УДК 519.21:004.421.2
DOI https://doi.org/10.32782/2521-6643-2026-2-72.12

Іляш Ю. Ю., кандидат технічних наук, доцент,
завідувач кафедри комп'ютерних наук та інформаційних систем
Карпатського національного університету імені Василя Стефаника
ORCID: 0009-0006-5786-7205

Ровінський В. А., кандидат технічних наук, доцент, доцент
кафедри комп'ютерних наук та інформаційних систем
Карпатського національного університету імені Василя Стефаника
ORCID: 0000-0001-8454-8580

Гейко А. О., студент-бакалавр кафедри комп'ютерних наук
та інформаційних систем
Карпатського національного університету імені Василя Стефаника
ORCID: 0009-0003-0649-5470

АРХІТЕКТУРИ ТА АЛГОРИТМИ ПРИЙНЯТТЯ РІШЕНЬ НЕІГРОВИХ ПЕРСОНАЖІВ У КОМП'ЮТЕРНИХ ІГРАХ

Стаття присвячена комплексному дослідженню моделі поведінкових дерев (Behavior Tree) як інструменту реалізації штучного інтелекту неігрових персонажів(Non-Player Character; NPC; Агент) у сучасних комп'ютерних іграх.

Обґрунтовано актуальність застосування ієрархічних архітектур прийняття рішень у контексті зростання вимог до інтерактивності, адаптивності та правдоподібності поведінки NPC. Показано, що структурована модель дерева поведінки забезпечує логічну впорядкованість дій агента, прозорий механізм пріоритетності та можливість масштабування системи без втрати керованості.

У роботі запропоновано формалізовану архітектуру поведінкового дерева з виокремленням композиційних вузлів типу Selector і Sequence, а також листових вузлів, що реалізують конкретні дії (переслідування, атака, патрулювання, очікування). Детально проаналізовано механізм тристанної моделі виконання (Success, Failure, Running) та принцип tick-based оцінювання, що забезпечує реактивність системи у реальному часі. Особливу увагу приділено побудові пріоритетної ієрархії поведінки, у межах якої бойові сценарії мають вищий рівень значущості, а альтернативні стани виконують роль фонових або резервних дій.

Практичну реалізацію моделі здійснено в середовищі Unity з використанням мови програмування C#. Унікальність запропонованих реалізації алгоритмів, полягає в адаптивності коду і для інших рушіїв на базі мови C++. Представлено приклади програмної реалізації базового класу вузла, композиційних структур та ініціалізації дерева, що демонструють відповідність теоретичних положень реальному програмному втіленню. Показано, що запропонована архітектура є модульною, розширювальною та придатною для інтеграції в ігрові проєкти різної складності.

Отримані результати підтверджують, що поведінкові дерева дозволяють поєднати алгоритмічну строгість із гнучкістю геймдизайну, забезпечуючи передбачувану, але водночас динамічну поведінку агентів. Запропонований підхід може бути використаний у навчальних прототипах, інді-проєктах і комерційних розробках, а також слугувати основою для подальших досліджень, пов'язаних Behavior Tree.

Ключові слова: *штучний інтелект, Behavior Tree, неігровий персонаж, прийняття рішень, ігровий ШІ, Unity, C#, C++, модульна архітектура.*

Iliash Y. Y., Rovinsky V. A., Heiko A. O. Algorithms and methods of adaptive behavior of non-player characters in computer games

The article is devoted to a comprehensive study of the Behavior Tree model as a tool for implementing artificial intelligence of Non-Player Characters (NPCs; Agents) in modern computer games.

The relevance of applying hierarchical decision-making architectures is substantiated in the context of increasing demands for interactivity, adaptability, and realism in NPC behavior. It is demonstrated that the structured behavior tree model ensures logical organization of an agent's actions, a transparent prioritization mechanism, and the possibility of system scalability without loss of manageability.

The paper proposes a formalized architecture of a behavior tree, distinguishing composite nodes such as Selector and Sequence, as well as leaf nodes that implement specific actions (pursuit, attack, patrol, idle/wait). The tri-state execution model

(Success, Failure, Running) and the tick-based evaluation principle are analyzed in detail, ensuring real-time system reactivity. Particular attention is given to constructing a priority hierarchy of behaviors, in which combat scenarios have higher significance, while alternative states serve as background or fallback actions.

