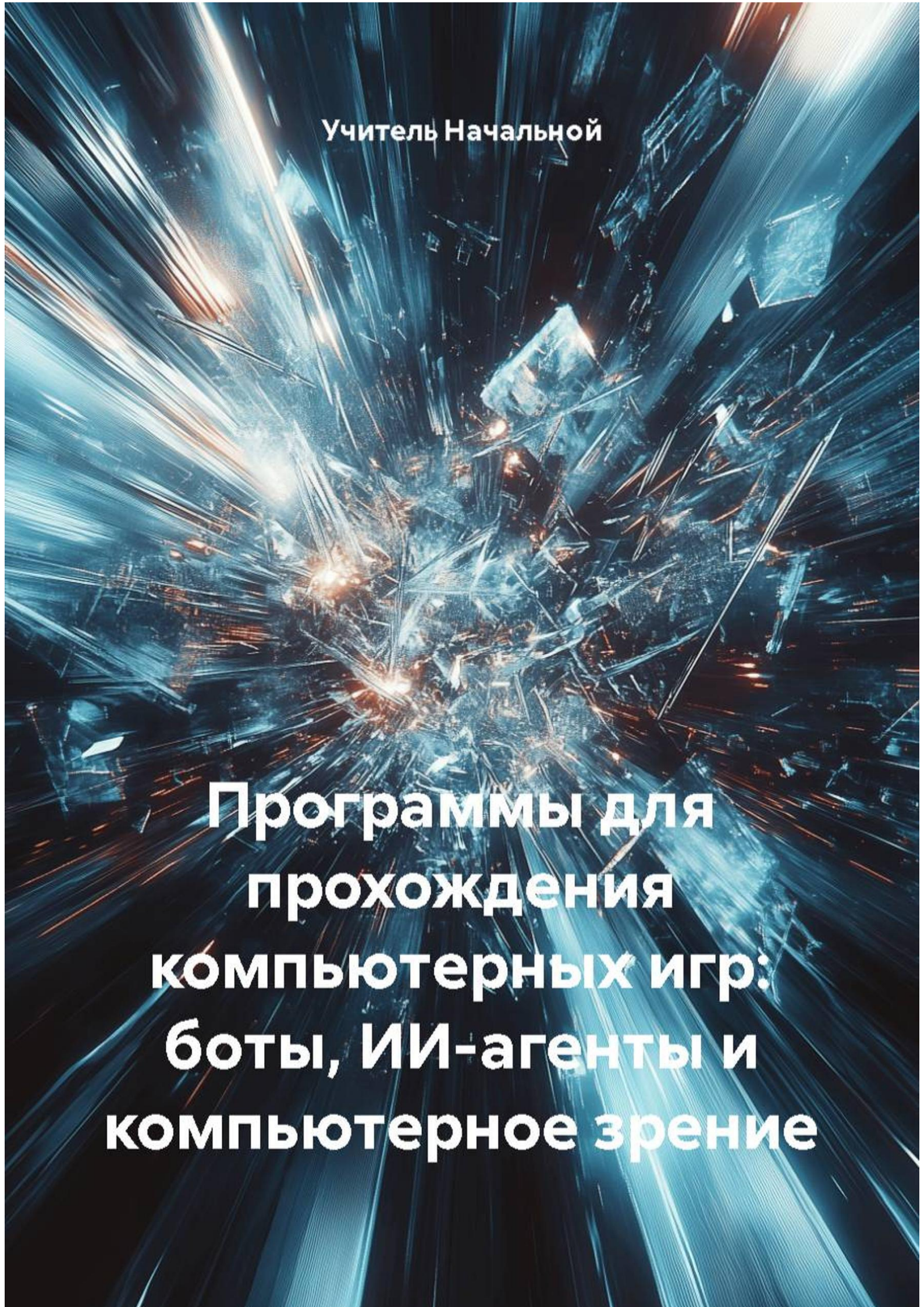




Учитель Начальной

**Программы для
прохождения
компьютерных игр:
боты, ИИ-агенты и
компьютерное зрение**

Учитель Начальной

**Программы для прохождения
компьютерных игр: боты, ИИ-
агенты и компьютерное зрение**

«Автор»

2026

Начальной У. Ш.

Программы для прохождения компьютерных игр: боты, ИИ-агенты и компьютерное зрение / У. Ш. Начальной — «Автор», 2026

Книга «Программы для прохождения компьютерных игр: боты, ИИ-агенты и компьютерное зрение» — практическое руководство по созданию программ, которые самостоятельно играют или помогают игроку. В 20 главах разбираются все основные подходы: от простых скриптов и макросов до vision-ботов на OpenCV и YOLO, reinforcement learning агентов и современных мультимодальных моделей (VLM/LLM). Читатель узнает, как захватывать экран, распознавать объекты и интерфейс, симулировать ввод, обучать нейросети и строить гибридные системы. Отдельные главы посвящены истории игрового ИИ, классификации ботов, античитам, этике и реальным open-source проектам (SIMA, YOLO-aim, MAA, Mineflayer и др.). Издание ориентировано на программистов, исследователей ИИ и энтузиастов, которые хотят понять, как устроены современные игровые боты, и научиться создавать их в образовательных и исследовательских целях.

Содержание

Глава	5
Конец ознакомительного фрагмента.	33

Учитель Начальной Программы для прохождения компьютерных игр: боты, ИИ- агенты и компьютерное зрение

Глава

ГЛАВА 1. Введение в автоматизацию компьютерных игр

Автоматизация компьютерных игр — междисциплинарная область на стыке программирования, компьютерного зрения, машинного обучения, теории игр и программной инженерии. Её цель — создание программных систем (ботов, агентов, ассистентов), способных воспринимать игровое окружение, принимать решения и выполнять действия с минимальным участием человека или полностью автономно.

