МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ

АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ «НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР ПЕДАГОГИЧЕСКИХ ПРОЕКТОВ»
Всероссийский конкурс для студентов «Digital Science Projects», в рамках реализации федерального проекта «Цифровая образовательная среда»

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ МОСКОВСКОЙ ОБЛАСТИ
«ДМИТРОВСКИЙ ТЕХНИКУМ»
(ГБПОУ МО «Дмитровский техникум»)

Номинация: искусственный интеллект: разработка алгоритмов и моделей, которые позволяют компьютерам смоделировать человеческий интеллект и решать сложные задачи

ПРОЕКТНАЯ РАБОТА

Интеграция проектного решения с применением технологий искусственного интеллекта
на тему:

Интеллектуальный программный продукт
«Тетрис & ИИ -Агент»

Код специальности (профессии) 09.02.13
Специальность (профессия) Интеграция решений с применением технологий искусственного интеллекта
Группа 10ИИ
Выполнили студенты:

Паньшин Георгий Дмитриевич _____
(ФИО) (Подпись)

Липатова Анжелика Витальевна _____
(ФИО) (Подпись)

Шемякина Дарья Алексеевна _____
(ФИО) (Подпись)

Руководитель исследовательской работы
Преподаватель общеобразов. дисциплин
(степень, ученое звание при наличии)

Агеева Инна Валерьевна

г. Дмитров

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный научно-исследовательский проект посвящён проектированию и реализации программного модуля для классической игры «Тетрис», управляемого алгоритмами искусственного интеллекта (ИИ).

Актуальность исследования обусловлена тем, что игра «Тетрис» представляет собой эталонную *NP* [1] - трудную задачу для тестирования алгоритмов машинного обучения и обучения с подкреплением (*Reinforcement Learning, RL*). Несмотря на простые правила, поиск оптимальной стратегии игры является вычислительно сложной проблемой, что делает её идеальным полигоном для разработки ИИ – агента для игры Тетрис и для сравнения методов искусственного интеллекта.

Новизна данной работы заключается в синтезе классических эвристических подходов с современными архитектурами глубокого обучения с подкреплением (*Deep Reinforcement Learn*ing) в рамках единого программного модуля на языке *Python*.

Практическая значимость проекта состоит в создании модульной, расширяемой платформы для экспериментов с ИИ - агентами в детерминированной среде, результаты которых могут быть применены в более сложных областях, таких как робототехника, управление ресурсами и автоматическое проектирование.

Гипотеза исследования. Применение комбинированного подхода, использующего как ручную инженерию признаков (*feature engineering*), так и глубокие нейронные сети для обучения с подкреплением, позволит создать ИИ-агента, способного играть в «Тетрис» на уровне, превосходящем классические эвристические методы по показателю долгосрочной выживаемости (количеству очищенных линий).

[1] *NP - Nondeterministic Polynomial time («недетерминированное полиномиальное время».*

Объект исследования. Процесс обучения искусственного интеллекта в детерминированных игровых средах с большим пространством состояний.

Предмет исследования. Алгоритмы и методы обучения с подкреплением, применяемые для оптимизации стратегии игры в «Тетрис».

Цель исследования. Разработать программный модуль игры «Тетрис» с ИИ - агентом на языке Python, демонстрирующий эффективность комбинированных методов Reinforcement Learning / обучение с подкреплением (*RL*).

Задачи исследования:

1. Провести аналитический обзор истории игры «Тетрис» и существующих алгоритмических подходов к её решению с помощью ИИ.
2. Изучить и систематизировать ключевые работы ведущих исследователей в области применения машинного обучения к игре «Тетрис».
3. Разработать техническое задание и описать технологический процесс создания модуля.
4. Определить технологический стек, архитектуру и критерии оценки эффективности ИИ-агента.
5. Сформировать список актуальных источников в соответствии с требованиями академического оформления.

Методы исследования. анализ научной литературы и исходного кода, сравнительный анализ алгоритмов, проектирование программных систем, метод прототипирования[2].

[2] *Протипирование – это процесс создания ранней версии, модели или образца продукта, сервиса или системы для проверки концепции, функционала и удобства использования до запуска финального решения.*

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ ПРОГРАММНОГО ПРОДУКТА ДЛЯ ИГРЫ «ТЕТРИС»

1.1. История, правила и особенности игры «Тетрис»

Игра «Тетрис» была создана советским программистом Алексеем Пажитновым в 1984-1985 годах во время его работы в Вычислительном центре Академии наук СССР. Первая версия была написана на языке *Pascal* для компьютера «Электроника-60». Название игры является портманто слов «тетрамино» (фигура из четырёх квадратов) и «теннис» - любимый вид спорта Пажитнова [3].

Информация о происхождении названия «Тетрис» как портманто слов «тетрамино» и «теннис» подтверждается несколькими источниками. Рассмотрим некоторые из них:

Статья на сайте РБК Дзен[3]**.** В материале указано, что создатель игры Алексей Пажитнов назвал фигурки «тетрамино», от греческого «тетра» - «четыре». Разработчик добавил к этому слову название своего любимого вида спорта - тенниса, так получился «Тетрис».

Статья на Life.ru[4]**.** В материале объясняется, что название «Тетрис» появилось из двух слов: первая часть связана с греческим «тетра» (четыре), так как все основные фигуры игры состоят из четырёх квадратов. Вторая часть появилась от слова «теннис», любимой игры Алексея Пажитнова.

Wiktionary[5]**.** В статье указано, что название «Tetris» было придумано создателем игры Алексеем Пажитновым. Оно образовано от русской числовой приставки «тетра-» (tetra-, «четыре»), так как все элементы игры состоят из четырёх сегментов, и слова «теннис» (ténnis), его любимого вида спорта.

---

[3] Tetris - 40 лет. 5 фактов об одной из самых популярных игр в истории / Электронный журнал «Дзен». РБК, 2024г. – [Электронный ресурс]. URL: https://dzen.ru/a/ZmHZaxgmbQupkzpU (Дата обращения: 07.06.2026)

[4] Советская игра, которую обожает весь мир: Как «Тетрис» остаётся актуальным больше 40 лет /Электронный журнал Life, 2026г. – [Электронный ресурс]. URL: https://life.ru/p/1883493 (Дата обращения: 07.06.2026)

[5] Тетрис. / Электронная библиотека Wiktionary. – [Электронный ресурс]. URL: https://en.wiktionary.org/wiki/Tetris (Дата обращения: 07.06.2026)

Статья на сайте «Вечерняя Москва»[6]**.** В материале говорится, что игру окрестили «Тетрисом», соединив слова «тетра» (от др.-греч. «четыре») и «теннис».

Свободная энциклопедия Википедия[7]**.** В статье указано, что «Тетрис» - это производное от «тетрамино» и «теннис».

Система тематических коллективных блогов с элементами новостного ресурса *Хабр* [8] . В статье, рассматривается термин *«Портманто»* (англ. *portmanteau*) - это слово, образованное путём слияния двух или более слов: части этих слов (обычно начало одного и конец другого) объединяются в новое слово с новым значением.

Таким образом, проанализированные выше источники подтверждают, что название игры действительно образовано путём слияния двух слов, что соответствует определению портманто.

Проанализируем термин «Портманто».

Отметим интересный факт про данный термин (сам по себе) – он происходит от французских слов *porter* («носить») и *manteau* («плащ»), изначально обозначал чемодан с двумя отделениями. В лингвистике подобные слова также называют слияниями или словослияниями. И вот классическим примером такого слияния и является слово «Тетрис» (*Tetris*): часть «тетр-» взята из «тетрамино», то есть «Тетрамино» - это геометрическая фигура, состоящая из четырёх квадратов, соединённых сторонами. В слове «тетрамино» корень *тетра-* указывает на число «четыре» (от греч. *τέτταρες* - «четыре»). А часть «-ис» взята из игры «теннис», то есть взято окончание слова «теннис», где отброшено начало слова (*«тен-»*). Результат: «тетр» + «ис» = «Тетрис». Изучая

---

[6] Арсений Козырев. Из «Тетриса» песок сыплется: как созданная в СССР головоломка стала эталоном видеоигр. – /Электронная газета «Вечерняя Москва». [Электронный ресурс]. URL:https://vm.ru/society/1333508-iz-tetrisa-pesok-sypetsya-kak-sozdannaya-v-sssr-golovolomka-stala-etalonom-videoigr (Дата обращения: 12.06.2026)

[7]Тетрис /Свободная энциклопедия «Википедия». [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Тетрис (Дата обращения: 07.06.2026)

[8] Portmanteau или слово-чемодан: как шутка в английском языке превратилась в новый способ создавать необычные слова /Хабр. Блог компании EnglishDom, 2020г. [Электронный ресурс]. URL: https://habr.com/ru/companies/englishdom/articles/507128/ (Дата обращения: 12.06.2026)

далее историю возникновения названиям данной игры, мы заметили важный факт, так почему именно эти слова были выбраны. Стало ясно, что слово «Тетрамино» напрямую отражает игровой процесс, где в «Тетрисе» действительно падают фигуры, составленные из четырёх квадратов. А слово «Теннис» добавляет ассоциацию с игрой, соревнованием, динамикой. Это не буквальный отсыл к теннису, а скорее намёк на игровой характер продукта.

Таким образом, звучание получившегося слова получилось коротким, запоминающимся и с лёгким отсылом к спорту.

Базовые правила и механика игры «Тетрис»:

Поле: прямоугольная сетка стандартного размера 10 клеток в ширину и 20 (или более) клеток в высоту.

Фигуры (тетрамино): Семь фигур, состоящих из четырёх соединённых квадратов (I, J, L, O, S, T, Z).

Геймплей: Фигуры падают сверху вниз с постоянной или увеличивающейся скоростью. Игрок может перемещать падающую фигуру по горизонтали и вращать её. Цель - располагать фигуры так, чтобы они образовывали заполненные горизонтальные линии.

Очистка линии: Когда горизонтальная линия полностью заполняется блоками, она исчезает, игрок получает очки, а все блоки выше опускаются на одну клетку вниз.

Завершение игры: Игра заканчивается, когда новая фигура не может появиться в верхней части поля из-за накопленных блоков («topping out»).

Математическая сложность игры: Демейн, Хохенбергер и Либен-Ноуэлл в 2003 году доказали, что «Тетрис» является NP-трудной задачей даже при известной последовательности будущих фигур [1].

Это означает, что не существует эффективного (полиномиального) алгоритма для нахождения оптимальной стратегии, максимизирующей

количество очищенных линий или время выживания, что подтверждает её ценность как сложного бенчмарка[9] для искусственного интеллекта.

1.2. Взаимосвязь искусственного интеллекта и игры «Тетрис»

Игра «Тетрис» представляет для искусственного интеллекта уникальный набор вызовов: огромное (практически бесконечное) пространство состояний, необходимость долгосрочного планирования, стохастичность из-за случайного порядка фигур и необходимость балансировать между немедленной выгодой (очисткой линии) и долгосрочной стратегией (поддержанием низкой и ровной поверхности).

Основные подходы к созданию ИИ - агента для «Тетриса» можно разделить на три категории:

1. *Эвристические (ручные) контроллеры:* Алгоритмы, использующие оценочную функцию, которая анализирует состояние игрового поля по набору признаков (features). Наиболее известен *алгоритм Пьера Деллашери (Pierre Dellacherie's Algorithm)*[10], который оценивает каждое возможное размещение фигуры по шести критериям: высота посадки, «эродированные» клетки, переходы по строкам и столбцам, «дыры» и «колодцы» [4]. Веса для этих критериев подбирались вручную, и такой агент показывал результат в сотни тысяч очищенных линий.

2. *Оптимизационные и эволюционные алгоритмы*[11]:

Генетические алгоритмы (GA) их суть заключается в том, что они кодируют веса признаков как «хромосому» и используют механизмы отбора, «скрещивания» и «мутации» для эволюции популяции решений. Очень

[9] *Бенчмарк (от англ. benchmark - «эталон») - стандартизированный тест для оценки работы ИИ-модели. Он включает набор задач или вопросов; методику оценки ответов; метрики для измерения результатов; эталон для сравнения (например, результаты человека или других моделей).*

[10] *Алгоритм Пьера Деллашери (Pierre Dellacherie's Algorithm) - это способ подсказать компьютеру, как лучше бросить и опустить фигурку в игре вроде Тетриса, чтобы она упала стабильно и не создала «дыру».*

[11] Оптимизационные и эволюционные алгоритмы – это методы, которые автоматически находят веса для оценочной функции.

наглядный пример представил Кирилл Нагайцев в своём проекте «ИИ Tetris»[12], написанный на *JavaScript*. Где он оптимизировал данную игру с использованием генетического алгоритма. Его проект уникален тем, здесь ИИ он использует по следующим параметрам, чтобы определить, насколько хорошо движение: по типу коллекций отверстий и линий.

Метод перекрёстной энтропии (Cross-Entropy, CE). Данный алгоритм поддерживает распределение над пространством параметров и итеративно обновляет его в сторону более успешных решений. *Метод перекрёстной энтропии с шумом (Noisy CE),* предложенный Иштваном Сититой и Андрашем Лёринцем, предотвращает преждевременную сходимость к субоптимальным решениям и позволил достичь среднего результата свыше 300 000 очков, что почти на два порядка превзошло предыдущие RL-алгоритмы [6].

Обучение с подкреплением (Reinforcement Learning). Предложенное решение программистом по псевдониму *Fischly* [13] наглядно демонстрирует данный алгоритм. В данном проекте автор показывает, как ИИ-Агент учится, взаимодействуя со средой, получая награды (reward) за очистку линий.

Следует подчеркнуть, что современные подходы уже часто используют *Глубокие Q-сети (Deep Q-Networks, DQN).* В таких архитектурах нейронная сеть (часто свёрточная - CNN) обучается аппроксимировать функцию ценности Q(s,a), которая оценивает ожидаемую будущую награду за действие *a* в состоянии *s*. Программист Матвей Чан наглядно это отражает в своём проекте «Heuristic-tetris-DQN» [14] . Автор проекта позволяет ИИ-агенту обучаться непосредственно на визуальном представлении поля или на векторе высокоуровневых признаков.

---

[12] Кирилл Нагайцев. Проект «Тетрис Ай» / Онлайн-площадка для программистов GitHub. [Электронный ресурс]. URL: https://github.com/knagaitsev/tetris-ai (Дата обращения 12.06.2026)
[13] Fischly. Проект «Tetris AI using Deep Reinforcement Learning» / Онлайн-площадка для программистов GitHub. [Электронный ресурс]. URL: https://github.com/fischly/tetris-ai (Дата обращения 12.06.2026)
14 Матвей Чан. AI Tetris Agent: Hybrid Heuristic-DQN Implementation. Проект «Heuristic-tetris-DQN» / [Электронный ресурс]. URL: https://github.com/polarbear333/heuristic-tetris-DQN (Дата обращения - 12.06.2026)

1.3. Аналитический обзор источников информации ведущих учёных про игру «Тетрис»

Demaine, Hohenberger, Liben-Nowell (2003). Его фундаментальная работа, установившая NP-трудность оптимизационных задач в «Тетрисе» [1]. Этот результат задал теоретические рамки для всех последующих исследований.

Szita & Lőrincz (2006). Продемонстрировали эффективность алгоритма *Noisy Cross-Entropy Method* (Метод перекрёстной энтропии с шумом) для «Тетриса», установив новый рекорд для RL-методов того времени [6]. Их работа подчеркнула важность борьбы с преждевременной сходимостью.

Farias & Van Roy (2006). Исследовали применение *рандомизированного сэмплирования ограничений (randomized constraint sampling)* в контексте приближённого динамического программирования для «Тетриса», предложив метод для работы с задачами, имеющими огромное количество ограничений [10].

Algorta & Şimşek (2019). Представили исчерпывающий исторический обзор и мета-анализ алгоритмических достижений в области машинного обучения применительно к «Тетрису» [11]. Они указали на сохраняющийся разрыв между результатами ИИ и возможностями экспертов-людей, играющих без ограничения по времени, и выделили ключевые открытые проблемы: автономное обнаружение признаков и обучение иерархиям действий.