The practical implementation of the model was carried out in the Unity environment using the C# programming language. The uniqueness of the proposed algorithm implementations lies in the adaptability of the code for other engines based on C++. Examples of programmatic implementation of the base node class, composite structures, and tree initialization are presented, demonstrating the correspondence between theoretical principles and real software implementation. It is shown that the proposed architecture is modular, extensible, and suitable for integration into game projects of varying complexity.

The obtained results confirm that behavior trees allow combining algorithmic rigor with game design flexibility, ensuring predictable yet dynamic agent behavior. The proposed approach can be used in educational prototypes, indie projects, and commercial developments, and can also serve as a foundation for further research related to Behavior Trees.

Key words: *artificial intelligence, Behavior Tree, non-player character, decision-making, game AI, Unity, C#, C++, modular architecture.*

Постановка проблеми. У XXI столітті комп'ютерні ігри перетворилися з нішевого виду розваг на потужну індустрію цифрового контенту, що охоплює мільйони користувачів у всьому світі. Стрімкий розвиток апаратного забезпечення та програмних технологій сприяв не лише зростанню якості ігор, а й значному спрощенню процесу їх розробки. Сучасні безкоштовні та відкриті ігрові рушії, такі як Unity (C#), Unreal Engine (C++) та Cocos2dx (C++), надають розробникам широкий набір інструментів для створення інтерактивних проєктів без необхідності глибокого занурення в низькорівневе програмування або розробку власного рушія з нуля.

Завдяки цьому бар'єр входу в геймдев суттєво знизився: створення ігор стало доступним не лише для великих студій, але й для індивідуальних розробників та невеликих команд. Проте разом із зростанням кількості проєктів підвищилися й вимоги до їхньої якості. Сучасний гравець очікує не лише захопливого сюжету чи привабливої графіки, а й складного, динамічного ігрового процесу, продуманих механік та реалістичної поведінки персонажів. Важливою складовою формування ігрового досвіду стала інтерактивність середовища та здатність ігрового світу реагувати на дії користувача.

Одним із ключових інструментів забезпечення такої інтерактивності є системи штучного інтелекту (ШІ), що керують поведінкою неігрових персонажів. Саме поведінка NPC значною мірою визначає рівень складності гри, її динаміку та ступінь занурення гравця у віртуальний світ. Недостатньо продуманий або передбачуваний ШІ знижує інтерес до гри, тоді як адаптивна та логічна поведінка персонажів підвищує реіграбельність і загальну якість продукту [1; 7–9].

Серед підходів до реалізації ігрового ШІ широкого поширення набули поведінкові дерева (Behavior Trees). Ця модель прийняття рішень ґрунтується на ієрархічній структурі вузлів, які послідовно перевіряють умови та виконують відповідні дії. Кожен вузол повертає один із можливих станів – успіх, невдачу або виконання, що дозволяє гнучко організовувати логіку поведінки агента. Поведінкові дерева відзначаються простотою, наочністю, модульністю та зручністю інтеграції в сучасні ігрові рушії[3; 5–6].

Паралельно з ними популярності набув підхід Utility AI, або модель корисності. На відміну від жорстко структурованих дерев рішень, Utility AI передбачає оцінювання кожної можливої дії за певною числовою функцією корисності (utility). Агент аналізує поточний стан середовища, розраховує значення корисності для кожної альтернативи та обирає дію з найвищим показником. Такий підхід забезпечує більш плавну, адаптивну та контекстно-залежну поведінку NPC[4; 10].

Попри широке використання обох моделей, питання їх ефективності, гнучкості, масштабованості та доцільності застосування в різних типах ігор залишається актуальним. Виникає необхідність порівняльного аналізу поведінкових дерев та Utility AI з метою визначення їх переваг, недоліків і оптимальних сфер застосування в сучасних ігрових проєктах. Саме ця проблема й зумовлює актуальність даного дослідження[4–5; 10].