Игры представляют собой уникальную среду для разработки и тестирования интеллектуальных систем. В отличие от физического мира, игровая среда полностью контролируема, воспроизводима и безопасна. При этом многие игры моделируют сложные аспекты реальности: частичную наблюдаемость (туман войны, ограниченное поле зрения), стохастичность, необходимость долгосрочного планирования, конкуренцию и кооперацию нескольких агентов. Именно поэтому крупнейшие лаборатории искусственного интеллекта — DeepMind, OpenAI, Meta AI и другие — десятилетиями используют игры в качестве основного полигона для отработки алгоритмов.

1.1. Что такое игровой бот

Игровой бот — это программа, которая взаимодействует с компьютерной игрой от имени игрока или вместо него. Бот может выполнять самые разные функции:

- Полностью автономная игра — от начала до конца уровня, матча или даже всей кампании.
- Выполнение рутинных задач — фарм ресурсов, прокачка персонажа, сбор и разбор лута, выполнение ежедневных и еженедельных заданий.
- Помощь игроку в реальном времени — подсказки, автоприцеливание, автоматическое использование способностей и расходников, помощь в уклонении.
- Тестирование игры — поиск багов, проверка баланса, регрессионное тестирование после патчей, нагрузочное тестирование серверов.
- Исследование алгоритмов ИИ — обучение агентов, сравнение архитектур, бенчмаркинг новых методов perception и control.

Важно различать понятия «бот» и «читерский инструмент». Классические читы обычно дают нечестное преимущество за счёт нарушения правил клиента: чтение скрытой памяти, wallhack, speedhack, инъекция кода в процесс игры. Vision-based бот работает принципиально иначе — он анализирует только то изображение, которое видит человек на экране, и эмулирует действия мыши и клавиатуры. Такие системы ближе к настоящему искусственному интеллекту

и гораздо сложнее детектируются классическими сигнатурными античитами, хотя поведенческий анализ всё равно может их выявить.

1.2. Зачем нужна автоматизация игр

Практических причин заниматься этой областью множество, и они далеко не сводятся к желанию «фармить без усилий».

Во-первых, исследования искусственного интеллекта. Игры — идеальный тест-бед. Atari 2600, Doom, StarCraft II, Dota 2, Minecraft, NetHack — все эти среды стали стандартными бенчмарками. Успехи AlphaGo, OpenAI Five и AlphaStar убедительно доказали, что агенты могут превосходить лучших людей в чрезвычайно сложных доменах. Сегодня исследователи переходят к ещё более общим задачам: агентам, которые могут играть в ранее невиданные игры, следуя инструкциям на естественном языке. Яркий пример — проект SIMA (Scalable Instructable Multiworld Agent) от DeepMind.

Во-вторых, тестирование и обеспечение качества игр. Современные AAA-проекты содержат сотни часов контента и астрономическое количество возможных состояний. Ручное тестирование становится неэффективным и дорогим. Автоматические агенты могут проходить уровни тысячи раз, искать редкие edge-cases, проверять, не ломается ли прогресс после патча, и собирать статистику по балансу и производительности.

В-третьих, доступность. Люди с ограниченными возможностями движения часто не могут полноценно играть в игры, требующие быстрой реакции или сложного одновременного управления несколькими клавишами и мышью. Ассистенты на базе компьютерного зрения и голосового управления способны значительно снизить порог входа и сделать хобби доступнее.

В-четвёртых, автоматизация рутины. В ММО, idle-играх, gacha и многих live-service проектах значительная часть времени уходит на повторяющиеся действия. Боты позволяют игрокам сосредоточиться на интересном контенте, а не на гринде. Разумеется, использование таких инструментов почти всегда нарушает правила сервиса и может привести к блокировке аккаунта.

В-пятых, обучение. Написание бота — один из лучших способов изучить компьютерное зрение, работу с изображениями в реальном времени, основы reinforcement learning и инженерии программного обеспечения. Обратная связь мгновенная и наглядная: бот либо успешно проходит уровень, либо нет. Ошибки в perception или логике сразу видны на экране.

1.3. Основные подходы к автоматизации

Существует несколько фундаментально разных способов заставить программу «играть». Понимание этих подходов критично для выбора архитектуры под конкретную задачу.

Скриптовые боты и макросы. Самый простой уровень. Программа записывает последовательность нажатий клавиш и движений мыши и затем воспроизводит её. Инструменты: AutoHotkey, AutoIt, pyautogui, Keyboard Maestro. Такие боты крайне хрупкие — любое изменение интерфейса, разрешения или тайминга ломает сценарий. Тем не менее для очень предсказуемых задач (сбор ежедневных наград, простые кликеры, повторяющиеся комбинации в single-player) они до сих пор используются и имеют право на жизнь как первый шаг.

Memory-based боты. Программа читает и иногда изменяет оперативную память процесса игры. Это позволяет получать точные значения HP, координат, идентификаторов объектов, состояния анимаций без какого-либо анализа картинки. Подход требует reverse engineering: поиск указателей, сигнатур байт-кода, разбор структур данных (часто с помощью Cheat Engine и дизассемблеров). Плюсы — скорость и точность. Минусы — хрупкость после обновлений игры, практически гарантированная детекция современными античитами, юридические и этические риски.

Network / Packet bots. Перехват и подмена сетевых пакетов между клиентом и сервером. Классика для старых ММО. Сегодня встречается реже из-за шифрования трафика, обфускации протоколов и усиленных server-side проверок.