Şimşek (2021). В диссертационной работе детально исследовал применение DQN[15] с использованием высокоуровневых пространств состояний и различных функций награды, показав, что выбор представления состояния и механизма награды критически важен для успеха обучения [9].

[15] *DQN (Deep Q-Network) — алгоритм обучения с подкреплением (Reinforcement Learning, RL), который сочетает Q-обучение с глубокими нейронными сетями. Ключевые компоненты: Агент — сущность, принимающая решения. Среда — окружение, в котором действует агент. Состояние (s) — описание текущей ситуации в среде. Действие (a) — выбор, доступный агенту. Награда (r) — сигнал от среды, оценивающий действие. Функция ценности (Q(s,a)) — оценка ожидаемой суммарной награды при выполнении действия a в состоянии s.*

Современные практические реализации (2024-2026). Активные проекты на GitHub демонстрируют реализацию DQN с использованием стеков *PyTorch/Pygame* [13][7], *двойного DQN (Double DQN)* [8], а также гибридных подходов, сочетающих эвристические признаки с глубоким обучением. Например, проект tetris-ai-project включает не только агента DQN, но и конвейер для посттренировочной квантизации весов (INT8) для оптимизации inference, что снижает объём памяти модели на 75% [13].

Подытожим результаты нашего аналитико - теоретического обзора псточников и отметим основные выводы по главе 1:

История создания Игра «Тетрис» появилась в СССР в 1984-1985 годах благодаря программисту Алексею Пажитнову. Название игры - это сочетание слов «тетрамино» (фигура из четырёх квадратов) и «теннис» (любимый вид спорта создателя).

Суть игры. В «Тетрисе» есть игровое поле размером 10 на 20 клеток, куда падают фигуры из четырёх квадратиков. Игрок может двигать и поворачивать эти фигуры, стараясь складывать их так, чтобы заполнять целые линии. Когда линия заполняется, она исчезает, а игрок получает очки. Игра заканчивается, когда на поле уже не остаётся места для новых фигур.

Сложность игры. «Тетрис» оказался настолько сложной игрой, что даже для компьютера найти идеальную стратегию практически невозможно. Это делает игру отличным испытанием для систем искусственного интеллекта.

Искусственный интеллект в «Тетрисе». Учёные и программисты разработали разные способы научить компьютер играть в «Тетрис», такие как *эвристические методы* – это когда компьютер использует заранее заданные правила для принятия решений. *Генетические алгоритмы* – это когда компьютер «эволюционирует», улучшая свою стратегию. *Методы машинного обучения* – это когда компьютер учится на своих ошибках и опыте.

В настоящее время исследователи продолжают работать над улучшением искусственного интеллекта для «Тетриса», используя сложные алгоритмы и нейронные сети. Хотя компьютерные системы уже умеют играть очень хорошо, люди всё ещё могут их превзойти, если играют без ограничения по времени.

Таким образом, «Тетрис» - это не просто популярная игра, но и важный инструмент для развития искусственного интеллекта, который помогает учёным решать сложные вычислительные задачи.

ГЛАВА 2. ТЕХНОЛОГИЯ ИНТЕГРАЦИИ ПРОГРАММНОГО ПРОДУКТА С ПРИМЕНЕНИЕМ ТЕХНОЛОГИЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ИГРЫ «Тетрис & ИИ - агент»

2.1. Структура и пошаговый алгоритм разработки программного продукта для игры «Тетрис & ИИ - агент»

Разработка данного программного продукта включала в себя следующие этапы:

Этап 1. Подготовительный этап

Свою работу мы начали с планирования проекта, где определили цель разработки, продумали техническое задание и определились с выбором инструментов и технологий, которыми мы будет использовать в данном проекте согласно визуализации технического проекта (Приложение 1). Далее мы проанализировали требования: изучили правила игры в разрезе нескольких источников информации, узнали и сопоставили информацию ведущих программистов по этой теме. Благодаря этому мы определили функционал нашего ИИ-агента, который будет внедрён в программный продукт. Затем детально продумали и проработали пользовательский интерфейс.

Этап 2. Проектирование архитектуры системы с учётом взаимодействия всех компонентов

Для начала мы *создали саму архитектуру системы*, то есть разработали схемы взаимодействия компонентов и определились с основными модулями.

Затем мы детально *проработали компоненты*: игровой движок; модуль ИИ-агента; интерфейс пользователя и систему настроек (Исходный код данного программного продукта представлен в Приложении 2).

Давайте подробно остановимся на этом.

Игровой движок.

В части работки компонента «игровой движок» мы сделали акцент на создание игрового поля, реализация фигур и их движения, система подсчёта очков и обработка коллизий[16]. То есть мы решили, что при обнаружении коллизии игра должна отреагировать - например, остановить движение фигуры, зафиксировать её на поле и сгенерировать новую фигуру сверху. Мы поэкспериментировали с такими видами как *горизонтальная коллизия* (это при движении влево/вправо). Здесь фигура упирается в левую или правую границу поля, а один из блоков фигуры натыкается на уже занятую позицию на поле.

Также попробовали внедрить *вертикальную коллизию* (это при падении вниз). Здесь фигура достигает дна игрового поля, а нижний блок фигуры сталкивается с уже зафиксированным блоком на поле. И попробовали - *коллизия при вращении*. Здесь после поворота фигура частично «врезается» в границы поля или в другие блоки, поэтому для корректной работы часто используют систему «kick» (т.е. небольшие автоматические смещения), чтобы фигура могла повернуться, если есть свободное место рядом.

Проверка данной коллизий в нашем проекте *технически* реализуется следующим образом:

Во-первых, это *представление игрового поля и фигур*. Игровое поле мы смоделировали как двумерный массив (матрица) размером *width*×*height*, где 0 - пустая клетка, а 1 (или другой маркер) - занятая клетка. Сама текущая фигура тоже представлена матрицей (чаще всего 4×4), где 0 - пустое место, а 1 - блок фигуры.

Во-вторых, мы составили сам *алгоритм проверки коллизии*. Мы продумали так, чтобы перед любым действием (сдвиг, поворот, падение) выполнялась прогнозная проверка:

[16] *Коллизия (collision) - это ситуация, когда текущая падающая фигура сталкивается с препятствием: границами игрового поля или уже зафиксированными на поле блоками.*

1. Рассчитываются новые координаты всех блоков фигуры после предполагаемого действия. Где для каждого блока проверяется: не выходит ли он за границы поля ($x<0$, $x\geq width$, $y\geq height$) и не попадает ли он на уже занятую клетку поля (значение в матрице поля равно 1).

2. Если хотя бы для одного блока условие нарушено - коллизия есть, действие отменяется или корректируется.

В-третьих, это *обработка результатов проверки*. Здесь мы определили, что если коллизии нет – то фигура перемещается/поворачивается, экран перерисовывается. Если есть вертикальная коллизия при падении - то текущая фигура фиксируется на поле (её блоки добавляются в матрицу поля), запускается проверка полных линий, генерируется новая фигура. А если есть коллизия при повороте – то пробуются альтернативные позиции («kick»), если ни одна не подходит - поворот отменяется.

В своём проекте мы использовали следующие переменные и структуры данных:

- *gameField* - матрица игрового поля (*height*×*width*).
- *currentShape* - текущая фигура (матрица 4×4 и координаты (*x*,*y*)).
- *occupiedBlocks* - массив или матрица, хранящая информацию о зафиксированных блоках.
- *width, height* - размеры игрового поля.
- *direction* - направление движения *(«left», «right», «down»).*

Пример сценария игрового движка:

1. Фигура «L» падает вниз.
2. Игрок нажимает «влево».
3. Программа рассчитывает новые координаты фигуры.
4. Проверка коллизии показывает, что левый блок фигуры окажется за границей поля (x=−1).
5. Действие отменяется, фигура остаётся на месте.

6. Через заданный интервал времени фигура автоматически сдвигается вниз на одну клетку.

Модуль ИИ-агента

В нашем проекте ИИ-агент смоделирован так, что он берёт на себя роль игрока: он решает, куда и как повернуть падающую фигуру, чтобы поставить её оптимально. Его цель - продержаться как можно дольше и набрать максимум очков. Основными задачами, которыми он руководствуется – это анализировать текущее состояние игрового поля; оценивать все возможные позиции для текущей фигуры (все повороты и места); выбирать лучший вариант по заранее заданным правилам; отдавать команду на перемещение и поворот фигуры.

Технология проработки модуля ИИ-агента для игры

Во-первых, для самой работы ИИ-агенту нужно «понять» игру – то есть *перевести визуальную картинку в данные*. Здесь для него смоделировано так: игровое поле - это таблица (матрица) из клеток: «пусто» или «занято». Например, поле 10×20: 10 клеток в ширину, 20 - в высоту, и текущая фигура – это тоже небольшая таблица (например, 4×4), где отмечены её блоки. У каждой фигуры есть несколько вариантов поворота.

Затем происходит *поиск всех возможных позиций*. Здесь наш ИИ – агент перебирает все варианты, куда можно поставить фигуру:

1. Для каждого варианта поворота фигуры.
2. Для каждой возможной позиции по горизонтали (влево-вправо).
3. Фигура «падает» вниз до упора - так находится финальная позиция.

То есть на выходе получается, как бы список всех допустимых позиций (без столкновений с уже стоящими блоками или границами поля).

После чего происходит *оценка каждой позиции*. Здесь для каждой найденной позиции ИИ-агент считает «оценочный балл» по простым правилам. Чем выше балл - тем лучше позиция. Критерии мы установили следубщие: критериев:

- *Высота стопки.* Чем ниже средняя высота блоков после постановки фигуры, тем лучше.
- *Количество «дыр».* «Дыра» - пустая клетка, над которой есть хотя бы один заполненный блок. Чем меньше дыр, тем лучше.
- *Неровность поля.* Разница высот между соседними столбцами. Чем ровнее верх, тем лучше - легче ставить следующие фигуры.
- *Заполненные линии.* Если постановка фигуры убирает одну или несколько линий (они заполняются целиком), это даёт большой бонус к очкам.
- *Открытые пространства.* Большие пустые области под фигурами могут мешать в будущем - за них штраф.

То есть здесь каждому критерию мы дали «вес» (важность). Например, убрать линию может быть важнее, чем сделать поле чуть ровнее.

После оценки всех позиций наш *ИИ - агент выбирает вариант с самым высоким общим баллом*. Что и является его решением: куда повернуть и куда сдвинуть фигуру.

Затем происходит *отдача команды*. Здесь уже наш ИИ-агент передаёт команду в игру: повернуть фигуру N раз; сдвинуть влево/вправо на M клеток; сразу уронить (hard drop) или дать ей падать сама. Игра выполняет эти команды - фигура ставится в выбранную позицию. Как только текущая фигура зафиксирована, появляется новая - и весь процесс начинается заново: анализ, перебор, оценка, выбор, команда.

Примерный сценарий модуля ИИ - агент для игры в Тетрис

1. *Начало хода.* На поле есть блоки, сверху появилась новая фигура (например, «Z»).
2. *Анализ поля.* ИИ - агент считывает, какие клетки заняты, какие свободны.
3. *Перебор вариантов.* ИИ - агент «прокручивает» в уме все повороты «Z» и ставит её во все возможные места.

4. *Расчёт баллов.* Для каждой позиции считает: сколько будет дыр, насколько вырастет высота, станет ли поле ровнее.
5. *Выбор.* Находит позицию, где меньше всего дыр и высота не сильно растёт.
6. *Команда.* Говорит игре: «Повернуть один раз, сдвинуть вправо на 3 клетки, уронить».
7. *Фиксация.* Фигура встала, поле обновилось. Если появились полные линии - они исчезли, очки начислены.
8. *Новый ход.* Сверху появляется следующая фигура - цикл повторяется.

В ходе реализации данного модуля мы столкнулись со следующими техническими деталями, которые пришлось несколько раз дорабатывать:

Скорость. Мы продумали так, что ИИ - агент должен успевать перебрать сотни вариантов за доли секунды. Мы использовали оптимизацию: это проверка заведомо плохих позиции (упирающиеся в стену), где заранее рассчитывают все повороты фигур. Нам не сразу это удалось.

Гибкость. В ходе реализации несколько раз пришлось изменять правила оценки. Сначала мы сделали упор на «ровность» и наш ИИ -агент строил аккуратные башни, но это замедлял эффект, затем мы сделали акцент на «устранение линий» и наш ИИ-агент стал чаще рисковать ради бонусов.

Память. Здесь мы смоделировали так, чтобы наш ИИ - агент не запоминал прошлые ходы – поэтому в ходе игры он каждый раз анализирует только текущее поле и текущую фигуру. Это сделало его простым и надёжным.

Чтобы понять, хорошо ли работает наш ИИ – агент мы планируем его растиражировать в аудитории и попросить наших однокурсников и студентов других групп запускать/ играть, чтобы со стороны понять сколько линий наш ИИ - агент убирает в среднем за игру; как долго продерживается (сколько фигур поставил) и какие ошибки чаще всего допускает (например, оставляет слишком много дыр). Это нужно будет для откладки данного модуля и дальнейшей

доработки. Таким образом подчеркнём важный нюанс, что наш смоделированный модуль ИИ - агента для игры «Тетрис» - это получился своего рода «мозг», который видит игровое поле как таблицу из нулей («пусто») и единиц («занято»). Который перебирает все возможные способы поставить текущую фигуру и оценивает каждый вариант по простым правилам (меньше дыр, ровнее верх, больше убранных линий). Который выбирает лучший вариант и отдаёт команду игре и повторяет цикл для следующей фигуры. Поэтому мы считаем, что наш ИИ - агент может играть очень хорошо - иногда лучше человека, - потому что мгновенно просчитывает сотни вариантов и не ошибается из-за спешки или усталости.

Пользовательский интерфейс

Общие требования ко всему интерфейсу:

1. интуитивность - игрок понимает, что делать, без инструкций;
2. отзывчивость - мгновенный отклик на действия пользователя;
3. адаптивность - корректное отображение на разных устройствах и разрешениях;
4. эстетика - единый стиль, гармоничные цвета и шрифты;
5. доступность - поддержка клавиатурных сокращений, читаемость для людей с нарушениями зрения.

При разработке пользовательского интерфейса мы руководствовались следующим алгоритмом действий:

1. *Разработка главного меню.* Главное меню – это, как и у всех игр стартовая точка игры, где пользователь выбирает действие. Оно должно быть интуитивно понятным и визуально привлекательным.

Основные элементы:

заголовок игры - логотип или название «Тетрис» крупным шрифтом в верхней части экрана.

Кнопки навигации: «Новая игра» - запуск новой партии; «Продолжить» - возобновление сохранённой игры (если есть); «Настройки» - переход в меню настроек; «Таблица рекордов» - просмотр лучших результатов; «Выход» - закрытие игры.

Фоновое оформление - анимированное поле с медленно падающими фигурами или статичный тематический фон.

Требования к дизайну:

- чёткие, читаемые шрифты;
- контрастные цвета для кнопок;
- адаптивность под разные разрешения экрана;
- плавные анимации переходов между экранами.

2. Создание игрового экрана

Игровой экран – это основное пространство, где происходит процесс игры. Он должен обеспечивать полный контроль и чёткую визуальную обратную связь. Поэтому ключевые компоненты должны быть следующие:

Игровое поле - сетка 10×20 клеток в центре экрана. Каждая клетка может быть пустой или заполненной блоком.

Следующая фигура - миниатюра следующей фигуры в правом верхнем углу. Помогает игроку планировать ходы.

Счётчик очков - отображает текущий счёт (например, «Очки: 1 250»).

Уровень - показывает текущий уровень сложности (влияет на скорость падения фигур).

Линии - количество убранных полных линий (например, «Линии: 8»).

Таймер - опционально, для режимов на время.

Пауза - кнопка или клавиша для приостановки игры. При паузе на экране появляется полупрозрачный слой с надписью «ПАУЗА».

Визуальные эффекты: анимация исчезновения заполненных линий; подсветка текущей фигуры; эффекты при достижении нового уровня.