Аналіз літературних даних та постановка проблеми. Проблематика створення ефективних моделей прийняття рішень є одним із ключових напрямів досліджень у сфері Game AI. З розвитком інтерактивності, у користувачів зросли вимоги щодо адаптивності, масштабованості та реалістичності неігрових персонажів, що призвело до активного пошуку оптимальних архітектур керування.

В роботі M. Colledanchise та P. Ögren [5] описано основи побудови моделі поведінки ШІ. Автори узагальнили загальні принципи побудови поведінкових дерев (Behavior Trees; BT), формально описали їх структуру, властивості та математичні основи. Таку ж саму проблему узагальнення та аналізу сучасного стану розвитку BT розглянули M. Iovino та ін. у праці [10], де проведено глибокий огляд використання поведінкових дерев у робототехніці та ігровому ШІ. У роботі продемонстровано, що BT забезпечують модульність, масштабованість та можливість повторного використання компонентів в порівнянні з скінченними автоматами станів (FSM).

Практичні аспекти застосування поведінкових дерев у розробці ігор продемонстровано в доповіді A. J. Champandard [6], де аргументовано доцільність їх використання для реалізації складної поведінки NPC у сучасних іграх. Аналогічну проблему керування складністю ієрархічної поведінки агентів

у великомасштабних ігрових проєктах розглянув D. Isla під час розробки системи ШІ для гри Halo 2 [7]. У цій роботі продемонстровано практичні переваги ієрархічних структур прийняття рішень у ААА-проєктах.

Загальні підходи до побудови систем штучного інтелекту в іграх, включно з FSM, деревами рішень, поведінковими деревами та плануванням, систематизовано в працях I. Millington та J. Funge [8], а також у збірниках S. Rabin [1; 9]. Дані джерела розглядають архітектури керування агентами та аналізують їх ефективність з позиції практичного застосування у геймдеві залежно від жанру гри.

У доповіді K. Dill [3] розглянуто альтернативний підхід до прийняття рішень: Utility AI. Автор пропонує використання теорії корисності для побудови гнучкіших систем вибору дій; використання числової оцінки дозволить створити більш адаптивну поведінку, яка плавно реагує на зміни станів середовища. Робота [4] M. Świechowski та ін. продовжила цей напрямок та розглянула проблематику комбінування Utility AI з алгоритмом Monte Carlo Tree Search, що продемонструвало більшу стратегічну глибину поведінки агентів у відеоіграх.

Також M. Buckland в своїй праці [2] розглянув класичні підходи до реалізації поведінкових моделей, зокрема FSM та керування на основі станів. Він продемонстрував практичні приклади реалізації агентів та проаналізував переваги й обмеженні різноманітних архітектур.

Згідно вище сказаного, аналіз останніх досліджень демонструє, що: поведінкові дерева є добре формалізованою і масштабованою архітектурою для реалізації складної поведінки NPC, класичні FSM залишаються актуальними для простих сценаріїв, але мають обмеження масштабованості, Utility AI забезпечує більш гнучкий, но в той же час контекстно-залежний механізм вибору дій, а комбіновані підходи дозволяють підвищити адаптивність і стратегічну складність агентів.

Натомість, поведінкові дерева природно підтримують ієрархічну структуру; пріоритетність вибору та формалізовану послідовність виконання дій завдяки вузлам типу Selector і Sequence. Це спрощує масштабування, модифікацію логіки та повторне використання компонентів.

Отже, поведінкові дерева можна розглядати як збалансований підхід, що поєднує структурованість, модульність і практичну зручність реалізації, що й обумовлює їх вибір у даному дослідженні.

Мета дослідження. Розвиток методів організації архітектури поведінки NPC є одним із основних напрямків сучасної ігрової розробки. Зі зростанням вимог до правдоподібності взаємодії між гравцем та неігровими персонажами постає проблема у формалізованих, гнучких і масштабованих моделях прийняття рішень. Однією з найефективніших та простих архітектур є модель поведінкових дерев, що забезпечує ієрархічну логіку дій та прозорий механізм керування станами виконання.