Vision-based боты. Самый «честный» и интересный с точки зрения искусственного интеллекта подход. Бот видит ровно то же, что и человек — пиксели на экране. Он анализирует изображение с помощью классических алгоритмов компьютерного зрения или нейросетей, принимает решение и эмулирует действия мыши и клавиатуры. Поскольку бот не вмешивается в процесс игры, античитам гораздо сложнее обнаружить его по сигнатурам и факту чтения памяти. Основная сложность — добиться достаточной скорости и надёжности распознавания в условиях меняющегося освещения, эффектов и UI.

API-based и SDK-based решения. Некоторые игры предоставляют официальные или полуофициальные интерфейсы. Примеры: python-sc2 и pysc2 для StarCraft II, Mineflayer для Minecraft, различные бот-API в старых многопользовательских шутерах. Это самый удобный путь, когда он доступен: вы получаете структурированные данные о мире и можете сосредоточиться на логике, а не на perception.

Reinforcement Learning и end-to-end агенты. Агент обучается напрямую на пикселях или на состоянии игры методом проб и ошибок, максимизируя заданную функцию награды. Это вершина пирамиды сложности и вычислительных затрат. Именно так были созданы агенты, победившие чемпионов мира в го, Dota 2 и StarCraft II.

Гибридные системы. На практике лучшие современные боты комбинируют несколько подходов. Дешёвые локальные эвристики и классическое компьютерное зрение обрабатывают 95–98 % кадров и ситуаций. Тяжёлые нейросети или большие языковые модели вызываются только тогда, когда нужно принять сложное стратегическое решение, разобрать нестандартную ситуацию или спланировать последовательность действий на минуты вперёд.

1.4. Архитектура типичного vision-бота

Большинство vision-based ботов строятся по похожей многослойной схеме:

1. Захват кадра (screen capture) — получение изображения окна игры или нужной области экрана с минимальной задержкой.
2. Предобработка изображения — приведение к нужному размеру и цветовому пространству, выделение ROI, фильтрация.
3. Восприятие (perception) — детекция объектов, классификация общего состояния экрана, OCR, трекинг целей между кадрами.

4. Представление состояния (state representation) — преобразование сырых выходов CV в структурированное описание: «где кто находится, сколько HP, какая фаза боя, какие кнопки доступны».

5. Принятие решения (decision making) — state machine, behaviour tree, нейросетевая политика или вызов LLM.

6. Планирование действий (action planning) — разбиение высокоуровневого решения на конкретные команды ввода.

7. Исполнение (execution) — симуляция движений мыши и нажатий клавиш с учётом требований «человечности».

8. Обратная связь и логирование — запись того, что произошло, для отладки и последующего улучшения.

Каждый из этих блоков можно реализовывать с разной степенью сложности — от пары строк на OpenCV до полноценных нейросетевых пайплайнов с GPU-ускорением. В следующих главах мы разберём каждый слой подробно.

1.5. Для кого эта книга

Книга рассчитана на читателей, которые уже знакомы с основами программирования (желательно на Python) и хотят углубиться в практическое применение компьютерного зрения и искусственного интеллекта. Предыдущий опыт в геймдеве или машинном обучении полезен, но не обязателен. Все ключевые концепции объясняются с нуля и сопровождаются примерами и рекомендациями по дальнейшему изучению.

Мы сознательно делаем акцент на vision-based и RL-подходах, а также на современных мультимодальных моделях. Memory-hacking и чисто читерские техники рассматриваются лишь обзорно: цель книги — понимание технологий восприятия и принятия решений, а не создание инструментов для нарушения правил онлайн-сервисов.

В следующих главах мы последовательно пройдем весь путь: от истории и классификации ботов через основы компьютерного зрения и симуляции ввода к полноценным системам на базе глубокого обучения и большим языковым моделям.

ГЛАВА 2. История ботов и игрового ИИ

История игрового искусственного интеллекта почти так же стара, как и история самих компьютеров. Задолго до появления графических интерфейсов и трёхмерных миров люди уже пытались научить машины играть в игры.

2.1. Первые шаги (1950–1970-е)

В 1951 году Кристофер Стрейчи написал программу для игры в ним на компьютере Ferranti Mark 1 в Манчестерском университете. Годом позже Александр Дуглас создал ОХО — крестики-нолики на EDSAC. Эти программы использовали полный перебор возможных ходов или простые эвристики. В те же годы Алан Тьюринг и Клод Шеннон независимо размышляли о том, как научить машину играть в шахматы. Тьюринг описал алгоритм Turochamp, который, впрочем, так и не был полностью реализован на работающем компьютере при его жизни.

Шахматные программы стали главной «витриной» игрового ИИ на десятилетия. Путь от простых минимаксных алгоритмов с альфа-бета отсечением до Deep Blue, победившего Гарри Каспарова в 1997 году, занял почти полвека. Важно понимать: Deep Blue победил за счёт огромной вычислительной мощности и тщательно настроенных эвристик оценки позиции, а не за счёт машинного обучения в современном смысле. Это был триумф инженерии поиска и оценки, а не обучения.

2.2. Эра аркад и конечных автоматов (1970–1990-е)

С появлением аркадных автоматов и домашних консолей возникла необходимость создавать противников, которые ведут себя интересно, но не являются ни беспомощными, ни непобедимыми. Классический пример — Pac-Man (1980). Четыре призрака — Blinky, Pinky, Inky и Clyde — имели разные алгоритмы выбора цели. Blinky всегда преследовал игрока напрямую. Pinky пытался устроить засаду в четырёх клетках впереди по направлению движения игрока. Inky использовал более сложную формулу, зависящую от положения и Blinky, и игрока. Clyde вблизи игрока уходил в свой «домашний» угол. Всё это было реализовано через конечные автоматы и простые правила — никаких нейросетей, но поведение уже казалось «живым».

