteinander austauscht und die Welt besser verstehen kann. URL: https://www.quora.com/How-is-Tetris-related-to-math<br>30. Tetris Rotation Issue. Stack Overflow. URL: https://stackoverflow.com/questions/52827159/tetris-rotation-issue<br>31. Игра с искусственным интеллектом, который был «слишком умным». URL: http://muz4in.net/news/igra_s_iskusstvennym_intellektom_kotoryj_byl_slishkom_umnym/2018-10-09-46899<br>32. S.T.A.L.K.E.R. - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/S.T.A.L.K.E.R.<br>33. https://github.com/danilboiko1302/AI-Tetris |