У цьому контексті теоретичне осмислення моделі BT разом із аналізом її програмної реалізації дозволяє не лише продемонструвати відповідність практичного коду концептуальним положенням, а й сформувати узагальнені підходи до проєктування систем ігрового штучного інтелекту.

Такий підхід дозволяє розглядати поведінкові дерева не лише як інструмент розробки, а як формалізовану модель організації поведінки агента, що має як прикладне, так і науково-методичне значення для розвитку сучасних систем ігрового штучного інтелекту.

Таким чином, метою дослідження є аналіз моделі поведінкових дерев як формалізованого підходу до організації поведінки NPC та дослідження особливостей її програмної реалізації в системах ігрового штучного інтелекту.

Викладення основного матеріалу дослідження. Під час розробки системи штучного інтелекту для неігрового персонажа особлива увага приділяється формуванню логічно впорядкованої, передбачуваної та масштабованої поведінки. Поведінка агента повинна одночасно реагувати на зміну стану середовища, зберігати внутрішню узгодженість і забезпечувати природність взаємодії з гравцем. Саме тому в межах дослідження було обрано модель поведінкового дерева (Behavior Tree) як основу архітектури прийняття рішень.

Поведінкове дерево розглядається як ієрархічна структура, у якій кожен вузол представляє або логічну операцію керування (Selector, Sequence), або конкретну дію чи перевірку умови (Node).

Така структура дозволяє розділити складну поведінку на незалежні модулі, що підвищує керованість системи та спрощує її розширення.

Запропонована модель(рис. 1) побудована за принципом пріоритетної ієрархії, де верхній рівень визначає загальний контекст поведінки (бойовий або пасивний режими), а нижчі рівні деталізують конкретні дії агента. У цьому підході логіка прийняття рішень формується не через складні вкладені умовні оператори, а через послідовну оцінку вузлів дерева.

Структура реалізованої моделі складається з таких компонентів:

1. Кореневий вузол (Selector) – визначає пріоритетність поведінки.
2. Гілка бойової взаємодії (Sequence) – реалізує логіку «виявлення → пересування → атака»
3. Гілка альтернативної поведінки (Selector) – забезпечує патрулювання або стан очікування.

Принцип роботи селектора полягає у послідовній перевірці дочірніх вузлів і поверненні першого стану Success або Running. Якщо всі вузли повертають Failure, селектор завершується з відповідним станом невдачі.

Sequence, у свою чергу, виконує вузли послідовно і припиняє виконання при першому Failure. Якщо всі вузли виконані успішно – повертається Success, а якщо хоча б один перебуває у процесі виконання – Running.

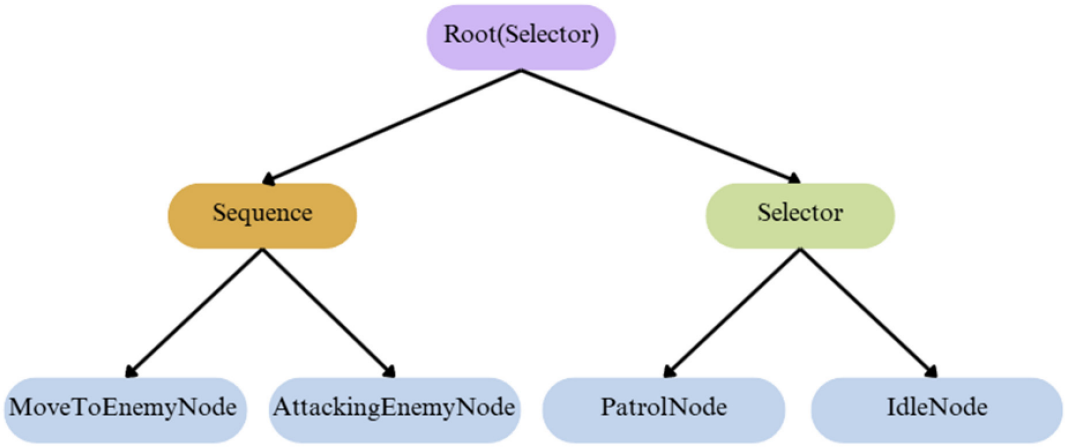

Рис. 1. Схема поведінкового дерева

Таким чином, структура дерева реалізує чітку логіку пріоритетності: бойова поведінка має вищий пріоритет за патрулювання, а патрулювання – за бездіяльність.