В 1990-е годы расцвели шутеры от первого лица. Quake, Unreal Tournament, Half-Life породили целую культуру ботов. Появились фреймворки вроде Omni-bot, которые позволяли создавать ИИ-противников для многих движков. Бот должен был уметь передвигаться по карте, подбирать оружие и броню, вести огонь, использовать укрытия и иногда даже общаться в чате шаблонными фразами. В 2008–2012 годах проводился турнир 2K BotPrize — своеобразный тест Тьюринга для игровых ботов. Цель состояла в том, чтобы судьи-люди не смогли отличить бота от человека по стилю игры в Unreal Tournament 2004. В 2012 году несколько команд впервые превзошли этот порог.

2.3. Эпоха массовых ММО и фарм-ботов (2000–2010-е)

С взрывным ростом World of Warcraft, Lineage, Runescape и других ММО появился огромный спрос на автоматизацию. Игроки хотели прокачивать персонажей и зарабатывать внутриигровую валюту, не сидя у экрана сутками. Возникла целая индустрия: от простых кликеров до сложных ботов, читающих память клиента и обходящих защиту.

Разработчики игр ответили созданием всё более изощрённых античитов. Warden в World of Warcraft, затем BattlEye, Easy Anti-Cheat, PunkBuster, Valve Anti-Cheat. Началась классическая гонка вооружений: ботоделы находили новые уязвимости, античиты закрывали их, и так по кругу. Многие методы того времени — DLL-инъекции, хуки API, прямое чтение памяти — сегодня практически гарантированно приводят к бану в современных защищённых играх.

2.4. Революция глубокого обучения (2013–2019)

Переломным моментом стали 2013–2015 годы. DeepMind опубликовал работу о DQN — глубокой Q-сети, которая научилась играть в десятки игр Atari, получая на вход только пиксели экрана и счёт. Это было первое убедительное доказательство того, что end-to-end обучение с подкреплением на сырых пикселях возможно и даёт результат, сопоставимый с человеком в ряде игр.

В 2016 году AlphaGo победил Ли Седоля в го — игре, которую долго считали слишком сложной для машин из-за огромного пространства состояний и важности «интуиции». Затем последовали OpenAI Five (Dota 2, 2018) и AlphaStar (StarCraft II, 2019). Эти системы уже работали в условиях частичной наблюдаемости, командной игры и очень длинных эпизодов, длящихся десятки минут.

Параллельно развивались исследовательские среды: OpenAI Gym (позже Gymnasium), ViZDoom, Unity ML-Agents, MineRL, NetHack Learning Environment. Появилась возможность сравнивать алгоритмы на стандартизированных бенчмарках и воспроизводить результаты других групп.

2.5. Современный этап (2020–2026)

После 2020 года акцент сместился в двух направлениях.

Первое — создание всё более общих агентов. DeepMind представил SIMA — систему, которая может выполнять инструкции на естественном языке в разных 3D-играх, в том числе в тех, на которых не обучалась напрямую. Microsoft показал Muse — модель, способную генерировать согласованные последовательности геймплея по скриншоту. Исследователи работают над агентами, которые переносят навыки между доменами.

Второе — массовое появление vision-based и LLM-powered ассистентов. Благодаря доступности мощных детекторов объектов (семейство YOLO) и мультимодальных моделей (GPT-4o, Claude, Gemini, LLaVA, Qwen-VL) стало возможным создавать ботов, которые «видят» экран почти как человек и рассуждают о происходящем на языке. Появились гибридные архитектуры, где дешёвые локальные методы обрабатывают большую часть кадров, а тяжёлые модели вызываются редко и по делу.

Одновременно вырос интерес к agentic AI внутри самих игр: умные NPC, которые запоминают игрока, координируются друг с другом и адаптируются к стилю прохождения. Это уже не «боты против игроков», а часть геймдизайна.

2.6. Уроки истории

История показывает несколько устойчивых закономерностей:

- Простые методы — конечные автоматы, эвристики, template matching — остаются полезными даже в эпоху нейросетей. Их легко отлаживать, они дешёвы по вычислениям и предсказуемы.
- Полностью end-to-end решение часто оказывается дороже и менее надёжным в продакшене, чем хорошо спроектированный гибридный подход.
- Гонка между ботами и античитами никогда не заканчивается — меняются только технологии на обеих сторонах.
- Игры продолжают оставаться одним из лучших полигонов для развития общего искусственного интеллекта.

В следующих главах мы будем опираться на этот исторический контекст, выбирая методы, которые доказали свою практическую ценность, и избегая тупиковых путей.

ГЛАВА 3. Классификация ботов: типы и архитектуры

ПРОГРАММЫ ДЛЯ ПРОХОЖДЕНИЯ КОМПЬЮТЕРНЫХ ИГР

Боты • ИИ-агенты • Компьютерное зрение

Практическое руководство из 20 глав

2026

ОГЛАВЛЕНИЕ

Глава 1. Введение в автоматизацию компьютерных игр
Глава 2. История ботов и игрового ИИ
Глава 3. Классификация ботов: типы и архитектуры
Глава 4. Правовые, этические и социальные аспекты
Глава 5. Основы компьютерного зрения для игр
Глава 6. Захват экрана и предобработка изображений
Глава 7. Распознавание объектов: от Template Matching до YOLO
Глава 8. OCR и чтение игрового интерфейса
Глава 9. Симуляция ввода: мышь, клавиатура, геймпад
Глава 10. Простые боты на Python + OpenCV
Глава 11. Машинное обучение и нейронные сети в ботах
Глава 12. Reinforcement Learning для игровых агентов
Глава 13. Платформы и фреймворки (ViZDoom, ML-Agents, Gym)
Глава 14. ИИ-боты для FPS, ММО и стратегий
Глава 15. Античит-системы и методы детекции
Глава 16. Vision-Language Models и LLM-агенты в играх
Глава 17. Кейс-стади: реальные проекты и архитектуры
Глава 18. Инструменты, библиотеки и экосистема
Глава 19. Будущее ИИ-ботов в гейминге
Глава 20. Заключение и практические рекомендации