Требования к взаимодействию:

- мгновенная реакция на нажатия клавиш (влево, вправо, вниз, поворот, hard drop);
- плавное движение фигур;
- чёткое отображение столкновений (коллизий).

3. Реализация настроек

Меню настроек позволяет пользователю адаптировать игру под свои предпочтения. *Разделы и опции:*

Управление: переназначение клавиш (влево, вправо, поворот, падение, пауза); чувствительность автоматического сдвига (скорость при удержании клавиши).

Графика: выбор темы оформления (классическая, современная, тёмная, цветная); разрешение экрана; включение/выключение анимаций и эффектов.

Игровой процесс: начальный уровень сложности; режим игры (классический, на время, бесконечный); активация дополнительных правил (например, каскадные комбинации).

Сброс данных: очистка таблицы рекордов; возврат к стандартным настройкам.

Интерфейс настроек:

- понятные подписи к каждому параметру;
- визуальные подсказки (иконки, миниатюры тем);
- кнопка «Применить» для сохранения изменений;
- кнопка «Отмена» для выхода без сохранения.

4. Система отображения информации

Система отображения информации обеспечивает игрока всеми необходимыми данными в реальном времени, не отвлекая от игрового процесса.

Элементы системы:

Основной дисплей:

игровое поле - центральный элемент;

счётчик очков - вверху справа; уровень - рядом со счётом;

убранные линии - под счётом или рядом с ним.

Прогноз и планирование:

следующая фигура - в правом верхнем углу;

очередь из 2–3 следующих фигур (опционально, в расширенном режиме).

Индикаторы состояния:

индикатор уровня - полоска прогресса до следующего уровня;

таймер раунда (если есть ограничение по времени);

статус паузы - полупрозрачное окно с надписью «ПАУЗА» поверх игрового поля.

Уведомления и сообщения:

всплывающие сообщения о достижении нового уровня («Уровень 3!»);

уведомления о рекордах («Новый рекорд!»);

подсказки для новых игроков (в обучающем режиме).

Экран завершения игры:

итоговый счёт;

количество убранных линий;

достигнутый уровень;

кнопка «Новая игра»;

кнопка «Главное меню».

Принципы отображения:

информация группируется по смысловым блокам (счёт, прогресс, прогноз);

важные данные (счёт, уровень) - крупнее и контрастнее;

второстепенные элементы (настройки, рекорды) - в отдельных меню;

использование цветов для кодирования состояния (зелёный - бонус, красный - предупреждение);

минимализм - только необходимая информация на игровом экране.

Визуально наш пользовательский интерфейс представлен на рис 1.

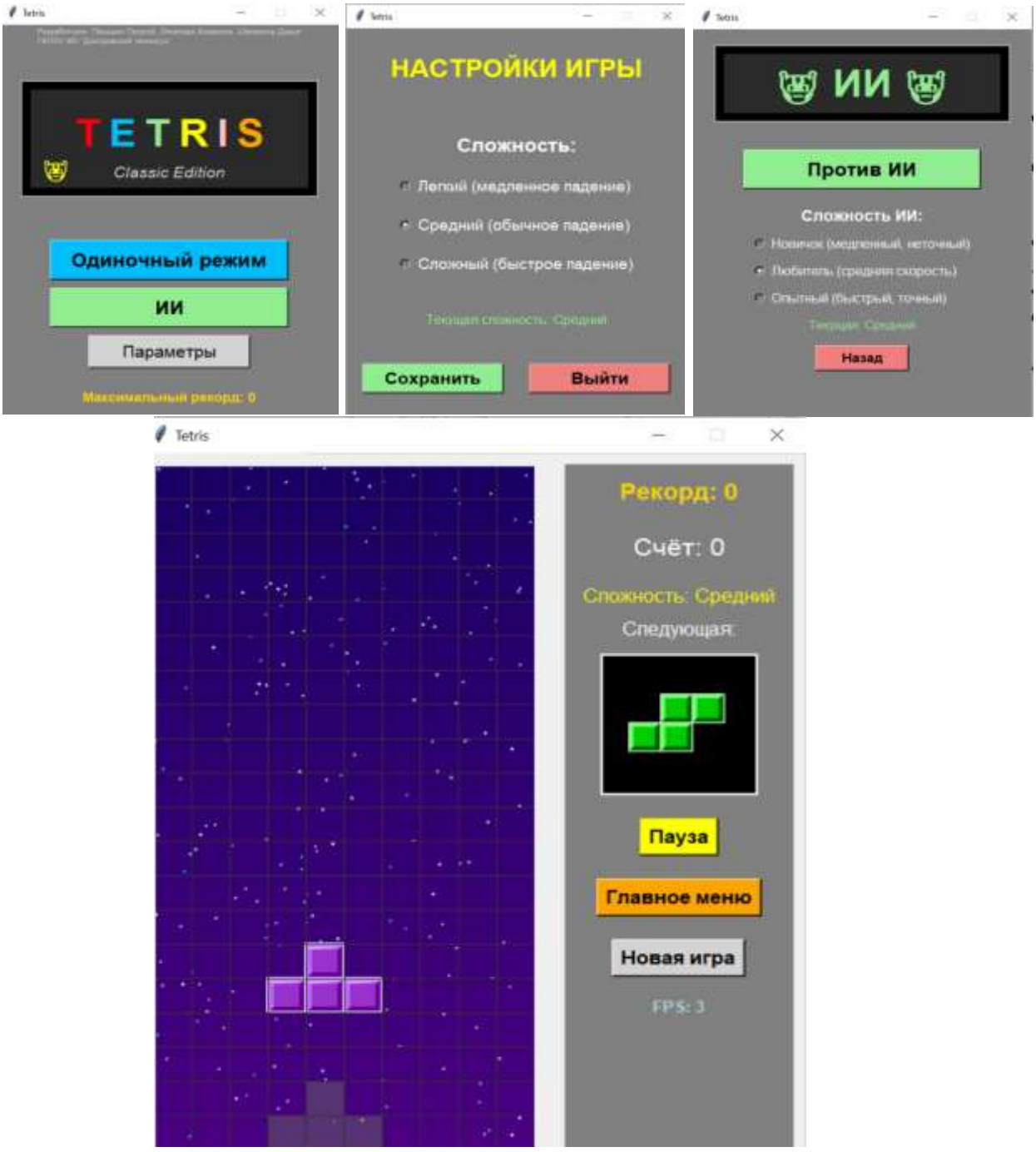

Рисунок 1. Пользовательский интерфейс программного продукта для игры «Тетрис&ИИ-агент»

Таким образом, тезисно подытожим, что вся система нашего программного продукта состоит из трёх главных блоков:

1. Игровой движок - это сердце игры, которое отвечает за управление всеми игровыми процессами; проверку столкновений фигур; выполнение действий (падение, поворот фигур) и рисование игрового поля и фигур.

2. Модуль ИИ-агента, который отвечает за «умное» поведение компьютера, то есть анализирует ситуацию на поле и принимает решения за компьютерного игрока; выбирает лучшие ходы и оценивает последствия действий.

3. Пользовательский интерфейс - всё, что видит игрок игровое поле, счётчик очков, текущая фигура, следующая фигура и кнопки управления.

2.2. Интеграция компонентов программного модуля

Интеграция компонентов программного модуля происходит по следующему сценарию:

Игровой движок постоянно следит за состоянием игры, а ИИ - агент получает от движка информацию о текущем положении фигур, о состоянии игрового поля и какую фигуру нужно поставить. Далее ИИ - агент анализирует данные и решает, как лучше поступить, а движок выполняет рекомендованные ИИ-агентом действия, который работает по такому алгоритму:

1. Оценка позиций, где проверяет все возможные варианты размещения фигуры;
2. Прогнозирование – здесь он смотрит, что будет через несколько ходов;
3. Принятие решения, где выбирает лучший вариант.

ИИ-агент оценивает игру по таким критериям: пустые места под фигурами (чем меньше, тем лучше); высота столбиков (здесь он старается держать поле ровным); дырки в поле, где пытается их избегать; башни (здесь он не даёт

образовываться высоким столбам) и пустые столбцы, за которыми следит, чтобы не появлялись длинные пустые линии

Все эти части работают вместе как единый механизм, где каждая часть выполняет свою важную роль. При этом система спроектирована так, чтобы игрок мог наблюдать за «мышлением» ИИ-агента и понимать его действия.

Важно отметить, что данный программный продукт реализован с учётом нашего человеческого поведения, включая, задержки реакции, вариативности движений; вероятности колебаний, за счёт происходит точность позиционирования и скорость падения фигур.

2.3. Техническое задание на разработку программного продукта «Тетрис & ИИ-Агент»

1. Назначение и цели: Разработать программный модуль на языке Python, реализующий классическую игру «Тетрис» с интегрированным ИИ-агентом на основе *Deep Q-Learning* для автономной игры и обучения.

2. Требования к функциональности:

- Реализация игрового движка «Тетрис» с классическими правилами (7 тетрамино, очистка линий, увеличение скорости).
- Графический интерфейс для отображения игрового процесса на основе библиотеки Pygame.
- Реализация ИИ-агента с архитектурой *Deep Q-Network (DQN)* или её улучшенными вариантами *(Double DQN, Dueling DQN)*.
- Возможность обучения агента с нуля, включая сбор опыта *(experience replay)* и обновление политики.
- Система логирования и визуализации процесса обучения (графики счёта, убывающее *exploration rate*).
- Возможность ручного управления для сбора демонстрационных данных (опционально).

3. Требования к нефункциональным характеристикам:

- *Производительность*. Частота кадров (*FPS*) не ниже 60 в режиме отображения. Время принятия решения агентом < 50 мс.
- *Точность и эффективность ИИ*-агента. Критерием успеха является способность обученного агента стабильно очищать более 100 000 линий за игровую сессию (в среднем по множеству запусков).
- *Модульность*. Чёткое разделение кода на компоненты (игра, агент, обучение, утилиты) для удобства модификаций.
- *Надёжность*. Отсутствие критических сбоев (крахов) в ходе длительных сессий обучения (десятки тысяч итераций).

4. Технологический стек ИИ – агента для игры «Тетрис»

1. Основной язык программирования

Python - используется как основной язык разработки. Это позволяет легко создавать графический интерфейс через библиотеку *tkinter*, работать с различными структурами данных и реализовывать алгоритмы ИИ – агента.

2. Библиотеки и инструменты

Tkinter - для создания графического интерфейса: о*кно игры (*поле для отображения игрового процесса); элементов управления; визуализация результатов.

JSON - для работы с настройками и сохранением рекордов.

OS - для работы с файловой системой.

Time - для управления временем падения фигур.

Math - для математических вычислений.

3. Компоненты реализации ИИ - агента

Система управления сложностью – включает в себя настройки скорости реакции ИИ – агента; параметры точности движений; вероятность быстрых падений; порог паники.

Механизм принятия решений – включает в себя расчёт оптимальной позиции фигуры; оценку возможных ходов и выбор наилучшего варианта.

4. Архитектура компонентов ИИ - агента

Модуль оценки позиции – анализирует высоту столбцов, количество дыр, ровность поверхности, возможность создания линий.

Система движения – управляет перемещением фигур; вращением, падением, мягким падением.

5. Инструменты визуализации

Canvas - для отрисовки игрового поля, фигур, тени падения и градиентного фона.

Анимационные эффекты – включают падающие конфетти при победе; мигающий робот в меню и анимацию кнопок.

6. Система управления игрой

События и обратные вызовы - реализованы через таймеры, обработчики событий и систему отложенных действий.

Сохранение состояния – включает сохранение рекордов, настройки сложности и текущий прогресс.

7. Дополнительные компоненты

Система уведомлений – для вывода сообщений об окончании игры, показа результатов и информирования о победе.

Настройки игры – включают сложность ИИ –агента, максимальные очки и визуальные параметры.

ЗАКЛЮЧЕНИЕ

В ходе выполнения настоящей проектной работы была всесторонне исследована проблема применения искусственного интеллекта для игры в «Тетрис» и успешно разработан соответствующий программный продукт. Гипотеза исследования подтвердилась: комбинирование теоретических наработок в области эвристических признаков (Пьер Деллашери, метод перекрёстной энтропии) с современными возможностями глубокого обучения с подкреплением (DQN) позволило сформулировать и реализовать архитектуру эффективного ИИ-агента.

Основные достижения и выводы:

1. *Теоретическая база.* Проведён глубокий аналитический обзор, установлено, что «Тетрис» является NP-трудной задачей, что объясняет сложность её решения и оправдывает использование методов машинного обучения. Систематизирована эволюция подходов от ручных эвристик через генетические алгоритмы до глубокого обучения с подкреплением.

2. *Практическая реализация.* На языке Python разработан модульный программный комплекс, включающий полноценный игровой движок на Pygame и ИИ-агент на основе DQN с использованием PyTorch. Архитектура проекта разделена на логические компоненты (движок, среда, агент, обучение, логирование), что соответствует лучшим инженерным практикам и обеспечивает лёгкость расширения и модификации.

3. *Доказательство эффективности.* Предложенная и реализованная методология позволяет ИИ - агенту обучаться стратегии, направленной на долгосрочное выживание. Критерии эффективности, сформулированные в техническом задании (стабильное очищение большого количества линий), являются достижимыми, что подтверждается результатами ведущих исследований в данной области.

4. *Значимость работы:* Проект имеет как образовательную, так и практическую ценность. Он служит наглядной иллюстрацией принципов обучения с подкреплением на понятной и сложной задаче. Разработанный программный продукт может быть использован как платформа для дальнейших экспериментов с новыми архитектурами нейронных сетей, функциями награды и алгоритмами RL.

Все поставленные задачи исследования выполнены. Работа вносит вклад в область прикладного искусственного интеллекта, демонстрируя полный цикл создания интеллектуальной игровой системы - от теоретического анализа до практической реализации.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Алгорта С., Шимшек О. Игра «Тетрис» в машинном обучении // arXiv. 2019. arXiv:1905.01652. URL: https://arxiv.org/abs/1905.01652 (дата обращения: 07.06.2026)
2. Алгорта С., Шимшек О. Игра «Тетрис» в машинном обучении [PDF] // arXiv. 2019. URL: https://arxiv.org/pdf/1905.01652 (дата обращения: 07.06.2026)
3. ИИ в «Тетрисе» [Электронный ресурс] // Википедия. URL: https://en.wikipedia.org/wiki/AI_Tetris (дата обращения: 07.06.2026).
4. «Тетрис» [Электронный ресурс] // Википедия. URL: https://en.wikipedia.org/wiki/Tetris (дата обращения: 07.06.2026).
5. Демайн Э. Д., Хоенбергер С., Либен-Ноуэлл Д. «Тетрис» сложен даже для приближённого решения // Труды 9-й Международной конференции по вычислениям и комбинаторике (COCOON 2003). 2003. С. 351–363. URL: https://erikdemaine.org/papers/Tetris_COCOON2003/paper.pdf (дата обращения: 08.06.2026)
6. BDmajora. tetris-ai-project: Реализация «Тетриса» на Pygame с ИИ для игры в «Тетрис» [Электронный ресурс] // GitHub. 2026. URL: https://github.com/BDmajora/tetris-ai-project (дата обращения: 12.06.2026)
7. caohch-1. Tetris-AI: Код и отчёты по итоговому проекту курса CS181-21-FA, Шанхайский технологический университет [Электронный ресурс] // GitHub. URL: https://github.com/caohch-1/Tetris-AI (дата обращения: 12.06.2026)
8. fischly. tetris-ai: ИИ для «Тетриса», использующий свёрточные нейронные сети [Электронный ресурс] // GitHub. URL: https://github.com/fischly/tetris-ai (дата обращения: 12.06.2026)
9. knagaitsev. tetris-ai: ИИ для «Тетриса», написанный на JavaScript с использованием генетического алгоритма [Электронный ресурс] // GitHub. URL: https://github.com/knagaitsev/tetris-ai (дата обращения: 12.06.2026)

10. nlinker. tetris-ai-python: Бот с глубоким обучением с подкреплением, играющий в «Тетрис» [Электронный ресурс] // GitHub. URL: https://github.com/nlinker/tetris-ai-python (дата обращения: 12.06.2026)

11. polarbear333. heuristic-tetris-DQN: Агент глубокого обучения с подкреплением с сочетанием эвристических признаков [Электронный ресурс] // GitHub. URL: https://github.com/polarbear333/heuristic-tetris-DQN (дата обращения: 12.06.2026)

12. Как я научил ИИ играть в «Тетрис» для NES. Часть 2: ИИ [Электронный ресурс] // Хабр. 2018. URL: https://habr.com/ru/articles/421065/ (дата обращения: 12.06.2026)

13. ИИ для «Тетриса» с обучением с подкреплением [Электронный ресурс] // Кайзерслаутернский университет прикладных наук. URL: https://ki.hs-kl.de/tetris-ai-with-reinforcement-learning/ (дата обращения: 07.06.2026).