Кожен вузол повертає один із трьох можливих станів:

1. Success – дія виконана;
2. Failure – умова не виконана або дія неможлива;
3. Running – дія триває в часі.

Цей тристанний механізм дозволяє формалізувати поведінку агента як дискретну систему переходів між станами. Оцінювання здійснюється циклічно під час кожного кадру гри (tick-based evaluation), що забезпечує постійний контроль над поточною ситуацією.

Наприклад:

1. Якщо ворог знаходиться поза зоною досяжності, вузол переміщення повертає Running;
2. Якщо ворог досягнутий – Success;
3. Якщо ворога немає – Failure, і система переходить до альтернативної гілки.

Такий підхід гарантує реактивність системи без необхідності зберігати складні глобальні стани.

Реалізація моделі здійснена в середовищі Unity з використанням мови C#. Архітектура побудована за об’єктно-орієнтованим принципом, де базовим елементом системи є абстрактний клас Node, що визначає загальний інтерфейс для всіх типів вузлів.

```
public abstract class Node
{
public enum State { Success, Failure, Running }
public State state;
public abstract State Evaluate();
}
```

Лістинг 1 – реалізація базового класу вузла(Node).

Запропонована структура вводить тристанну модель виконання (Success, Failure, Running), що відповідає класичній формалізації Behavior Tree. Метод Evaluate() є поліморфним і реалізується в кожному конкретному типі вузла.

```
public class BTSelector : Node
{
private List<Node> children = new List<Node>();

public BTSelector(List<Node> nodes)
{
children = nodes;
}
public override State Evaluate()
{
    foreach (Node node in children)
        {
            switch (node.Evaluate())
            {
```

```
                case State.Success:
                    state = State.Success;
                    return state;
                case State.Running:
                    state = State.Running;
                    return state;
                    case State.Failure:
                    continue;
            }
        }
    state = State.Failure;
    return state;
    }
}
```

Лістинг 2 – реалізація класу селектора.

```
public class BTSequence : Node
{
private List<Node> children = new List<Node>();
public BTSequence(List<Node> nodes)
{
children = nodes;
}
public override State Evaluate()
{
bool anyRunning = false;
foreach (Node node in children)
    {
        switch (node.Evaluate())
            {
            case State.Failure:
                state = State.Failure;
                return state;
                case State.Running:
                anyRunning = true;
                break;
            case State.Success:
                continue;
            }
    }
    state = anyRunning ? State.Running : State.Success;
    return state;
    }
}
```

Лістинг 3 – реалізація класу послідовності.

```
public class MoveToEnemyNode : Node
{
protected BotController npc;
public MoveToEnemyNode(BotController npc)
{
    this.npc = npc;
}
public override State Evaluate()
{
    if (npc.isEnemyNear())
    {
        if (npc.isEnemyInAttackRange(npc.GetCurrentEnemy()))
        {
            return State.Success;
        }
        else
        {
            npc.MoveToEnemy(npc.GetCurrentEnemy());
```

```
            return State.Running;
            }
        }
        return State.Failure;
    }
}
```

Лістинг 4 – реалізація вузла переслідування противника.

```
public class AttackingEnemyNode : Node
{
protected BotController npc;

public AttackingEnemyNode(BotController npc)
{
    this.npc = npc;
}
public override State Evaluate()
{
    if (!npc.isEnemyInAttackRange(npc.GetCurrentEnemy()))
    {
        return State.Failure;
    }
    else if (npc.isAttacking())
    {
        return State.Running;
    }
    npc.AttackEnemy(npc.GetCurrentEnemy());
    return State.Success;
}
}
```

Лістинг 5 – реалізація вузла атаки противника.