ГЛАВА 1. Введение в автоматизацию компьютерных игр

Автоматизация компьютерных игр — это область на стыке программирования, компьютерного зрения, машинного обучения и игровой разработки. Она включает создание программ (ботов), которые могут самостоятельно играть, выполнять рутинные задачи или помогать игроку.

Зачем нужна автоматизация?

- Обучение и исследования — игры служат отличной средой для тестирования алгоритмов ИИ (DeepMind AlphaGo, OpenAI Five, SIMA).
- Тестирование игр — автоматические агенты позволяют находить баги, проверять баланс и проводить регрессионное тестирование.
- Доступность — боты и ассистенты помогают игрокам с ограниченными возможностями.
- Развлечение и эксперименты — создание собственных ботов — популярный способ изучения компьютерного зрения и RL.

- Автоматизация рутины — в ММО и idle-играх боты снимают необходимость выполнять повторяющиеся действия (фарм, сбор ресурсов).

Основные подходы

Существует несколько фундаментальных подходов к созданию игровых ботов:

1. Скриптовые / макросы — запись и воспроизведение последовательностей действий (AutoHotkey, AutoIt, pyautogui).
2. Memory-based — чтение и изменение памяти процесса игры (требуется reverse engineering).
3. Vision-based — анализ экрана с помощью компьютерного зрения (OpenCV, YOLO, нейросети).
4. API / SDK-based — использование официальных или неофициальных интерфейсов игры (python-sc2 для StarCraft II, Mineflayer для Minecraft).
5. Reinforcement Learning / End-to-end агенты — обучение агента напрямую на пикселях или состоянии игры.

В этой книге мы сосредоточимся прежде всего на vision-based и RL-подходах, а также на современных мультимодальных моделях (Vision-Language Models), которые позволяют агентам «видеть» и «понимать» игру почти как человек.

ГЛАВА 2. История ботов и игрового ИИ

История игрового ИИ начинается задолго до современных нейросетей. Уже в 1950-х годах появлялись программы для игры в шахматы и ним (Nim, OXO).

Ключевые вехи

- 1951–1952 — Christopher Strachey (Nim), Alexander Douglas (OXO) — одни из первых игровых программ с ИИ.
- 1970–80-е — аркадные игры (Space Invaders, Pac-Man). Призраки в Pac-Man имели индивидуальные паттерны поведения на основе конечных автоматов (FSM).
- 1990-е — расцвет FPS-ботов (Quake, Unreal Tournament). Появились framework'и вроде Omni-bot. Бот-турниры и BotPrize (Turing Test для игровых ботов).
- 2000–2010-е — массовый фарм-боты в ММО (World of Warcraft, Lineage). Развитие античитов (Warden, BattlEye, Easy Anti-Cheat).
- 2013–2016 — DeepMind: DQN на Atari, AlphaGo. OpenAI Gym. Появление ViZDoom.
- 2018–2019 — OpenAI Five (Dota 2), AlphaStar (StarCraft II) — агенты, побеждающие профессиональных игроков.
- 2024–2026 — SIMA (DeepMind), Vision-Language Models, LLM-агенты, которые понимают скриншоты и выполняют инструкции на естественном языке. Появление коммерческих agentic систем в играх.

Эволюция шла от жёстко прописанных правил (rule-based) к обучению с подкреплением и, наконец, к мультимодальным foundation-моделям, способным обобщать знания между играми.

ГЛАВА 3. Классификация ботов: типы и архитектуры

По способу получения информации

- Memory bots — читают память процесса (адреса HP, координат, состояния). Быстрые и точные, но легко детектируются античитами и требуют reverse engineering.
- Packet / Network bots — перехватывают и подделывают сетевые пакеты. Используются в основном в MMO.
- Vision bots — работают исключительно с пикселями экрана. Самый «человекоподобный» и устойчивый к античитам подход (при правильной реализации).
- Hybrid — комбинация методов.

По уровню интеллекта

- Clicker / Macro — простые последовательности кликов и нажатий.
- State-machine bots — конечные автоматы: «если вижу врага — атакую, если HP низкое — убегаю».
- Classical CV bots — template matching, цветовая сегментация, контуры + эвристики.
- Deep Learning bots — YOLO/детекторы объектов + классификаторы + планировщики.
- RL agents — обучаются end-to-end (PPO, DQN, A3C и т.д.).
- LLM / VLM agents — используют большие языковые и мультимодальные модели для принятия решений на основе скриншотов и текстовых инструкций.

ГЛАВА 4. Правовые, этические и социальные аспекты

Использование ботов почти всегда нарушает пользовательское соглашение (ToS) большинства онлайн-игр. Это может привести к бану аккаунта, а в некоторых юрисдикциях — к юридической ответственности (особенно при продаже ботов или использовании для мошенничества).

Этические вопросы

- Нарушение честной игры и опыта других игроков.
- Влияние на экономику виртуальных миров (инфляция, обесценивание труда).
- Потенциал для создания «ферм» и эксплуатации (gold farming).
- Положительные стороны: помощь людям с инвалидностью, тестирование, исследования ИИ.