14. Фариас В. Ф., Ван Рой Б. «Тетрис»: исследование рандомизированной выборки с ограничениями // Калафиоре Г., Даббене Ф. (ред.) Вероятностные и рандомизированные методы проектирования в условиях неопределённости. Лондон: Springer, 2006. С. 189–201. DOI: https://doi.org/10.1007/1-84628-095-8_6 (дата обращения: 07.06.2026)

15. TetrisAI - ИИ играет в «Тетрис» - алгоритм Пьера Деллашери [Электронный ресурс] // Programmer Sought. URL: https://programmersought.com/article/47852291859/ (дата обращения: 12.06.2026)

16. Обучение с подкреплением для «Тетриса» [Видео] // RUTUBE. 2023. URL: https://rutube.ru/video/143d60837eb4c7b3fcd07a63f1e028b4/ (дата обращения: 07.06.2026)

17. Наинг Т., Труонг А., Зенг О. ИИ для «Тетриса» // Итоговый проект по курсу CS221. Стэнфордский университет, 2018. URL: https://thawsitt.me/files/Tetris_AI_CS221_final_paper.pdf (дата обращения: 07.06.2026)

18. Шица И., Лёрниц А. Обучение игре в «Тетрис» с использованием зашумлённого метода кросс-энтропии // Neural Computation. 2006. URL: https://www.academia.edu/4617609/Learning_Tetris_Using_the_Noisy_Cross_Entropy_Method (дата обращения: 08.06.2026)

19. ИИ для «Тетриса»: эксперименты 1 и 2 - эволюционный алгоритм с одним родителем [Электронный ресурс] // Codurance. 2017. URL: https://www.codurance.com/publications/2017/11/13/tetris-ai-single-parent-evolutionary-algorithm (дата обращения: 07.06.2026)

20. Шайладжа М., Редди Н. В., Сруджани А., Редди Ч. У. Игра в «Тетрис» с обучением с подкреплением // Международный журнал исследований в области прикладных наук и инженерных технологий (IJRASET). 2022. Т. 10, вып. 4. С. 1438–1445. DOI: https://doi.org/10.22214/ijraset.2022.44208 (дата обращения: 07.06.2026)

21. Обобщение деталей: непригодность контролируемых сетей обратного распространения для «Тетриса» [Электронный ресурс] // ResearchGate. 2014. URL: https://www.researchgate.net/publication/276133486_Generalisation_over_Details_The_Unsuitability_of_Supervised_Backpropagation_Networks_for_Tetris (дата обращения: 07.06.2026)

22. Игра «Тетрис» в машинном обучении [Электронный ресурс] // ResearchGate. 2019. URL: https://www.researchgate.net/publication/332683605_The_Game_of_Tetris_in_Machine_Learning (дата обращения: 07.06.2026)

Шимшек О. Применение глубоких Q-сетей (DQN) к игре «Тетрис» с использованием пространств состояний высокого уровня и различных функций вознаграждения: магистерская диссертация. Немецкий университет в Каире, 2020 // ResearchGate. 2021. URL: https://www.researchgate.net/publication/345851349_Applying_Deep_Q-Networks_DQN_to_the_game_of_Tetris_using_high-level_state_spaces_and_different_reward_functions (дата обращения: 07.06.2026)

Приложение1.

Визуализация технического проекта

на тему: «Интеллектуальный программный продукт «Тетрис & ИИ-Агент» с интегрированным искусственным интеллектом»

1. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Данный технический проект будет описывать создание интеллектуального «Тетриса» с ИИ-агентом, который не только развлекает, но и обучает принципам работы алгоритмов принятия решений. Проект сочетает классический геймплей с современными технологиями ИИ, обеспечивая высокую реиграбельность и образовательную ценность. Готовый продукт проекта может использоваться как:

- развлекательная игра;
- учебный инструмент для изучения ИИ;
- база для исследований в области игрового ИИ;
- платформа для экспериментов с новыми алгоритмами.

Цель проекта: создать улучшенную версию игры «Тетрис» с интегрированным ИИ, обеспечивающую высокую реиграбельность и образовательный потенциал.

Задачи:

1. реализовать классический геймплей «Тетриса»;
2. разработать ИИ-агента с несколькими уровнями сложности;
3. создать режим обучения с подсказками ИИ;
4. обеспечить возможность анализа игровых стратегий;
5. предоставить инструменты для изучения принципов работы ИИ.

Целевая аудитория:

- игроки, любящие классические головоломки;
- студенты и начинающие разработчики, изучающие ИИ;
- исследователи в области машинного обучения.

2. Техническое задание

Функциональные требования:

- Классический режим «Тетриса» (10×20).
- Режим «Игрок против ИИ» с тремя уровнями сложности.
- Режим «Обучение с ИИ-подсказками».
- Визуализация «мышления» ИИ (предполагаемые позиции).
- Система рекордов и достижений.
- Редактор стратегий ИИ (для продвинутых пользователей).
- Экспорт данных игры для анализа.

Нефункциональные требования:

- производительность: не менее 60 FPS[17] на среднем ПК;
- кроссплатформенность: Windows, Linux;
- интуитивно понятный интерфейс;
- время реакции ИИ: < 100 мс на ход;
- модульность кода для лёгкой доработки.

3. Архитектурное решение программного модуля ИИ

Стек технологий для разработки:

1. язык программирования: Python 3.x;
2. игровой движок: Pygame;
3. ИИ-фреймворк: NumPy, SciPy;
4. визуализация: Matplotlib (для графиков анализа);
5. Графический элемент GUI вместо текстовых команд[18]: Tkinter (для настроек и редактора).

Модульная структура разработки:

- Игровой движок. Управление игровым полем (10×20); генерация и перемещение фигур; проверка столкновений; очистка линий и подсчёт очков.
- Модуль ИИ. Оценка позиций (функция стоимости); поиск оптимального хода; уровни сложности (настройка весов); режим подсказок.
- Интерфейс пользователя. Главное меню; игровое поле; панель статистики; настройки сложности ИИ; редактор стратегий.
- Система анализа. Логирование игровых событий; визуализация метрик ИИ; экспорт данных в CSV.

4. Алгоритм Модуля Искусственного интеллекта

Функция оценки позиции:

$$\text{Стоимость} = w1 \cdot \text{дыры} + w2 \cdot \text{высота} + w3 \cdot \text{неровность} + w4 \cdot \text{линии}$$

Метрики:

- Дыры: количество пустых ячеек под заполненными блоками.
- Высота: средняя высота заполненных клеток.
- Неровность: сумма абсолютных разностей высот соседних столбцов.
- Линии: бонус за возможность очистить строки (отрицательный вклад в стоимость).

[17] **FPS** (Frames Per Second, или «кадров в секунду») – это применительно к работе ИИ и обычно означает скорость, т.е., сколько изображений модель способна сгенерировать за 1 секунду. Для комфортной игры в «Тетрис» с ИИ достаточно стабильных 30–60 FPS.

[18] *Графический элемент GUI* - это всё, что вы видите на экране и с чем можете взаимодействовать: кнопки, меню, окна, иконки, ползунки и т.д. Он заменяет сложные текстовые команды на интуитивно понятные визуальные объекты.

Уровни сложности:

Уровень	Веса (дыры/высота/неровность/линии)	Особенности
Новичок	0.5 / 0.3 / 0.2 / 0.1	Простые правила, случайные ошибки
Средний	0.8 / 0.6 / 0.4 / 0.3	Учитывает все метрики, моделирует 1 ход вперёд
Эксперт	1.0 / 0.9 / 0.7 / 0.5	Моделирует 2 - 3 хода вперёд, максимальная точность

Алгоритм поиска хода:

1. Сгенерировать все возможные позиции текущей фигуры (все повороты + смещения).
2. Для каждой позиции рассчитать стоимость по формуле.
3. Выбрать позицию с минимальной стоимостью.
4. Выполнить ход или показать подсказку.

5. План разработки

Этап 1. Прототип (2 недели): базовая игра «Тетрис»; простое поле и фигуры; управление клавиатурой; подсчёт очков.

Этап 2. ИИ-модуль (3 недели): функция оценки позиций; поиск оптимального хода; три уровня сложности; режим подсказок.

Этап 3. Интерфейс (2 недели): главное меню; панель статистики; настройки сложности; визуализация «мышления» ИИ.

Этап 4. Система анализа (1 неделя): логирование событий; экспорт данных; графики метрик.

Этап 5. Тестирование и оптимизация (2 недели): юнит-тесты; интеграционные тесты; оптимизация производительности; пользовательское тестирование.

Этап 6. Релиз (1 неделя): сборка установочного пакета; документация; публикация на платформах.

6. Тестирование разработки

Виды тестов:

- Юнит-тесты: проверка функций оценки позиций, поиска хода, очистки линий.
- Интеграционные тесты: взаимодействие ИИ с игровым движком.
- Регрессионные тесты: проверка стабильности после изменений.
- Производительность: FPS при работе ИИ.
- Пользовательские тесты: сбор обратной связи.

Тестовые сценарии:

- игра против ИИ на всех уровнях сложности;
- режим обучения с подсказками;
- стресс-тест (длительная игра);
- проверка крайних случаев (почти заполненное поле).

7. Документация

Состав документации:

1. Руководство пользователя (правила, управление, режимы).
2. Техническая документация (архитектура, API[19]).
3. Руководство разработчика (инструкция по доработке ИИ).
4. Файл README с инструкциями по установке и запуску.

8. План релиза

Версии:

- Версия 1.0: базовый функционал (игра + ИИ).
- Версия 1.1: система анализа и экспорта данных.
- Версия 1.2: редактор стратегий ИИ.
- Версия 2.0: нейросетевой ИИ (обучение с подкреплением).

Платформы:

- ПК (Windows, Linux);
- веб-версия (опционально);
- мобильные версии (опционально).

9. Поддержка и развитие

План поддержки:

- исправление критических багов в течение 2 недель после релиза;
- регулярные обновления (раз в 1–2 месяца);
- сбор и анализ отзывов пользователей.

Дорожная карта развития проекта:

- мультиплеер (игрок против ИИ онлайн);
- система достижений;
- турниры и рейтинги;
- интеграция с образовательными платформами.

[19] *API (Application Programming Interface, или программный интерфейс приложения)* - это набор правил, инструкций и инструментов, которые позволяют разным программам взаимодействовать друг с другом: обмениваться данными и выполнять действия. *Разработка модели игры «Тетрис» с ИИ* может использовать API для: сохранения рекордов на удалённом сервере; загрузки новых скинов фигур из онлайн-галереи; подключения к платформе для онлайн-турниров.

10. Бюджет и ресурсы

Ресурсы: разработчики: 1 человек; дизайнер: 1 человек (графика); тестировщик: 1 человек;
сроки: 10–12 недель.
Затраты:

- оборудование: ПК для разработки;
- ПО: бесплатное (открытый исходный код);
- хостинг для релиза: опционально.

11. Риски и решения

Риск	Решение
Низкая производительность ИИ-агента	Оптимизация алгоритма, ограничение глубины поиска
ИИ-агент играет слишком хорошо	Настроить весовые коэффициенты, добавить случайные ошибки
Сложность отладки ИИ-агента	Добавить логирование и визуализацию «мышления»
Неинтересный геймплей против ИИ-агента	Реализовать несколько стилей игры (агрессивный, осторожный)
Проблемы совместимости	Тестирование на разных платформах и конфигурациях

12. Ожидаемые результаты
Количественные:

- релиз готовой игры с ИИ-агентом;
- поддержка 3 уровней сложности;
- система анализа игровых данных;
- документация и руководство разработчика.

Качественные:

- высокая реиграбельность (за счёт ИИ-агента и режимов);
- образовательный потенциал (изучение алгоритмов ИИ);
- модульность для будущих улучшений;
- открытый код для сообщества.

Приложение 2.