У межах розробленої архітектури бойова поведінка представлена як послідовна композиція вузлів дерева поведінки. Вона включає три основні стани: виявлення противника, зближення з ним та здійснення атаки, що концептуально описується схемою: Виявлення → Наближення → Атака.

```
public class PatrolNode : Node
{
protected BotController npc;

public PatrolNode(BotController npc)
{
    this.npc = npc;
}

public override State Evaluate()
{
    if (!npc.HasPatrolPoints())
    {
        return State.Failure;
    }
    if (npc.isMovingToPatrolPoint())
    {
        npc.Patroling();
        return State.Success;
    }
    else
    {
        return State.Failure;
    }
    }
}
```

Лістинг 6 – реалізація вузла патрулювання.

Вузол PatrolNode відповідає за реалізацію патрулювання: він перевіряє наявність заданих точок маршруту та ініціює рух між ними.

```
public class IdleNode : Node
{
protected BotController npc;
public IdleNode(BotController npc)
{
    this.npc = npc;
}
public override State Evaluate()
{
    return State.Success;
}
}
```

Лістинг 7 – реалізація вузла простоювання.

Вузол IdleNode виконує функцію резервної або базової поведінки. Він повертає стан Success, забезпечуючи завершеність логічної структури дерева та гарантуючи, що агент завжди перебуває у визначеному стані навіть за відсутності активних дій. Такий підхід формує цілісну ієрархію поведінки, де бойові сценарії мають вищий пріоритет, а патрулювання й очікування виконують роль фонових або альтернативних дій.

```
void Start()
{
if (npc == null)
{
    Destroy(gameObject);
    return;
}

List<Node> sequence_nodes = new List<Node>();
sequence_nodes.Add(new MoveToEnemyNode(npc));
sequence_nodes.Add(new AttackingEnemyNode(npc));
BTSequence sequence = new BTSequence(sequence_nodes);

List<Node> selector_childrens = new List<Node>();
selector_childrens.Add(new PatrolNode(npc));
selector_childrens.Add(new IdleNode(npc));
BTSelector selector = new BTSelector(selector_childrens);

List<Node> root_childrens = new List<Node>();
root_childrens.Add(sequence);
root_childrens.Add(selector);
root = new BTSelector(root_childrens);
}
```

Лістинг 8 - інцилізація дерева.

Оцінювання поведінки відбувається у методі Update():

```
void Update()
{
if (root != null)
root.Evaluate();
}
```

Лістинг 9 – метод Update() поведінкового дерева.

Це реалізує tick-based модель, при якій дерево переоцінюється на кожному кадрі.

Інтеграція теоретичної моделі Behavior Tree з програмною реалізацією в Unity демонструє відповідність концептуальних положень і практичного застосування. Архітектура забезпечує:

1. структурованість логіки;
2. чітку пріоритетність дій;
3. модульність і розширюваність;
4. передбачувану реактивність поведінки.

Включення кодових фрагментів у дослідження дозволяє наочно підтвердити, що поведінкове дерево виступає не лише теоретичною конструкцією, а й ефективним інструментом реалізації систем ігрового штучного інтелекту.

Висновки. На основі розробленої моделі продемонстровано формалізацію поведінки ігрового бота у вигляді чітко структурованої системи умов та дій. Показано, що використання дерева поведінки забезпечує модульність, масштабованість та прозорість логіки прийняття рішень. Запропонована структура дозволяє реалізувати пріоритетність поведінки (бойова активність), альтернативні стани (патрулювання) та резервну поведінку (очікування), що формує гнучку модель реагування на зміну ігрової ситуації.

Отримані результати підтверджують доцільність застосування поведінкових дерев у сучасній ігровій розробці як ефективного інструменту моделювання штучного інтелекту. Запропонований підхід може бути розширений шляхом ускладнення умов прийняття рішень, інтеграції систем оцінювання стану середовища або поєднання з іншими моделями ігрового ШІ, що становить перспективу подальших досліджень.