В этой книге материалы представлены исключительно в образовательных и исследовательских целях. Автор не поощряет нарушение правил игр и использование ботов в конкурентных онлайн-режимах.

ГЛАВА 5. Основы компьютерного зрения для игр

Компьютерное зрение (Computer Vision) — это ключевой инструмент vision-based ботов. Игра для такого бота — это просто поток изображений (кадров), из которых нужно извлекать полезную информацию: положение врагов, HP, миникарту, текст, кнопки интерфейса.

Основные задачи CV в контексте игр

- Object Detection — найти объекты определённых классов и их bounding box'ы (враги, NPC, предметы, кнопки).
- Classification — определить, что находится на экране (меню, бой, смерть, определённая локация).
- Segmentation — пиксельная маска объектов (полезно для точного позиционирования).
- OCR — чтение текста (уровень, золото, сообщения чата, названия предметов).
- Tracking — отслеживание объектов между кадрами (для предсказания движения).
- Template Matching — поиск заранее известных шаблонов изображений.

Классический пайплайн: захват кадра → предобработка → детекция/классификация → принятие решения → симуляция ввода → повтор.

ГЛАВА 6. Захват экрана и предобработка изображений

Качество и скорость захвата экрана критически важны. Медленный захват убивает производительность бота, особенно в быстрых играх (FPS).

Популярные способы захвата

- mss (Python) — один из самых быстрых кросс-платформенных вариантов.
- PIL / Pillow ImageGrab — простой, но медленнее.
- dxcam (Windows) — очень быстрый захват через Desktop Duplication API.
- OpenCV VideoCapture — можно захватывать окно или весь экран (медленнее).
- Нативные API (BitBlt, DXGI) — максимальная производительность, но сложнее в использовании.

Предобработка

Типичные операции: изменение размера (resize), перевод в оттенки серого, размытие (GaussianBlur), пороговая обработка (threshold), морфологические операции, выделение областей интереса (ROI), нормализация цветов, работа в разных цветовых пространствах (HSV, LAB).

Пример минимального захвата с mss:

```
import mss
import numpy as np
import cv2

with mss.mss() as sct:
    monitor = sct.monitors[1]
    img = np.array(sct.grab(monitor))
    frame = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
```

ГЛАВА 7. Распознавание объектов: от Template Matching до YOLO

Template Matching

Классический метод OpenCV (`cv2.matchTemplate`). Ищет точное или почти точное совпадение шаблона на изображении. Хорошо работает для статичных UI-элементов и объектов с неизменным внешним видом. Плохо — при изменении масштаба, поворота, освещения и динамических сценах.

Классические методы

Цветовая сегментация (HSV threshold), детекция контуров (`findContours`), каскады Хаара, HOG + SVM — всё ещё используются для простых задач и когда нужно максимальное быстродействие без GPU.

Deep Learning детекторы

- YOLO (v5–v12) — самый популярный выбор для игровых ботов. Реал-тайм, высокая точность, легко дообучается на своих данных.
- SSD, EfficientDet, RetinaNet — альтернативы.
- RT-DETR, RF-DETR и другие современные архитектуры.

Для обучения собственного детектора обычно собирают датасет скриншотов, размечают (LabelImg, CVAT, Roboflow), обучают модель (ultralytics YOLO) и экспортируют в ONNX / TensorRT для максимальной скорости инференса.

ГЛАВА 8. OCR и чтение игрового интерфейса

Многие решения бота зависят от числовых и текстовых данных: текущее HP, количество золота, таймеры, названия предметов, сообщения в чате.

Инструменты

- Tesseract OCR — классика, хорошо работает после правильной предобработки.
- EasyOCR, PaddleOCR — более современные и точные нейросетевые OCR.
- Специализированные модели — можно обучить маленький классификатор цифр под конкретный шрифт игры (часто надёжнее и быстрее общего OCR).

Практика: всегда вырезайте ROI с текстом, увеличивайте контраст, убирайте фон, при необходимости применяйте морфологию. Для цифр часто достаточно простого template matching по заранее вырезанным глифам.

ГЛАВА 9. Симуляция ввода: мышь, клавиатура, геймпад

После того как бот «увидел» ситуацию, ему нужно действовать. Симуляция ввода — критически важный и часто самый уязвимый для античита слой.

Библиотеки и методы

- pyautogui / pydirectinput — простые высокоуровневые обёртки.
- pynput — контроль и мониторинг ввода.
- Windows API (SendInput, mouse_event) — более низкий уровень.

- Виртуальные устройства / драйверы (Interception, HID) — выглядят для системы как настоящие устройства ввода.
- Arduino / hardware spoofing — физическая эмуляция мыши/клавиатуры.

Для «человечности» важно добавлять случайные задержки, кривые Безье для движения мыши, небольшие отклонения от идеальной точки прицеливания и вариативность в таймингах.

ГЛАВА 10. Простые боты на Python + OpenCV

Самый доступный стек для начинающих: Python + OpenCV + mss + pyautogui/pydirectinput. На нём можно собрать бота для казуальных, idle, puzzle и многих single-player игр.

Типичная архитектура простого бота

1. Захват кадра.
2. Определение текущего состояния (меню / бой / инвентарь) — по цвету, шаблону или маленькому классификатору.
3. Поиск целей (template matching или цвет).
4. Принятие решения (if-else или простой state machine).
5. Выполнение действия (клик, нажатие клавиши).
6. Небольшая пауза и повтор.