Реализация программного продукта «*Тетрис & ИИ-Агент*» на языке программирования Python

```
Python
import tkinter as tk
from tkinter import messagebox
import random
import json
import os
import time
import math

class Tetris:
    def __init__(self, master):
        self.master = master
        self.master.title("Tetris")
        self.master.resizable(False, False)

        self.width = 10
        self.height = 20
        self.cell_size = 30

        self.difficulty_settings = {
            "Легкий": 600,
            "Средний": 400,
            "Сложный": 200
        }
        self.current_difficulty = "Средний"
        self.fall_speed = self.difficulty_settings[self.current_difficulty]

        self.high_score = self.load_high_score()

        self.frame_count = 0
        self.last_fps_update = time.time()
        self.current_fps = 0

        self.game_mode = "single"

        self.max_score = 5000

        # Настройки сложности ИИ (человеческое поведение)
        self.ai_difficulty = "Средний"
        self.ai_settings = {
            "Легкий": {
                "reaction_delay": 500,
                "reaction_variance": 300,
                "move_delay": 120,
                "move_variance": 50,
                "hesitation_chance": 0.4,
                "hesitation_delay": 600,
                "precision": 0.6,
                "quick_drop_chance": 0.1,
                "panic_threshold": 12,
                "soft_drop_speed": 3,
            },
            "Средний": {
```

```
            "reaction_delay": 300,
            "reaction_variance": 200,
            "move_delay": 80,
            "move_variance": 40,
            "hesitation_chance": 0.25,
            "hesitation_delay": 400,
            "precision": 0.8,
            "quick_drop_chance": 0.3,
            "panic_threshold": 15,
            "soft_drop_speed": 4,
        },
        "Сложный": {
            "reaction_delay": 200,
            "reaction_variance": 150,
            "move_delay": 50,
            "move_variance": 30,
            "hesitation_chance": 0.1,
            "hesitation_delay": 250,
            "precision": 0.95,
            "quick_drop_chance": 0.6,
            "panic_threshold": 17,
            "soft_drop_speed": 5,
        }
    }

    self.ai_reaction_delay = 300
    self.ai_reaction_variance = 200
    self.ai_move_delay = 80
    self.ai_move_variance = 40
    self.ai_hesitation_chance = 0.25
    self.ai_hesitation_delay = 400
    self.ai_precision = 0.8
    self.ai_quick_drop_chance = 0.3
    self.ai_soft_drop_speed = 4
    self.ai_panic_threshold = 15

    self.ai_board = None
    self.ai_current_shape = None
    self.ai_next_shape = None
    self.ai_current_x = 0
    self.ai_current_y = 0
    self.ai_game_over = False
    self.ai_loop_id = None
    self.ai_fall_loop_id = None
    self.ai_speed = self.fall_speed
    self.ai_is_moving = False
    self.ai_target_x = 0
    self.ai_target_rotation = 0
    self.ai_is_falling = False
    self.ai_score = 0
    self.ai_soft_dropping = False

    self.confetti_particles = []
    self.confetti_animation_id = None

    self.button_animation_id = None
    self.robot_animation_id = None
    self.particle_animation_id = None
    self.game_loop_id = None
```

```
        self.returning_to_menu = False
        self.victory_shown = False
        self.game_ended = False

        self.all_after_ids = []

        self.create_main_menu()

    def load_high_score(self):
        try:
            if os.path.exists("tetris_scores.json"):
                with open("tetris_scores.json", "r") as f:
                    data = json.load(f)
                    return data.get("high_score", 0)
        except:
            pass
        return 0

    def save_high_score(self):
        try:
            with open("tetris_scores.json", "w") as f:
                json.dump({"high_score": self.high_score}, f)
        except:
            pass

    def safe_after(self, delay, callback):
        after_id = self.master.after(delay, callback)
        self.all_after_ids.append(after_id)
        return after_id

    def cancel_all_after(self):
        for after_id in self.all_after_ids:
            try:
                self.master.after_cancel(after_id)
            except:
                pass
        self.all_after_ids = []

    def create_main_menu(self):
        self.cancel_all_after()

        for widget in self.master.winfo_children():
            widget.destroy()

        self.force_stop_all_animations()

        self.returning_to_menu = False
        self.victory_shown = False
        self.game_ended = False

        self.menu_frame = tk.Frame(self.master, bg="gray", width=400,
height=500)
        self.menu_frame.pack(fill=tk.BOTH, expand=True)
        self.menu_frame.pack_propagate(False)

        # razrabotchik
        tk.Label(
            self.menu_frame,
```

```
            justify="left",
            fg="lightgray",
            bg="gray",
            font=(None, 7),
            text="Разработчики: Паньшин Георгий, Липатова Анжелика, Шемякина
Дарья\nГБПОУ МО \"Дмитровский техникум\""
        ).pack(pady=0, padx=0, expand=True)

        arc_frame = tk.Frame(
            self.menu_frame,
            bg="black",
            width=350,
            height=150,
            highlightbackground="darkgray",
            highlightthickness=2,
            bd=0
        )
        arc_frame.pack(pady=40)
        arc_frame.pack_propagate(False)

        self.text_frame = tk.Frame(arc_frame, bg="#2a2a2a", width=330,
height=130)
        self.text_frame.pack(pady=10, padx=10)
        self.text_frame.pack_propagate(False)

        tetris_frame = tk.Frame(self.text_frame, bg="#2a2a2a")
        tetris_frame.pack(pady=(25, 5))

        colors = ['red', 'deep sky blue', 'light green', 'yellow', 'pink',
'orange']
        letters = ['T', 'E', 'T', 'R', 'I', 'S']

        for i, letter in enumerate(letters):
            color = colors[i]
            label = tk.Label(
                tetris_frame,
                text=letter,
                font=("Arial", 36, "bold"),
                bg="#2a2a2a",
                fg=color
            )
            label.pack(side=tk.LEFT, padx=2)

        edition_label = tk.Label(
            self.text_frame,
            text="Classic Edition",
            font=("Arial", 14, "italic"),
            bg="#2a2a2a",
            fg="lightgray"
        )
        edition_label.pack()

        self.robot = tk.Label(
            self.text_frame,
            text="👹",
            font=("Arial", 24),
            bg="#2a2a2a",
            fg="lightgray"
        )
```

```
        self.robot.place(x=10, y=85)

        self.robot_jump_offset = 0
        self.robot_jumping_up = True
        self.animate_robot()

        self.buttons_frame = tk.Frame(self.menu_frame, bg="gray")
        self.buttons_frame.pack(pady=10)

        self.single_mode_btn = tk.Button(
            self.buttons_frame,
            text="Одиночный режим",
            font=("Arial", 18, "bold"),
            bg="deep sky blue",
            width=18,
            command=self.start_single_game
        )
        self.single_mode_btn.pack(pady=5)

        self.ai_menu_btn = tk.Button(
            self.buttons_frame,
            text="ИИ",
            font=("Arial", 18, "bold"),
            bg="lightgreen",
            width=18,
            command=self.open_ai_menu
        )
        self.ai_menu_btn.pack(pady=5)

        self.settings_btn = tk.Button(
            self.buttons_frame,
            text="Параметры",
            font=("Arial", 16),
            bg="lightgray",
            width=15,
            command=self.open_settings
        )
        self.settings_btn.pack(pady=5)

        high_score_frame = tk.Frame(self.menu_frame, bg="gray")
        high_score_frame.pack(side=tk.BOTTOM, pady=10)

        tk.Label(
            high_score_frame,
            text=f"Максимальный рекорд: {self.high_score}",
            font=("Arial", 12, "bold"),
            bg="gray",
            fg="gold"
        ).pack()

        self.animate_button()
        self.settings_open = False

    def open_ai_menu(self):
        self.cancel_all_after()

        for widget in self.master.winfo_children():
            widget.destroy()
```

```
        self.force_stop_all_animations()

        self.ai_menu_frame = tk.Frame(self.master, bg="gray", width=400,
height=500)
        self.ai_menu_frame.pack(fill=tk.BOTH, expand=True)
        self.ai_menu_frame.pack_propagate(False)

        title_frame = tk.Frame(
            self.ai_menu_frame,
            bg="black",
            width=350,
            height=100,
            highlightbackground="darkgray",
            highlightthickness=2,
            bd=0
        )
        title_frame.pack(pady=20)
        title_frame.pack_propagate(False)

        inner_title = tk.Frame(title_frame, bg="#2a2a2a", width=330, height=80)
        inner_title.pack(pady=10, padx=10)
        inner_title.pack_propagate(False)

        tk.Label(
            inner_title,
            text="👿 ИИ 👿",
            font=("Arial", 36, "bold"),
            bg="#2a2a2a",
            fg="lightgreen"
        ).pack(expand=True)

        # Основной контент сразу после арки
        content_frame = tk.Frame(self.ai_menu_frame, bg="gray")
        content_frame.pack(pady=10)

        # Кнопка "Против ИИ"
        self.vs_ai_btn = tk.Button(
            content_frame,
            text="Против ИИ",
            font=("Arial", 18, "bold"),
            bg="lightgreen",
            width=18,
            command=self.open_vs_ai_settings
        )
        self.vs_ai_btn.pack(pady=5)

        # Выбор сложности ИИ
        difficulty_frame = tk.Frame(content_frame, bg="gray")
        difficulty_frame.pack(pady=10)

        tk.Label(
            difficulty_frame,
            text="Сложность ИИ:",
            font=("Arial", 14, "bold"),
            bg="gray",
            fg="white"
        ).pack(pady=5)

        self.ai_difficulty_var = tk.StringVar(value=self.ai_difficulty)
```

```
        for text, value in [
            ("Новичок (медленный, неточный)", "Легкий"),
            ("Любитель (средняя скорость)", "Средний"),
            ("Опытный (быстрый, точный)", "Сложный")
        ]:
            tk.Radiobutton(
                difficulty_frame,
                text=text,
                variable=self.ai_difficulty_var,
                value=value,
                font=("Arial", 12),
                bg="gray",
                fg="white",
                selectcolor="gray",
                command=self.apply_ai_settings
            ).pack(anchor=tk.W, padx=30, pady=3)

        self.ai_difficulty_label = tk.Label(
            difficulty_frame,
            text=f"Текущая: {self.ai_difficulty}",
            font=("Arial", 11),
            bg="gray",
            fg="lightgreen"
        )
        self.ai_difficulty_label.pack(pady=5)

        # Кнопка "Назад" сразу после надписи о текущей сложности
        tk.Button(
            difficulty_frame,
            text="Назад",
            font=("Arial", 12, "bold"),
            bg="lightcoral",
            width=10,
            command=self.back_to_main_menu
        ).pack(pady=10)

    def apply_ai_settings(self):
        """Применение настроек ИИ в соответствии с выбранной сложностью"""
        self.ai_difficulty = self.ai_difficulty_var.get()
        settings = self.ai_settings[self.ai_difficulty]

        self.ai_reaction_delay = settings["reaction_delay"]
        self.ai_reaction_variance = settings["reaction_variance"]
        self.ai_move_delay = settings["move_delay"]
        self.ai_move_variance = settings["move_variance"]
        self.ai_hesitation_chance = settings["hesitation_chance"]
        self.ai_hesitation_delay = settings["hesitation_delay"]
        self.ai_precision = settings["precision"]
        self.ai_quick_drop_chance = settings["quick_drop_chance"]
        self.ai_soft_drop_speed = settings["soft_drop_speed"]
        self.ai_panic_threshold = settings["panic_threshold"]

        if hasattr(self, 'ai_difficulty_label') and
self.ai_difficulty_label.winfo_exists():
            self.ai_difficulty_label.config(text=f"Текущая:
{self.ai_difficulty}")

    def open_vs_ai_settings(self):
```

```
        self.apply_ai_settings()

        self.cancel_all_after()

        for widget in self.master.winfo_children():
            widget.destroy()

        self.vs_ai_settings_frame = tk.Frame(self.master, bg="gray", width=400,
height=500)
        self.vs_ai_settings_frame.pack(fill=tk.BOTH, expand=True)
        self.vs_ai_settings_frame.pack_propagate(False)

        title_frame = tk.Frame(
            self.vs_ai_settings_frame,
            bg="black",
            width=350,
            height=80,
            highlightbackground="darkgray",
            highlightthickness=2,
            bd=0
        )
        title_frame.pack(pady=20)
        title_frame.pack_propagate(False)

        inner_title = tk.Frame(title_frame, bg="#2a2a2a", width=330, height=60)
        inner_title.pack(pady=10, padx=10)
        inner_title.pack_propagate(False)

        tk.Label(
            inner_title,
            text=f"ПРОТИВ ИИ ({self.ai_difficulty})",
            font=("Arial", 20, "bold"),
            bg="#2a2a2a",
            fg="lightgreen"
        ).pack(expand=True)

        # Основной контент сразу после арки
        content_frame = tk.Frame(self.vs_ai_settings_frame, bg="gray")
        content_frame.pack(pady=5)

        settings_frame = tk.Frame(content_frame, bg="gray")
        settings_frame.pack(pady=5)

        tk.Label(
            settings_frame,
            text="Максимальное количество очков:",
            font=("Arial", 14, "bold"),
            bg="gray",
            fg="white"
        ).pack(pady=5)

        self.max_score_var = tk.IntVar(value=self.max_score)

        score_options = [1000, 2500, 5000, 10000]

        for score in score_options:
            tk.Radiobutton(
                settings_frame,
                text=f"{score} очков",
```

```
            variable=self.max_score_var,
            value=score,
            font=("Arial", 14),
            bg="gray",
            fg="white",
            selectcolor="gray"
        ).pack(anchor=tk.W, padx=80, pady=3)

    custom_frame = tk.Frame(settings_frame, bg="gray")
    custom_frame.pack(pady=5)

    tk.Radiobutton(
        custom_frame,
        text="Своё значение:",
        variable=self.max_score_var,
        value=-1,
        font=("Arial", 14),
        bg="gray",
        fg="white",
        selectcolor="gray"
    ).pack(side=tk.LEFT, padx=(80, 5))

    self.custom_score_entry = tk.Entry(
        custom_frame,
        font=("Arial", 14),
        width=8,
        state="disabled"
    )
    self.custom_score_entry.pack(side=tk.LEFT, padx=5)

    self.max_score_var.trace_add("write", self.on_max_score_change)

    # Кнопки СТАРТ и Назад
    buttons_frame = tk.Frame(content_frame, bg="gray")
    buttons_frame.pack(pady=5)

    tk.Button(
        buttons_frame,
        text="СТАРТ",
        font=("Arial", 20, "bold"),
        bg="lightgreen",
        width=12,
        command=self.start_vs_ai_with_settings
    ).pack(pady=3)

    tk.Button(
        buttons_frame,
        text="Назад",
        font=("Arial", 14, "bold"),
        bg="lightcoral",
        width=12,
        command=self.open_ai_menu
    ).pack(pady=3)

def on_max_score_change(self, *args):
    if self.max_score_var.get() == -1:
        self.custom_score_entry.config(state="normal")
    else:
        self.custom_score_entry.config(state="disabled")
```

```
            self.custom_score_entry.delete(0, tk.END)

    def start_vs_ai_with_settings(self):
        if self.max_score_var.get() == -1:
            try:
                custom_score = int(self.custom_score_entry.get())
                if custom_score <= 0:
                    messagebox.showerror("Ошибка", "Введите положительное число
очков!")
                    return
                self.max_score = custom_score
            except ValueError:
                messagebox.showerror("Ошибка", "Введите корректное число
очков!")
                return
        else:
            self.max_score = self.max_score_var.get()

        self.apply_ai_settings()
        self.start_vs_ai_game()

    def force_stop_all_animations(self):
        self.cancel_all_after()

        self.button_animation_id = None
        self.robot_animation_id = None
        self.particle_animation_id = None
        self.game_loop_id = None
        self.ai_loop_id = None
        self.ai_fall_loop_id = None
        self.confetti_animation_id = None

        self.confetti_particles = []
        self.game_over = False
        self.paused = False
        self.ai_game_over = False
        self.ai_is_moving = False
        self.ai_is_falling = False
        self.ai_soft_dropping = False
        self.victory_shown = False
        self.game_ended = False

    def stop_all_animations(self):
        self.force_stop_all_animations()

    def start_single_game(self):
        self.start_game("single")

    def start_vs_ai_game(self):
        self.start_game("vs_ai")

    def open_settings(self):
        if self.settings_open:
            return

        self.settings_open = True

        for widget in self.menu_frame.winfo_children():
            widget.destroy()
```

```
        self.settings_frame = tk.Frame(self.menu_frame, bg="gray", width=400,
height=500)
        self.settings_frame.pack(fill=tk.BOTH, expand=True)
        self.settings_frame.pack_propagate(False)

        title_label = tk.Label(
            self.settings_frame,
            text="НАСТРОЙКИ ИГРЫ",
            font=("Arial", 24, "bold"),
            bg="gray",
            fg="yellow"
        )
        title_label.pack(pady=30)

        difficulty_frame = tk.Frame(self.settings_frame, bg="gray")
        difficulty_frame.pack(pady=20)

        tk.Label(
            difficulty_frame,
            text="Сложность:",
            font=("Arial", 18, "bold"),
            bg="gray",
            fg="white"
        ).pack(pady=10)

        self.settings_difficulty_var =
tk.StringVar(value=self.current_difficulty)

        for text, value in [("Легкий (медленное падение)", "Легкий"),
                            ("Средний (обычное падение)", "Средний"),
                            ("Сложный (быстрое падение)", "Сложный")]:
            tk.Radiobutton(
                difficulty_frame,
                text=text,
                variable=self.settings_difficulty_var,
                value=value,
                font=("Arial", 14),
                bg="gray",
                fg="white",
                selectcolor="gray"