Список використаних джерел:

1. Buckland M. Programming Game AI by Example. Sudbury: Jones & Bartlett, 2005. URL: https://www.oreilly.com/library/view/programming-game-ai/9781556220784/
2. Champandard A. Behavior Trees for Next-Gen Game AI // Game Developers Conference (GDC). URL: https://www.gdcvault.com/play/1018040/Behavior-Trees-for-Next-Gen
3. Colledanchise M., Ögren P. Behavior Trees in Robotics and AI: An Introduction. Boca Raton: CRC Press, 2018. DOI: 10.1201/9780429489105
4. Cui Y. The exploring of AI applications in game development. 2025. DOI: 10.54254/2755-2721/2025.TJ23324
5. Iovino M., Scukins E., Styrud J., Ögren P., Smith C. A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems*. 2022. DOI: 10.1016/j.robot.2022.104096
6. Kacprzyk S., Hutsenko V. Comparison of artificial intelligence models used in computer games on the Unity platform. *Journal of Computer Sciences Institute*. 2023. DOI: 10.35784/jcsi.6471
7. Millington I., Funge J. Artificial Intelligence for Games. 2nd ed. Boca Raton: CRC Press, 2009. URL: https://www.routledge.com/Artificial-Intelligence-for-Games/Millington-Funge/p/book/9780123747310
8. Rabin S. (Ed.). Game AI Pro: Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, 2013. URL: https://www.routledge.com/Game-AI-Pro-Collected-Wisdom-of-Game-AI-Professionals/Rabin/p/book/9781466565975
9. Rabin S. (Ed.). Game AI Pro 2: Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, 2015. URL: https://www.routledge.com/Game-AI-Pro-2-Collected-Wisdom-of-Game-AI-Professionals/Rabin/p/book/9781482254792
10. Świechowski M., Lewiński D., Tyl R. Combining Utility AI and MCTS towards creating intelligent agents in video games // Proc. IEEE Symposium Series on Computational Intelligence (SSCI). 2021. DOI: 10.1109/SSCI50451.2021.9660170

References:

1. Buckland, M. (2005). Programming Game AI by Example. Sudbury: Jones & Bartlett, Retrieved from: https://www.oreilly.com/library/view/programming-game-ai/9781556220784/
2. Champandard, A. Behavior Trees for Next-Gen Game AI. Game Developers Conference (GDC). Retrieved from: https://www.gdcvault.com/play/1018040/Behavior-Trees-for-Next-Gen
3. Colledanchise, M., Ögren, P. (2018). Behavior Trees in Robotics and AI: An Introduction. Boca Raton: CRC Press, DOI: 10.1201/9780429489105
4. Cui, Y. (2025). The exploring of AI applications in game development. DOI: 10.54254/2755-2721/2025.TJ23324
5. Iovino, M., Scukins, E., Styrud, J., Ögren, P., Smith, C. (2022). A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems*. DOI: 10.1016/j.robot.2022.104096
6. Kacprzyk, S., Hutsenko, V. (2023). Comparison of artificial intelligence models used in computer games on the Unity platform. *Journal of Computer Sciences Institute*. DOI: 10.35784/jcsi.6471
7. Millington, I., Funge, J. (2009). Artificial Intelligence for Games. 2nd ed. Boca Raton: CRC Press, Retrieved from: https://www.routledge.com/Artificial-Intelligence-for-Games/Millington-Funge/p/book/9780123747310
8. Rabin, S. (Ed.). (2013). Game AI Pro: Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, Retrieved from: https://www.routledge.com/Game-AI-Pro-Collected-Wisdom-of-Game-AI-Professionals/Rabin/p/book/9781466565975
9. Rabin, S. (Ed.). (2015). Game AI Pro 2: Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, Retrieved from: https://www.routledge.com/Game-AI-Pro-2-Collected-Wisdom-of-Game-AI-Professionals/Rabin/p/book/9781482254792
10. Świechowski, M., Lewiński, D., Tyl, R. (2021). Combining Utility AI and MCTS towards creating intelligent agents in video games. Proc. IEEE Symposium Series on Computational Intelligence (SSCI). DOI: 10.1109/SSCI50451.2021.9660170

Дата першого надходження статті до видання: 20.03.2026
Дата прийняття статті до друку після рецензування: 17.04.2026
Дата публікації (оприлюднення) статті: 30.05.2026