Такие боты отлично подходят для обучения: рыбалка в играх, сбор ресурсов, простые пазлы, автобой в turn-based играх. Многие open-source проекты (боты для Tetris, fishing bots, clicker-боты) построены именно по этой схеме.

ГЛАВА 11. Машинное обучение и нейронные сети в ботах

Когда эвристик и template matching становится недостаточно (динамика, разнообразие объектов, сложные сцены), на помощь приходит глубокое обучение.

Основные применения

- Детекция врагов и объектов (YOLO, RT-DETR).
- Классификация состояния экрана / фазы боя.
- Сегментация (для точного понимания сцены).
- Предсказание траекторий (для aimbot'ов и упреждения).
- Обучение политик поведения (имитационное обучение, behavioral cloning).

Behavioral Cloning — важный подход: записываете, как играете сами (скриншоты + действия), обучаете сеть предсказывать действия по изображению. Это позволяет создавать ботов, которые играют «в вашем стиле».

ГЛАВА 12. Reinforcement Learning для игровых агентов

Обучение с подкреплением (Reinforcement Learning) — это парадигма, в которой агент учится максимизировать награду, взаимодействуя со средой. Игры — идеальная среда для RL.

Ключевые алгоритмы

- DQN (и улучшения: Double DQN, Dueling, Rainbow) — value-based.
- PPO (Proximal Policy Optimization) — один из самых стабильных и популярных policy-gradient методов.
- A3C / A2C — асинхронные actor-critic.
- SAC, TD3 — для непрерывных пространств действий.
- AlphaZero / MuZero — планирование + обучение (для игр с perfect information).

Основные сложности: sample efficiency (нужно очень много кадров), reward shaping, частичная наблюдаемость (partial observability), исследование (exploration) и перенос знаний между задачами.

ГЛАВА 13. Платформы и фреймворки (ViZDoom, ML-Agents, Gym)

Для исследований и обучения RL-агентов существуют специализированные среды:

- OpenAI Gym / Gymnasium — стандартный интерфейс сред.
- ViZDoom — Doom как среда для RL с визуальными наблюдениями.
- Unity ML-Agents — мощный toolkit для обучения агентов внутри Unity.
- DeepMind Lab, Procgen, MineRL, NetHack Learning Environment — другие популярные бенчмарки.
- Stable-Baselines3, RLlib, CleanRL, Sample Factory — библиотеки алгоритмов RL.
- python-sc2, pysc2 — для StarCraft II.

Для реальных коммерческих игр чаще используют собственный захват экрана + свой обучающий пайплайн, либо imitation learning на записях игроков.

ГЛАВА 14. ИИ-боты для FPS, MMO и стратегий

FPS

Типичная архитектура: YOLO-детектор врагов → Kalman filter / трекер → расчёт упреждения → плавное наведение мыши → стрельба. Дополнительно — навигация (иногда отдельная RL-политика) и использование способностей. Примеры open-source: различные YOLO-aim проекты, BOTTI и подобные.

MMO / RPG

Больше внимания к state machine, распознаванию UI, pathfinding по миникарте, обработке инвентаря и квестов. Часто используют комбинацию CV + скриптов + (опционально) imitation learning. Примеры: боты для Genshin Impact, New World, различные МАА-подобные ассистенты.

Стратегии и RTS

Здесь сильны API-подходы (StarCraft II) и классические методы планирования + микроконтроль. AlphaStar показал, что RL способен достигать гроссмейстерского уровня.

ГЛАВА 15. Античит-системы и методы детекции

Современные античиты (BattlEye, Easy Anti-Cheat, Vanguard, VAC, Ricochet и др.) используют множество сигналов:

- Сигнатуры известных читов и инжектов.
- Поведенческий анализ (слишком идеальное прицеливание, неестественные траектории, реакции быстрее человеческих).
- Проверка целостности памяти и модулей.
- Драйверный и kernel-level мониторинг.
- Статистический анализ и machine learning на стороне сервера.

Vision-based боты без инжекта в процесс теоретически безопаснее memory-ботов, но всё равно могут быть выявлены по поведению. Поэтому для исследовательских целей рекомендуется работать только в offline / single-player / собственных средах.

ГЛАВА 16. Vision-Language Models и LLM-агенты в играх

С 2023–2026 годов произошёл качественный скачок: мультимодальные модели (GPT-4o, Claude, Gemini, LLaVA, Qwen-VL, DeepSeek-VL и др.) умеют принимать скриншот и отвечать на вопросы о происходящем, а также предлагать действия.

Архитектуры агентов

- Прямой VLM-агент — каждый кадр (или раз в N секунд) отправляется в модель вместе с инструкцией «что делать».
- Гибридная архитектура — дешёвые локальные методы (пиксели, OCR, state machine) обрабатывают 95–98 % ситуаций, а LLM вызывается только для сложных решений (стратегия, диалоги, неожиданные события).
- SIMA-подобные агенты — обучаются выполнять инструкции на естественном языке в разных 3D-мирах.

Такой подход значительно снижает стоимость (можно уложиться в центы в час) и повышает надёжность по сравнению с pure-LLM решениями.

ГЛАВА 17. Кейс-стади: реальные проекты и архитектуры

- YOLO + aim-системы — множество open-source проектов для FPS с real-time детекцией и наведением.
- MAA Assistant (Arknights) и аналогичные мобильные ассистенты — зрелые vision + script системы с огромным сообществом.
- Mineflayer — высокоуровневый API для Minecraft-ботов.
- BOT-MMORPG-AI и подобные — imitation learning + computer vision для MMO.
- Peek-a-bot — обучение RL-агентов только по зрению внутри Unreal Engine 5.
- DeepMind SIMA — generalist agent, способный следовать инструкциям в разных 3D-играх.
- GamingAgent / LMGame — агенты на базе LLM для grid-based и platformer игр.