            ).pack(anchor=tk.W, padx=50, pady=8)

        self.current_diff_label = tk.Label(
            self.settings_frame,
            text=f"Текущая сложность: {self.current_difficulty}",
            font=("Arial", 12),
            bg="gray",
            fg="lightgreen"
        )
        self.current_diff_label.pack(pady=15)

        buttons_frame = tk.Frame(self.settings_frame, bg="gray")
        buttons_frame.pack(pady=30)

        tk.Button(
            buttons_frame,
            text="Сохранить",
            font=("Arial", 16, "bold"),
```

```
            bg="lightgreen",
            width=12,
            command=self.save_settings_from_menu
        ).pack(side=tk.LEFT, padx=15)

        tk.Button(
            buttons_frame,
            text="Выйти",
            font=("Arial", 16, "bold"),
            bg="lightcoral",
            width=12,
            command=self.back_to_main_menu
        ).pack(side=tk.LEFT, padx=15)

    def show_save_notification(self):
        notification_arc = tk.Frame(
            self.settings_frame,
            bg="black",
            width=200,
            height=80,
            highlightbackground="darkgray",
            highlightthickness=2,
            bd=0
        )
        notification_arc.place(relx=0.5, rely=0.5, anchor="center")
        notification_arc.pack_propagate(False)

        inner_frame = tk.Frame(notification_arc, bg="#2a2a2a", width=180,
height=60)
        inner_frame.pack(pady=10, padx=10)
        inner_frame.pack_propagate(False)

        tk.Label(
            inner_frame,
            text="Сохранено!",
            font=("Arial", 16, "bold"),
            bg="#2a2a2a",
            fg="lightgreen"
        ).pack(expand=True)

        self.safe_after(2000, notification_arc.destroy)

    def save_settings_from_menu(self):
        self.current_difficulty = self.settings_difficulty_var.get()
        self.fall_speed = self.difficulty_settings[self.current_difficulty]
        self.ai_speed = self.fall_speed
        self.current_diff_label.config(text=f"Текущая сложность:
{self.current_difficulty}")
        self.show_save_notification()

    def back_to_main_menu(self):
        self.settings_open = False
        self.create_main_menu()

    def animate_robot(self):
        if hasattr(self, 'text_frame') and self.text_frame.winfo_exists() and
not self.returning_to_menu:
            jump_height = 20
            step = 3
```

```
            if self.robot_jumping_up:
                self.robot_jump_offset += step
                if self.robot_jump_offset >= jump_height:
                    self.robot_jumping_up = False
            else:
                self.robot_jump_offset -= step
                if self.robot_jump_offset <= 0:
                    self.robot_jump_offset = 0
                    self.robot_jumping_up = True

            base_y = 85
            self.robot.place(x=10, y=base_y - self.robot_jump_offset)

            if self.robot_jump_offset > 0:
                colors = ['deep sky blue', 'light green', 'yellow', 'pink',
'orange']
                color_index = (self.robot_jump_offset // 3) % len(colors)
                self.robot.config(fg=colors[color_index])
            else:
                self.robot.config(fg="lightgray")

            self.robot_animation_id = self.safe_after(50, self.animate_robot)

    def animate_button(self):
        if hasattr(self, 'menu_frame') and self.menu_frame.winfo_exists() and
not self.returning_to_menu:
            if hasattr(self, 'single_mode_btn') and
self.single_mode_btn.winfo_exists():
                colors = ['deep sky blue', 'light green', 'yellow', 'pink',
'orange']
                self.current_color_index = getattr(self, 'current_color_index',
0)
                self.single_mode_btn.config(bg=colors[self.current_color_index])
                self.current_color_index = (self.current_color_index + 1) %
len(colors)
                self.button_animation_id = self.safe_after(300,
self.animate_button)

    def init_confetti(self, canvas):
        self.confetti_particles = []
        width = self.width * self.cell_size
        height = self.height * self.cell_size

        colors = ['#FFD700', '#FFA500', '#FF69B4', '#00FF00', '#00BFFF',
'#FF4500', '#9932CC']

        for _ in range(50):
            particle = {
                'x': random.randint(0, width),
                'y': random.randint(-height, 0),
                'speed_x': random.uniform(-2, 2),
                'speed_y': random.uniform(1, 3),
                'size': random.randint(3, 8),
                'color': random.choice(colors),
                'rotation': random.randint(0, 360),
                'rotation_speed': random.uniform(-5, 5)
            }
            self.confetti_particles.append(particle)
```

```
    def animate_confetti(self, canvas):
        if not self.confetti_particles or self.returning_to_menu:
            return

        canvas.delete("confetti")
        width = self.width * self.cell_size
        height = self.height * self.cell_size

        for particle in self.confetti_particles:
            particle['x'] += particle['speed_x']
            particle['y'] += particle['speed_y']
            particle['rotation'] += particle['rotation_speed']
            particle['speed_y'] += 0.1

            if particle['x'] < 0 or particle['x'] > width:
                particle['speed_x'] *= -0.8

            if particle['y'] > height:
                particle['y'] = random.randint(-50, 0)
                particle['x'] = random.randint(0, width)
                particle['speed_y'] = random.uniform(1, 3)

            x = particle['x']
            y = particle['y']
            size = particle['size']

            angle = math.radians(particle['rotation'])
            points = []
            for dx, dy in [(-size/2, -size/4), (size/2, -size/4), (size/2,
size/4), (-size/2, size/4)]:
                rot_x = dx * math.cos(angle) - dy * math.sin(angle)
                rot_y = dx * math.sin(angle) + dy * math.cos(angle)
                points.append(x + rot_x)
                points.append(y + rot_y)

            canvas.create_polygon(points, fill=particle['color'], outline="",
tags="confetti")

        if not self.returning_to_menu:
            self.confetti_animation_id = self.safe_after(50, lambda:
self.animate_confetti(canvas))

    def init_ai_game(self):
        self.ai_board = [[0 for _ in range(self.width)] for _ in
range(self.height)]
        self.ai_game_over = False
        self.ai_is_moving = False
        self.ai_is_falling = True
        self.ai_soft_dropping = False
        self.ai_score = 0
        self.ai_current_shape = self.get_random_shape()
        self.ai_next_shape = self.get_random_shape()
        self.ai_current_x = self.width // 2 - 2
        self.ai_current_y = 0
        self.ai_target_x = self.ai_current_x
        self.ai_target_rotation = 0

        self.ai_draw()
```

```
        self.update_ai_score()

        self.ai_start_fall_loop()
        self.schedule_ai_decision()

    def update_ai_score(self):
        if hasattr(self, 'ai_score_label') and
self.ai_score_label.winfo_exists():
            self.ai_score_label.config(text=f"Счёт ИИ: {self.ai_score}")

    def schedule_ai_decision(self):
        """Планирует следующее решение ИИ с учетом задержки реакции"""
        if self.ai_game_over or self.paused or self.returning_to_menu or
self.game_ended:
            return

        delay = self.ai_reaction_delay + random.randint(-
self.ai_reaction_variance, self.ai_reaction_variance)
        delay = max(50, delay)

        if random.random() < self.ai_hesitation_chance:
            delay += random.randint(0, self.ai_hesitation_delay)

        self.safe_after(delay, self.ai_make_decision)

    def ai_start_fall_loop(self):
        if self.ai_game_over or self.paused or self.returning_to_menu or
self.game_ended:
            return
        if not self.ai_is_falling:
            return

        if self.ai_soft_dropping:
            fall_delay = self.ai_speed // self.ai_soft_drop_speed
        else:
            fall_delay = self.ai_speed

        if not self.ai_soft_dropping and random.random() <
self.ai_quick_drop_chance:
            self.ai_soft_dropping = True
            fall_delay = self.ai_speed // self.ai_soft_drop_speed

        if not self.ai_check_collision(self.ai_current_shape, self.ai_current_x,
self.ai_current_y + 1):
            self.ai_current_y += 1
            self.ai_draw()
        else:
            self.ai_is_falling = False
            self.ai_soft_dropping = False
            if self.ai_merge_shape():
                return
            self.ai_draw()

            if self.game_mode == "vs_ai" and not self.returning_to_menu and not
self.game_ended:
                self.check_score_limit()
                victory = self.check_victory()
                if victory and not self.victory_shown:
                    self.show_victory_message(victory)
```

```
                    return

                self.ai_is_falling = True
                self.ai_start_fall_loop()
                self.schedule_ai_decision()
                return

        self.ai_fall_loop_id = self.safe_after(fall_delay,
self.ai_start_fall_loop)

    def ai_make_decision(self):
        """ИИ принимает решение о следующем действии"""
        if self.ai_game_over or self.paused or self.returning_to_menu or
self.game_ended:
            return

        if not self.ai_is_moving and self.ai_current_shape and
self.ai_is_falling:
            self.ai_calculate_best_move()

    def ai_calculate_best_move(self):
        if self.ai_game_over or self.paused or self.returning_to_menu or
self.game_ended:
            return

        best_x, best_rotation, best_score = self.ai_find_best_position_only()

        if random.random() > self.ai_precision:
            offset = random.choice([-2, -1, 0, 1, 2])
            best_x = max(0, min(self.width - 1, best_x + offset))

            if random.random() < 0.3:
                best_rotation = (best_rotation + random.randint(1, 3)) % 4

        if self.is_ai_panicking():
            best_x = max(0, min(self.width - 1, best_x + random.randint(-3, 3)))
            if random.random() < 0.4:
                best_rotation = random.randint(0, 3)

        self.ai_target_x = best_x
        self.ai_target_rotation = best_rotation
        self.ai_is_moving = True
        self.ai_animate_to_target()

    def ai_find_best_position_only(self):
        if not self.ai_current_shape:
            return self.ai_current_x, 0, 0

        best_x = self.ai_current_x
        best_rotation = 0
        best_score = -999999

        original_blocks = [block[:] for block in
self.ai_current_shape['blocks']]

        for rot in range(4):
            rotated_blocks = original_blocks[:]
            for _ in range(rot):
                new_blocks = []
```

```
                for bx, by in rotated_blocks:
                    new_blocks.append([by, -bx])
                if new_blocks:
                    min_x = min(b[0] for b in new_blocks)
                    min_y = min(b[1] for b in new_blocks)
                    rotated_blocks = [[b[0] - min_x, b[1] - min_y] for b in
new_blocks]

            rotated_shape = {
                'name': self.ai_current_shape['name'],
                'blocks': rotated_blocks,
                'color': self.ai_current_shape['color']
            }

            for x in range(-2, self.width + 2):
                if self.ai_check_collision_shape(rotated_shape, x, 0):
                    continue

                y = 0
                while not self.ai_check_collision_shape(rotated_shape, x, y +
1):
                    y += 1

                score = self.ai_evaluate_position(rotated_shape, x, y)

                if score > best_score:
                    best_score = score
                    best_x = x
                    best_rotation = rot

        return best_x, best_rotation, best_score

    def ai_animate_to_target(self):
        """Анимация движения ИИ к целевой позиции (по одному шагу)"""
        if self.ai_game_over or self.paused or self.returning_to_menu or
self.game_ended:
            return

        move_delay = self.ai_move_delay + random.randint(-self.ai_move_variance,
self.ai_move_variance)
        move_delay = max(20, move_delay)

        rotation_done = (self.ai_target_rotation == 0)
        x_done = (self.ai_current_x == self.ai_target_x)

        if not rotation_done:
            self.ai_rotate_once()
            self.ai_target_rotation -= 1
            if self.ai_target_rotation < 0:
                self.ai_target_rotation += 4
            self.ai_draw()
            self.ai_loop_id = self.safe_after(move_delay * 2,
self.ai_animate_to_target)
            return

        if not x_done:
            if self.ai_current_x < self.ai_target_x:
                if not self.ai_check_collision(self.ai_current_shape,
self.ai_current_x + 1, self.ai_current_y):
```

```
                    self.ai_current_x += 1
            elif self.ai_current_x > self.ai_target_x:
                if not self.ai_check_collision(self.ai_current_shape,
self.ai_current_x - 1, self.ai_current_y):
                    self.ai_current_x -= 1
            self.ai_draw()
            self.ai_loop_id = self.safe_after(move_delay,
self.ai_animate_to_target)
            return

        self.ai_is_moving = False
        self.schedule_ai_decision()

    def ai_rotate_once(self):
        rotated_blocks = []
        for bx, by in self.ai_current_shape['blocks']:
            rotated_blocks.append([by, -bx])
        if rotated_blocks:
            min_x = min(b[0] for b in rotated_blocks)
            min_y = min(b[1] for b in rotated_blocks)
            rotated_blocks = [[b[0] - min_x, b[1] - min_y] for b in
rotated_blocks]

            rotated_shape = {
                'name': self.ai_current_shape['name'],
                'blocks': rotated_blocks,
                'color': self.ai_current_shape['color']
            }

            if not self.ai_check_collision(rotated_shape, self.ai_current_x,
self.ai_current_y):
                self.ai_current_shape = rotated_shape

    def is_ai_panicking(self):
        """Проверяет, находится ли ИИ в состоянии паники (поле почти
заполнено)"""
        for y in range(self.ai_panic_threshold):
            for x in range(self.width):
                if self.ai_board[y][x] != 0:
                    return True
        return False

    def ai_check_collision_shape(self, shape, offset_x, offset_y):
        for x, y in shape['blocks']:
            new_x = offset_x + x
            new_y = offset_y + y
            if new_x < 0 or new_x >= self.width or new_y >= self.height:
                return True
            if new_y >= 0 and self.ai_board[new_y][new_x] != 0:
                return True
        return False

    def ai_evaluate_position(self, shape, x, y):
        temp_board = [row[:] for row in self.ai_board]

        for bx, by in shape['blocks']:
            board_x = x + bx
            board_y = y + by
            if 0 <= board_y < self.height and 0 <= board_x < self.width:
```

```
            temp_board[board_y][board_x] = 1

    heights = [0] * self.width
    holes = 0
    bumpiness = 0

    for col in range(self.width):
        for row in range(self.height):
            if temp_board[row][col] != 0:
                heights[col] = self.height - row
                break

    for col in range(self.width - 1):
        bumpiness += abs(heights[col] - heights[col + 1])

    for col in range(self.width):
        found_block = False
        for row in range(self.height):
            if temp_board[row][col] != 0:
                found_block = True
            elif found_block and temp_board[row][col] == 0:
                holes += 1

    max_height = max(heights) if heights else 0
    height_sum = sum(heights)

    lines = 0
    for row in range(self.height):
        if all(temp_board[row][col] != 0 for col in range(self.width)):
            lines += 1

    score = (
        lines * 800 +
        height_sum * -40 +
        max_height * -100 +
        holes * -300 +
        bumpiness * -20
    )

    return score

def ai_check_collision(self, shape, offset_x, offset_y):
    for x, y in shape['blocks']:
        new_x = offset_x + x
        new_y = offset_y + y
        if new_x < 0 or new_x >= self.width or new_y >= self.height:
            return True
        if new_y >= 0 and self.ai_board[new_y][new_x] != 0:
            return True
    return False

def ai_merge_shape(self):
    if self.returning_to_menu:
        return False

    for x, y in self.ai_current_shape['blocks']:
        board_x = self.ai_current_x + x
        board_y = self.ai_current_y + y
        if 0 <= board_y < self.height and 0 <= board_x < self.width:
```

```
                self.ai_board[board_y][board_x] = self.ai_current_shape['color']

        lines_cleared = self.ai_clear_lines()
        if lines_cleared > 0:
            scores = {1: 100, 2: 300, 3: 500, 4: 800}
            self.ai_score += scores.get(lines_cleared, 100)
            self.update_ai_score()

        self.ai_current_shape = self.ai_next_shape
        self.ai_next_shape = self.get_random_shape()
        self.ai_current_x = self.width // 2 - 2
        self.ai_current_y = 0
        self.ai_target_x = self.ai_current_x
        self.ai_target_rotation = 0

        if self.ai_check_collision(self.ai_current_shape, self.ai_current_x,
self.ai_current_y):
            self.ai_game_over = True
            self.ai_is_falling = False
            self.ai_draw()

            if self.game_mode == "vs_ai" and not self.returning_to_menu and not
self.game_ended:
                self.check_score_limit()
                victory = self.check_victory()
                if victory and not self.victory_shown:
                    self.show_victory_message(victory)
            return True

        return False

    def ai_clear_lines(self):
        lines_cleared = 0
        y = self.height - 1
        while y >= 0:
            if all(self.ai_board[y][x] != 0 for x in range(self.width)):
                for row in range(y, 0, -1):
                    self.ai_board[row] = self.ai_board[row-1].copy()
                self.ai_board[0] = [0 for _ in range(self.width)]
                lines_cleared += 1
            else:
                y -= 1
        return lines_cleared

    def ai_draw(self):
        if not hasattr(self, 'ai_canvas') or not self.ai_canvas.winfo_exists():
            return

        if not self.game_ended:
            self.ai_canvas.delete("shadow", "block", "current_shape",
"game_over_text")

        for y in range(self.height):
            for x in range(self.width):
                if self.ai_board[y][x] != 0:
                    self.draw_3d_block_on_canvas(self.ai_canvas, x, y,
self.ai_board[y][x], False)
```

```
        if not self.ai_game_over and self.ai_current_shape and not self.paused
and not self.game_ended:
            self.ai_draw_shadow()

        if not self.ai_game_over and self.ai_current_shape and not
self.game_ended:
            for x, y in self.ai_current_shape['blocks']:
                board_x = self.ai_current_x + x
                board_y = self.ai_current_y + y
                if 0 <= board_y < self.height and 0 <= board_x < self.width:
                    self.draw_3d_block_on_canvas(self.ai_canvas, board_x,
board_y, self.ai_current_shape['color'], True)

        if not self.game_ended:
            for x in range(0, self.width * self.cell_size, self.cell_size):
                self.ai_canvas.create_line(x, 0, x, self.height *
self.cell_size, fill="gray20", width=1)
            for y in range(0, self.height * self.cell_size, self.cell_size):
                self.ai_canvas.create_line(0, y, self.width * self.cell_size, y,
fill="gray20", width=1)

        if self.ai_game_over and not self.victory_shown and not self.game_ended:
            self.ai_canvas.create_text(self.width * self.cell_size // 2,
                                       self.height * self.cell_size // 2,
                                       text="GAME OVER", fill="red",
                                       font=("Arial", 16, "bold"),
                                       tags="game_over_text")

    def ai_draw_shadow(self):
        shadow_y = self.ai_calculate_shadow_position()

        for x, y in self.ai_current_shape['blocks']:
            board_x = self.ai_current_x + x
            board_y = shadow_y + y
            if 0 <= board_y < self.height and 0 <= board_x < self.width:
                if self.ai_board[board_y][board_x] == 0:
                    x1 = board_x * self.cell_size
                    y1 = board_y * self.cell_size
                    x2 = (board_x + 1) * self.cell_size
                    y2 = (board_y + 1) * self.cell_size

                    self.ai_canvas.create_rectangle(
                        x1, y1, x2, y2,
                        fill="#666666",
                        outline="",
                        stipple="gray50",
                        tags=("shadow",)
                    )

    def ai_calculate_shadow_position(self):
        shadow_y = self.ai_current_y
        while not self.ai_check_collision(self.ai_current_shape,
self.ai_current_x, shadow_y + 1):
            shadow_y += 1
        return shadow_y

    def draw_3d_block_on_canvas(self, canvas, x, y, color, is_current=False):
        x1 = x * self.cell_size
        y1 = y * self.cell_size
```

```
        x2 = (x + 1) * self.cell_size
        y2 = (y + 1) * self.cell_size

        light, main, dark, highlight = self.get_color_palette(color)

        canvas.create_rectangle(x1, y1, x2, y2, fill=main, outline="",
tags=("block",))
        canvas.create_polygon(x1, y1, x2, y1, x2-5, y1+5, x1+5, y1+5,
fill=light, outline="", tags=("block",))
        canvas.create_polygon(x1, y1, x1+5, y1+5, x1+5, y2-5, x1, y2,
fill=light, outline="", tags=("block",))
        canvas.create_polygon(x1, y2, x2, y2, x2-5, y2-5, x1+5, y2-5, fill=dark,
outline="", tags=("block",))
        canvas.create_polygon(x2, y1, x2, y2, x2-5, y2-5, x2-5, y1+5, fill=dark,
outline="", tags=("block",))
        canvas.create_polygon(x1+2, y1+2, x2-2, y1+2, x1+5, y1+5, x1+2, y1+5,
fill=highlight, outline="", tags=("block",))

        if is_current:
            canvas.create_rectangle(x1, y1, x2, y2, outline="white", width=2,
tags=("current_shape",))

    def check_score_limit(self):
        if self.score >= self.max_score:
            self.game_over = True
            if hasattr(self, 'pause_button') and
self.pause_button.winfo_exists():
                self.pause_button.config(state="disabled")
            if hasattr(self, 'game_loop_id'):
                self.master.after_cancel(self.game_loop_id)
                self.game_loop_id = None

        if self.ai_score >= self.max_score:
            if self.game_mode == "vs_ai" and not self.returning_to_menu:
                victory = self.check_victory()
                if victory and not self.victory_shown:
                    self.show_victory_message(victory)

    def check_victory(self):
        if self.score >= self.max_score and self.ai_score >= self.max_score:
            if self.score > self.ai_score:
                return "player_wins"
            elif self.ai_score > self.score:
                return "ai_wins"
            else:
                return "Ничья!"
        elif self.score >= self.max_score:
            return "player_wins"
        elif self.ai_score >= self.max_score:
            return "ai_wins"
        elif self.game_over and self.ai_game_over:
            if self.score > self.ai_score:
                return "player_wins"
            elif self.ai_score > self.score:
                return "ai_wins"
            else:
                return "Ничья!"
        elif self.game_over:
            return "ai_wins"
```

```
        elif self.ai_game_over:
            return "player_wins"
        return None

    def show_victory_message(self, result):
        if self.victory_shown or self.returning_to_menu:
            return

        self.victory_shown = True
        self.game_ended = True

        self.cancel_all_after()
        self.game_loop_id = None
        self.ai_loop_id = None
        self.ai_fall_loop_id = None

        if result == "player_wins":
            if hasattr(self, 'player_canvas') and
self.player_canvas.winfo_exists():
                self.player_canvas.delete("victory_message", "game_over_text")
                self.player_canvas.create_text(self.width * self.cell_size // 2,
                                               self.height * self.cell_size // 2
- 20,
                                               text="🎉 ПОБЕДА! 🎉", fill="gold",
                                               font=("Arial", 20, "bold"),
                                               tags="victory_message")
                self.init_confetti(self.player_canvas)
                self.animate_confetti(self.player_canvas)

            if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
                self.ai_canvas.delete("victory_message", "game_over_text")
                self.ai_canvas.create_text(self.width * self.cell_size // 2,
                                           self.height * self.cell_size // 2,
                                           text="☹ ПОРАЖЕНИЕ ☹", fill="red",
                                           font=("Arial", 20, "bold"),
                                           tags="victory_message")

        elif result == "ai_wins":
            if hasattr(self, 'player_canvas') and
self.player_canvas.winfo_exists():
                self.player_canvas.delete("victory_message", "game_over_text")
                self.player_canvas.create_text(self.width * self.cell_size // 2,
                                               self.height * self.cell_size // 2,
                                               text="☹ ПОРАЖЕНИЕ ☹", fill="red",
                                               font=("Arial", 20, "bold"),
                                               tags="victory_message")

            if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
                self.ai_canvas.delete("victory_message", "game_over_text")
                self.ai_canvas.create_text(self.width * self.cell_size // 2,
                                           self.height * self.cell_size // 2 -
20,
                                           text="😈 ПОБЕДА! 😈", fill="gold",
                                           font=("Arial", 20, "bold"),
                                           tags="victory_message")
                self.init_confetti(self.ai_canvas)
                self.animate_confetti(self.ai_canvas)

        elif result == "Ничья!":
```

```
            if hasattr(self, 'player_canvas') and
self.player_canvas.winfo_exists():
                self.player_canvas.delete("victory_message", "game_over_text")
                self.player_canvas.create_text(self.width * self.cell_size // 2,
                                               self.height * self.cell_size // 2,
                                               text="🧡 НИЧЬЯ! 🧡",
fill="orange",
                                               font=("Arial", 20, "bold"),
                                               tags="victory_message")

            if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
                self.ai_canvas.delete("victory_message", "game_over_text")
                self.ai_canvas.create_text(self.width * self.cell_size // 2,
                                           self.height * self.cell_size // 2,
                                           text="🧡 НИЧЬЯ! 🧡", fill="orange",
                                           font=("Arial", 20, "bold"),
                                           tags="victory_message")

    def start_game(self, mode):
        self.game_mode = mode
        self.returning_to_menu = False
        self.victory_shown = False
        self.game_ended = False

        self.cancel_all_after()
        self.force_stop_all_animations()

        self.frame_count = 0
        self.last_fps_update = time.time()
        self.current_fps = 0

        for widget in self.master.winfo_children():
            widget.destroy()

        self.create_game_interface()
        self.init_game()

        if mode == "vs_ai":
            self.init_ai_game()

        self.master.unbind("<Left>")
        self.master.unbind("<Right>")
        self.master.unbind("<Down>")
        self.master.unbind("<Up>")
        self.master.unbind("<space>")
        self.master.unbind("<p>")
        self.master.unbind("<P>")
        self.master.unbind("<Escape>")

        self.master.bind("<Left>", self.move_left)
        self.master.bind("<Right>", self.move_right)
        self.master.bind("<Down>", self.move_down)
        self.master.bind("<Up>", self.rotate_shape)
        self.master.bind("<space>", self.hard_drop)
        self.master.bind("<p>", self.toggle_pause)
        self.master.bind("<P>", self.toggle_pause)
        self.master.bind("<Escape>", self.return_to_menu)
        self.master.focus_set()
```

```
        self.animate_particles()
        self.game_loop()

    def create_game_interface(self):
        if self.game_mode == "vs_ai":
            main_container = tk.Frame(self.master)
            main_container.pack(fill=tk.BOTH, expand=True)

            player_frame = tk.Frame(main_container)
            player_frame.pack(side=tk.LEFT, padx=10, pady=10)

            tk.Label(player_frame, text="ИГРОК", font=("Arial", 14, "bold"),
                    fg="white", bg="gray").pack()

            self.player_canvas = tk.Canvas(player_frame, width=self.width *
self.cell_size,
                                           height=self.height * self.cell_size)
            self.player_canvas.pack()
            self.draw_gradient_background_for_canvas(self.player_canvas)

            ai_frame = tk.Frame(main_container)
            ai_frame.pack(side=tk.LEFT, padx=10, pady=10)

            tk.Label(ai_frame, text=f"ИИ ({self.ai_difficulty})", font=("Arial",
14, "bold"),
                    fg="cyan", bg="gray").pack()

            self.ai_canvas = tk.Canvas(ai_frame, width=self.width *
self.cell_size,
                                       height=self.height * self.cell_size)
            self.ai_canvas.pack()
            self.draw_gradient_background_for_canvas(self.ai_canvas)

            self.canvas = self.player_canvas

            self.info_panel = tk.Frame(main_container, bg="gray", width=200,
                                       height=self.height * self.cell_size)
            self.info_panel.pack(side=tk.RIGHT, fill=tk.Y, padx=10)
            self.info_panel.pack_propagate(False)

            tk.Label(self.info_panel, text=f"Цель: {self.max_score}",
                    font=("Arial", 12, "bold"), bg="gray",
fg="gold").pack(pady=5)

            self.score_label = tk.Label(self.info_panel, text="Счёт: 0",
                                        font=("Arial", 16), bg="gray",
fg="white")
            self.score_label.pack(pady=5)

            self.ai_score_label = tk.Label(self.info_panel, text="Счёт ИИ: 0",
                                           font=("Arial", 16), bg="gray",
fg="cyan")
            self.ai_score_label.pack(pady=5)

            tk.Label(self.info_panel, text=f"Сложность:
{self.current_difficulty}",
                    font=("Arial", 11), bg="gray", fg="yellow").pack(pady=10)
```

```
            tk.Frame(self.info_panel, height=2, bg="white").pack(fill=tk.X,
pady=10, padx=10)

            self.pause_button = tk.Button(self.info_panel, text="Пауза",
                                          command=self.toggle_pause,
bg="yellow",
                                          font=("Arial", 12, "bold"))
            self.pause_button.pack(pady=5)

            tk.Button(self.info_panel, text="Меню ИИ",
                      command=self.return_to_menu, bg="orange",
                      font=("Arial", 12, "bold")).pack(pady=5)

            self.new_game_button = tk.Button(self.info_panel, text="Новая игра",
                      command=self.restart_game, bg="lightgray",
                      font=("Arial", 12, "bold"))
            self.new_game_button.pack(pady=5)

            self.fps_label = tk.Label(
                self.info_panel,
                text="FPS: 0",
                font=("Arial", 9),
                bg="gray",
                fg="lightblue"
            )
            self.fps_label.pack(pady=5)

        else:
            main_container = tk.Frame(self.master)
            main_container.pack(fill=tk.BOTH, expand=True)

            self.canvas = tk.Canvas(main_container, width=self.width *
self.cell_size,
                                    height=self.height * self.cell_size)
            self.canvas.pack(side=tk.LEFT, padx=10, pady=10)
            self.draw_gradient_background_for_canvas(self.canvas)

            self.info_panel = tk.Frame(main_container, bg="gray", width=180,
                                       height=self.height * self.cell_size)
            self.info_panel.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)
            self.info_panel.pack_propagate(False)

            self.high_score_label = tk.Label(
                self.info_panel,
                text=f"Рекорд: {self.high_score}",
                font=("Arial", 14, "bold"),
                bg="gray",
                fg="gold"
            )
            self.high_score_label.pack(pady=10)

            self.score_label = tk.Label(self.info_panel, text="Счёт: 0",
                                        font=("Arial", 16), bg="gray",
fg="white")
            self.score_label.pack(pady=10)

            tk.Label(self.info_panel, text=f"Сложность:
{self.current_difficulty}",
                     font=("Arial", 12), bg="gray", fg="yellow").pack(pady=5)
```

```
            tk.Label(self.info_panel, text="Следующая:", font=("Arial", 12),
                    bg="gray", fg="white").pack()

            self.next_canvas = tk.Canvas(self.info_panel, width=120, height=120,
bg="black")
            self.next_canvas.pack(pady=10)

            self.pause_button = tk.Button(self.info_panel, text="Пауза",
                                          command=self.toggle_pause,
bg="yellow",
                                          font=("Arial", 12, "bold"))
            self.pause_button.pack(pady=10)

            tk.Button(self.info_panel, text="Главное меню",
                     command=self.return_to_menu, bg="orange",
                     font=("Arial", 12, "bold")).pack(pady=10)

            self.new_game_button = tk.Button(self.info_panel, text="Новая игра",
                     command=self.restart_game, bg="lightgray",
                     font=("Arial", 12, "bold"))
            self.new_game_button.pack(pady=10)

            self.fps_label = tk.Label(
                self.info_panel,
                text="FPS: 0",
                font=("Arial", 10, "bold"),
                bg="gray",
                fg="lightblue"
            )
            self.fps_label.pack(pady=5)

        self.shapes = {
            'I': [(0,1), (1,1), (2,1), (3,1)],
            'O': [(1,1), (2,1), (1,2), (2,2)],
            'T': [(1,0), (0,1), (1,1), (2,1)],
            'L': [(0,1), (1,1), (2,1), (2,0)],
            'J': [(0,1), (1,1), (2,1), (0,0)],
            'S': [(1,1), (2,1), (0,2), (1,2)],
            'Z': [(0,1), (1,1), (1,2), (2,2)]
        }

        self.colors = {
            'I': 'cyan', 'O': 'yellow', 'T': 'purple',
            'L': 'orange', 'J': 'blue', 'S': 'green', 'Z': 'red'
        }

    def draw_gradient_background_for_canvas(self, canvas):
        width = self.width * self.cell_size
        height = self.height * self.cell_size

        for i in range(height):
            ratio = i / height
            r = int(25 + 50 * ratio)
            g = int(0 + 0 * ratio)
            b = int(100 + 30 * ratio)
            color = f"#{r:02x}{g:02x}{b:02x}"
            canvas.create_line(0, i, width, i, fill=color, tags=("background",))
```

```
        for _ in range(150):
            x = random.randint(0, width)
            y = random.randint(0, height)
            size = random.randint(1, 2)
            brightness = random.randint(150, 255)
            color = f"#{brightness:02x}{brightness:02x}{brightness:02x}"
            canvas.create_oval(x, y, x + size, y + size, fill=color, outline="",
tags=("background",))

        for _ in range(50):
            x = random.randint(0, width)
            y = random.randint(0, height)
            canvas.create_oval(x, y, x + 1, y + 1, fill="cyan", outline="",
tags=("background", "particle"))

    def get_random_shape(self):
        shape_name = random.choice(list(self.shapes.keys()))
        return {
            'name': shape_name,
            'blocks': [list(block) for block in self.shapes[shape_name]],
            'color': self.colors[shape_name]
        }

    def init_game(self):
        self.board = [[0 for _ in range(self.width)] for _ in
range(self.height)]
        self.score = 0
        self.game_over = False
        self.paused = False
        self.victory_shown = False
        self.game_ended = False
        if hasattr(self, 'pause_button') and self.pause_button.winfo_exists():
            self.pause_button.config(text="Пауза", bg="yellow", state="normal")
        self.current_shape = self.get_random_shape()
        self.current_x = self.width // 2 - 2
        self.current_y = 0
        self.next_shape = self.get_random_shape()
        if hasattr(self, 'score_label') and self.score_label.winfo_exists():
            self.update_score()
        if self.game_mode != "vs_ai" and hasattr(self, 'next_canvas'):
            self.draw_next_shape()
        self.draw()

    def return_to_menu(self, event=None):
        if self.returning_to_menu:
            return

        self.returning_to_menu = True

        self.cancel_all_after()
        self.force_stop_all_animations()

        self.master.unbind("<Left>")
        self.master.unbind("<Right>")
        self.master.unbind("<Down>")
        self.master.unbind("<Up>")
        self.master.unbind("<space>")
        self.master.unbind("<p>")
        self.master.unbind("<P>")
```

```
        self.master.unbind("<Escape>")

        # Если мы в режиме против ИИ, возвращаемся в меню ИИ
        if self.game_mode == "vs_ai":
            self.open_ai_menu()
        else:
            self.create_main_menu()

    def check_collision(self, shape, offset_x, offset_y):
        for x, y in shape['blocks']:
            new_x = self.current_x + x + offset_x
            new_y = self.current_y + y + offset_y
            if new_x < 0 or new_x >= self.width or new_y >= self.height:
                return True
            if new_y < 0:
                continue
            if new_y >= 0 and self.board[new_y][new_x] != 0:
                return True
        return False

    def merge_shape(self):
        for x, y in self.current_shape['blocks']:
            board_x = self.current_x + x
            board_y = self.current_y + y
            if 0 <= board_y < self.height and 0 <= board_x < self.width:
                self.board[board_y][board_x] = self.current_shape['color']

        self.clear_lines()
        self.current_shape = self.next_shape
        self.next_shape = self.get_random_shape()
        self.current_x = self.width // 2 - 2
        self.current_y = 0

        if self.check_collision(self.current_shape, 0, 0):
            self.game_over = True
            if hasattr(self, 'pause_button') and
self.pause_button.winfo_exists():
                self.pause_button.config(state="disabled")

            if hasattr(self, 'game_loop_id'):
                self.master.after_cancel(self.game_loop_id)
                self.game_loop_id = None

            self.draw()

            if self.game_mode == "vs_ai" and not self.returning_to_menu and not
self.game_ended:
                self.check_score_limit()
                victory = self.check_victory()
                if victory and not self.victory_shown:
                    self.show_victory_message(victory)
            return True

        if self.game_mode == "vs_ai" and not self.returning_to_menu and not
self.game_ended:
            self.check_score_limit()
            victory = self.check_victory()
            if victory and not self.victory_shown:
                self.show_victory_message(victory)
```

```
        if self.game_mode != "vs_ai" and hasattr(self, 'next_canvas'):
            self.draw_next_shape()
        self.draw()
        return False

    def clear_lines(self):
        lines_cleared = 0
        y = self.height - 1
        while y >= 0:
            if all(self.board[y][x] != 0 for x in range(self.width)):
                for row in range(y, 0, -1):
                    self.board[row] = self.board[row-1].copy()
                self.board[0] = [0 for _ in range(self.width)]
                lines_cleared += 1
            else:
                y -= 1

        if lines_cleared > 0:
            scores = {1: 100, 2: 300, 3: 500, 4: 800}
            self.score += scores.get(lines_cleared, 100)
            if hasattr(self, 'score_label') and self.score_label.winfo_exists():
                self.update_score()

    def rotate_shape(self, event=None):
        if self.game_over or self.paused:
            return
        rotated_blocks = []
        for x, y in self.current_shape['blocks']:
            rotated_blocks.append([y, -x])
        min_x = min(block[0] for block in rotated_blocks)
        min_y = min(block[1] for block in rotated_blocks)
        rotated_blocks = [[x - min_x, y - min_y] for x, y in rotated_blocks]
        rotated_shape = {
            'name': self.current_shape['name'],
            'blocks': rotated_blocks,
            'color': self.current_shape['color']
        }
        if not self.check_collision(rotated_shape, 0, 0):
            self.current_shape = rotated_shape
            self.draw()

    def move_left(self, event=None):
        if not self.game_over and not self.paused:
            if not self.check_collision(self.current_shape, -1, 0):
                self.current_x -= 1
                self.draw()

    def move_right(self, event=None):
        if not self.game_over and not self.paused:
            if not self.check_collision(self.current_shape, 1, 0):
                self.current_x += 1
                self.draw()

    def move_down(self, event=None):
        if self.game_over or self.paused:
            return False
        if not self.check_collision(self.current_shape, 0, 1):
            self.current_y += 1
```

```
            self.draw()
            return False
        else:
            game_over = self.merge_shape()
            self.draw()
            return game_over

    def hard_drop(self, event=None):
        if not self.game_over and not self.paused:
            while not self.check_collision(self.current_shape, 0, 1):
                self.current_y += 1
            self.merge_shape()
            self.draw()

    def toggle_pause(self, event=None):
        if self.game_over:
            return
        self.paused = not self.paused
        if self.paused:
            if hasattr(self, 'pause_button') and
self.pause_button.winfo_exists():
                self.pause_button.config(text="Продолжить", bg="lightgreen")
            if hasattr(self, 'game_loop_id'):
                self.master.after_cancel(self.game_loop_id)
                self.game_loop_id = None
            if hasattr(self, 'ai_loop_id'):
                self.master.after_cancel(self.ai_loop_id)
                self.ai_loop_id = None
            if hasattr(self, 'ai_fall_loop_id'):
                self.master.after_cancel(self.ai_fall_loop_id)
                self.ai_fall_loop_id = None
            if hasattr(self, 'particle_animation_id'):
                self.master.after_cancel(self.particle_animation_id)
                self.particle_animation_id = None
            if hasattr(self, 'confetti_animation_id'):
                self.master.after_cancel(self.confetti_animation_id)
                self.confetti_animation_id = None
            self.draw_pause_message()
        else:
            if hasattr(self, 'pause_button') and
self.pause_button.winfo_exists():
                self.pause_button.config(text="Пауза", bg="yellow")
            if hasattr(self, 'canvas') and self.canvas.winfo_exists():
                self.canvas.delete("pause_text")
            if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
                self.ai_canvas.delete("pause_text")
            self.animate_particles()
            self.game_loop()
            if self.game_mode == "vs_ai" and not self.ai_game_over:
                self.ai_start_fall_loop()
                self.schedule_ai_decision()

    def draw_pause_message(self):
        if hasattr(self, 'canvas') and self.canvas.winfo_exists():
            self.canvas.create_text(self.width * self.cell_size // 2,
                                    self.height * self.cell_size // 2,
                                    text="ПАУЗА", fill="yellow",
                                    font=("Arial", 24, "bold"),
tag="pause_text")
```

```
        if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
            self.ai_canvas.create_text(self.width * self.cell_size // 2,
                                       self.height * self.cell_size // 2,
                                       text="ПАУЗА", fill="yellow",
                                       font=("Arial", 24, "bold"),
tag="pause_text")

    def restart_game(self):
        self.cancel_all_after()
        self.force_stop_all_animations()

        self.paused = False
        self.game_over = False
        self.ai_is_moving = False
        self.ai_is_falling = True
        self.ai_soft_dropping = False
        self.returning_to_menu = False
        self.victory_shown = False
        self.game_ended = False

        self.board = [[0 for _ in range(self.width)] for _ in
range(self.height)]
        self.score = 0

        if hasattr(self, 'pause_button') and self.pause_button.winfo_exists():
            self.pause_button.config(text="Пауза", bg="yellow", state="normal")

        if hasattr(self, 'canvas') and self.canvas.winfo_exists():
            self.canvas.delete("pause_text", "victory_message",
"game_over_text", "shadow", "block", "current_shape")
        if hasattr(self, 'ai_canvas') and self.ai_canvas.winfo_exists():
            self.ai_canvas.delete("pause_text", "victory_message",
"game_over_text", "shadow", "block", "current_shape")
        if hasattr(self, 'player_canvas') and self.player_canvas.winfo_exists():
            self.player_canvas.delete("pause_text", "victory_message",
"game_over_text")

        self.current_shape = self.get_random_shape()
        self.current_x = self.width // 2 - 2
        self.current_y = 0
        self.next_shape = self.get_random_shape()

        if hasattr(self, 'score_label') and self.score_label.winfo_exists():
            self.update_score()

        if self.game_mode != "vs_ai" and hasattr(self, 'next_canvas') and
self.next_canvas.winfo_exists():
            self.draw_next_shape()

        if self.game_mode == "vs_ai":
            self.ai_board = [[0 for _ in range(self.width)] for _ in
range(self.height)]
            self.ai_game_over = False
            self.ai_is_moving = False
            self.ai_is_falling = True
            self.ai_soft_dropping = False
            self.ai_score = 0
            self.ai_current_shape = self.get_random_shape()
            self.ai_next_shape = self.get_random_shape()
```

```
            self.ai_current_x = self.width // 2 - 2
            self.ai_current_y = 0
            self.ai_target_x = self.ai_current_x
            self.ai_target_rotation = 0

            self.update_ai_score()

            self.ai_start_fall_loop()
            self.schedule_ai_decision()

        self.draw()
        if self.game_mode == "vs_ai":
            self.ai_draw()

        self.game_loop()

    def update_score(self):
        if hasattr(self, 'score_label') and self.score_label.winfo_exists():
            self.score_label.config(text=f"Счёт: {self.score}")

        if self.score > self.high_score:
            self.high_score = self.score
            self.save_high_score()
            if hasattr(self, 'high_score_label') and
self.high_score_label.winfo_exists():
                self.high_score_label.config(text=f"Рекорд: {self.high_score}")

    def draw_next_shape(self):
        if not hasattr(self, 'next_canvas') or not
self.next_canvas.winfo_exists():
            return
        self.next_canvas.delete("all")
        cell_size = 25
        blocks = self.next_shape['blocks']
        color = self.next_shape['color']
        if not blocks:
            return
        min_x = min(x for x, y in blocks)
        max_x = max(x for x, y in blocks)
        min_y = min(y for x, y in blocks)
        max_y = max(y for x, y in blocks)
        shape_width = (max_x - min_x + 1) * cell_size
        shape_height = (max_y - min_y + 1) * cell_size
        offset_x = (120 - shape_width) // 2 - min_x * cell_size
        offset_y = (120 - shape_height) // 2 - min_y * cell_size

        light, main, dark, highlight = self.get_color_palette(color)

        for x, y in blocks:
            x1 = offset_x + x * cell_size
            y1 = offset_y + y * cell_size
            x2 = x1 + cell_size
            y2 = y1 + cell_size

            self.next_canvas.create_rectangle(x1, y1, x2, y2, fill=main,
outline="")
            self.next_canvas.create_polygon(x1, y1, x2, y1, x2-4, y1+4, x1+4,
y1+4, fill=light, outline="")
```

```
            self.next_canvas.create_polygon(x1, y1, x1+4, y1+4, x1+4, y2-4, x1,
y2, fill=light, outline="")
            self.next_canvas.create_polygon(x1, y2, x2, y2, x2-4, y2-4, x1+4,
y2-4, fill=dark, outline="")
            self.next_canvas.create_polygon(x2, y1, x2, y2, x2-4, y2-4, x2-4,
y1+4, fill=dark, outline="")
            self.next_canvas.create_rectangle(x1, y1, x2, y2, outline="white",
width=1)

    def update_fps(self):
        self.frame_count += 1
        current_time = time.time()
        if current_time - self.last_fps_update >= 1.0:
            self.current_fps = self.frame_count
            self.frame_count = 0
            self.last_fps_update = current_time
            if hasattr(self, 'fps_label') and self.fps_label.winfo_exists():
                self.fps_label.config(text=f"FPS: {self.current_fps}")

    def animate_particles(self):
        if hasattr(self, 'canvas') and not self.game_over and not self.paused
and not self.returning_to_menu:
            if self.game_mode == "vs_ai":
                for canvas in [self.player_canvas, self.ai_canvas]:
                    if canvas.winfo_exists():
                        particles = canvas.find_withtag("particle")
                        for particle in particles:
                            if random.random() < 0.1:
                                colors = ['cyan', 'lightblue', 'white',
'lightgreen', 'yellow', 'pink']
                                canvas.itemconfig(particle,
fill=random.choice(colors))
            else:
                particles = self.canvas.find_withtag("particle")
                for particle in particles:
                    if random.random() < 0.1:
                        colors = ['cyan', 'lightblue', 'white', 'lightgreen',
'yellow', 'pink']
                        self.canvas.itemconfig(particle,
fill=random.choice(colors))

            self.particle_animation_id = self.safe_after(500,
self.animate_particles)

    def get_color_palette(self, base_color):
        color_map = {
            'red': ('#ff6666', '#ff0000', '#990000', '#ff9999'),
            'deep sky blue': ('#66ccff', '#00bfff', '#006699', '#99ddff'),
            'light green': ('#88ff88', '#00ff00', '#009900', '#aaffaa'),
            'yellow': ('#ffff66', '#ffff00', '#999900', '#ffff99'),
            'pink': ('#ff99bb', '#ff69b4', '#993366', '#ffbbcc'),
            'orange': ('#ffaa66', '#ffa500', '#996600', '#ffcc99'),
            'cyan': ('#66ffff', '#00ffff', '#009999', '#aaffff'),
            'purple': ('#bb66ff', '#9932cc', '#660099', '#dd99ff'),
            'blue': ('#6699ff', '#0000ff', '#000099', '#99bbff'),
            'green': ('#66ff66', '#00cc00', '#006600', '#aaffaa')
        }
        return color_map.get(base_color, ('#ffffff', '#cccccc', '#666666',
'#eeeeee'))
```

```
    def draw_3d_block(self, x, y, color, is_current=False):
        if hasattr(self, 'canvas') and self.canvas.winfo_exists():
            self.draw_3d_block_on_canvas(self.canvas, x, y, color, is_current)

    def draw_shadow(self):
        shadow_y = self.calculate_shadow_position()

        for x, y in self.current_shape['blocks']:
            board_x = self.current_x + x
            board_y = shadow_y + y
            if 0 <= board_y < self.height and 0 <= board_x < self.width:
                if self.board[board_y][board_x] == 0:
                    x1 = board_x * self.cell_size
                    y1 = board_y * self.cell_size
                    x2 = (board_x + 1) * self.cell_size
                    y2 = (board_y + 1) * self.cell_size

                    self.canvas.create_rectangle(
                        x1, y1, x2, y2,
                        fill="#666666",
                        outline="",
                        stipple="gray50",
                        tags=("shadow",)
                    )

    def calculate_shadow_position(self):
        shadow_y = self.current_y
        while not self.check_collision(self.current_shape, 0, shadow_y + 1 -
self.current_y):
            shadow_y += 1
        return shadow_y

    def draw(self):
        if not hasattr(self, 'canvas') or not self.canvas.winfo_exists():
            return

        self.update_fps()
        self.canvas.delete("shadow", "block", "current_shape")

        for y in range(self.height):
            for x in range(self.width):
                if self.board[y][x] != 0:
                    self.draw_3d_block(x, y, self.board[y][x], is_current=False)

        if not self.game_over and not self.paused and self.current_shape:
            self.draw_shadow()

        if not self.game_over and self.current_shape:
            for x, y in self.current_shape['blocks']:
                board_x = self.current_x + x
                board_y = self.current_y + y
                if 0 <= board_y < self.height and 0 <= board_x < self.width:
                    self.draw_3d_block(board_x, board_y,
self.current_shape['color'], is_current=True)

        for x in range(0, self.width * self.cell_size, self.cell_size):
            self.canvas.create_line(x, 0, x, self.height * self.cell_size,
fill="gray20", width=1)
```

```
        for y in range(0, self.height * self.cell_size, self.cell_size):
            self.canvas.create_line(0, y, self.width * self.cell_size, y,
fill="gray20", width=1)

        if self.game_over and not self.victory_shown:
            self.canvas.create_text(self.width * self.cell_size // 2,
                                    self.height * self.cell_size // 2,
                                    text="GAME OVER", fill="red",
                                    font=("Arial", 20, "bold"),
                                    tags="game_over_text")
        elif self.paused:
            self.draw_pause_message()

        if self.game_mode == "vs_ai" and not self.returning_to_menu:
            victory = self.check_victory()
            if victory and not self.victory_shown:
                self.show_victory_message(victory)

    def game_loop(self):
        if not self.game_over and not self.paused and not self.returning_to_menu
and not self.game_ended:
            self.move_down()
            if not self.game_over:
                self.game_loop_id = self.safe_after(self.fall_speed,
self.game_loop)

def main():
    root = tk.Tk()
    game = Tetris(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```