ГЛАВА 18. Инструменты, библиотеки и экосистема

Компьютерное зрение и ML

OpenCV, ultralytics (YOLO), torchvision, timm, ONNX Runtime, TensorRT, OpenVINO, EasyOCR / PaddleOCR, supervision.

Захват и ввод

mss, dxcam, pyautogui, pydirectinput, pynput, keyboard, mouse.

RL и агенты

Gymnasium, Stable-Baselines3, CleanRL, RLlib, PettingZoo, Unity ML-Agents, ViZDoom.

Вспомогательные

NumPy, PyTorch / TensorFlow, Roboflow, Label Studio, Weights & Biases, Hydra / OmegaConf.

ГЛАВА 19. Будущее ИИ-ботов в гейминге

Тренды ближайших лет:

- Переход от узкоспециализированных ботов к generalist-агентам, способным играть в новые игры с минимальной адаптацией.
- Интеграция agentic AI прямо в игры разработчиками (умные NPC, динамические компаньоны, procedural storytelling).
- Улучшение sample efficiency и обучение в реальном времени.
- Более жёсткая гонка античитов против всё более человекоподобных агентов.
- Использование игровых агентов как тестовых стендов для embodied AI и робототехники.
- Этические и регуляторные рамки вокруг автономных агентов в онлайн-мирах.

ГЛАВА 20. Заключение и практические рекомендации

Создание игровых ботов и ИИ-агентов — одна из самых увлекательных и practical-областей применения компьютерного зрения и reinforcement learning. Она даёт быстрый feedback и наглядные результаты.

Рекомендации начинающим

1. Начните с простой single-player или offline игры и классического OpenCV + template matching.
2. Освойте захват экрана, ROI, базовую предобработку и симуляцию ввода.
3. Переходите к YOLO: соберите небольшой датасет, обучите детектор, измерьте FPS.
4. Изучите state machines и простую логику принятия решений.
5. Только после этого углубляйтесь в RL и мультимодальные модели.
6. Всегда работайте в этических рамках: offline, собственные проекты, research-only.

Игры остаются одной из лучших песочниц для развития технологий искусственного интеллекта. Понимая, как устроены боты и агенты, вы лучше понимаете и ограничения, и возможности современных систем ИИ.

Книга подготовлена в образовательных целях. Используйте знания ответственно.

ГЛАВА 3. Классификация ботов: типы и архитектуры

Чтобы ориентироваться в разнообразии решений, полезна чёткая таксономия. Ботов классифицируют по нескольким независимым осям.

3.1. По источнику информации о состоянии игры

Memory-based боты читают оперативную память процесса. Получают точные HP, координаты, ID объектов. Используют ReadProcessMemory и reverse engineering (Cheat Engine, сигнатуры). Плюсы — скорость и точность. Минусы — хрупкость после патчей, высокая детектируемость, этические риски.

Packet-based боты перехватывают трафик клиент–сервер. В современных играх трафик зашифрован и протоколы обфусцированы, поэтому метод усложнился.

Vision-based боты используют только изображение экрана. Это наиболее человекоподобный подход: не вмешивается в процесс, лучше переживает обновления (меняется внешний вид, а не внутренние структуры), сложнее детектируется по факту вмешательства. Требует больше вычислений на perception.

Hybrid комбинирует источники. Audio и multimodal дополняют vision редко, но полезны для событий по звуку.

3.2. По уровню интеллекта

Clicker/Macro — жёсткая последовательность без адаптации.

Rule-based и State-machine — «если HP < 30% — зелье», режимы поведения.

Classical CV — template matching, цвет, контуры + эвристики.

Deep Learning perception + rules — YOLO/классификатор понимает сцену, дальше логика.

Imitation Learning — сеть копирует действия человека; хорош стиль, слабое обобщение, compounding errors.

RL agents — обучение политики наградой; дорого и мощно.

LLM/VLM agents — модель рассуждает по скриншоту/описанию; универсально, пока дорого и медленно, почти всегда в гибриде.

3.3. По автономности

Полностью автономные; полуавтоматические ассистенты; оверлеи-подсказки; тестовые инструменты под CI.

3.4. Типичные архитектуры

«Детектор → правила»: кадр → YOLO → объекты → if-else/SM → ввод. Просто, надёжно, отлично для FPS aim и многих MMO-задач.

«Состояние экрана → сценарий»: сначала режим (меню/бой/инвентарь), затем обработчик. Удобно для UI-heavy игр.

Иерархические агенты: high-level выбирает задачу, low-level исполняет.

Модульные с символическим слоем: CV/OCR → описание мира → любой планировщик (вплоть до LLM).

3.5. Как выбирать

Стабилен ли визуал? Нужна ли реакция < 50–100 мс? Есть ли API? Критична ли детектируемость? Можно ли собрать датасет? Нужно ли обобщение или закрытый набор сценариев? Для большинства практических задач 2020-х оптимальна гибридная схема: быстрое локальное восприятие + лёгкая логика + редкие вызовы умных моделей.

ГЛАВА 4. Правовые, этические и социальные аспекты

Почти все онлайн-игры запрещают ботов в ToS. Нарушение ведёт к бану аккаунта, HWID-бану, иногда к претензиям при продаже ботов или RMT. Даже чистый vision-бот, как правило, запрещён правилами.

Законодательство зависит от страны. Написание программы анализа экрана само по себе обычно не преступление; распространение коммерческих ботов для обхода защиты и массовый RMT — уже серая или чёрная зона.

Этические проблемы: разрушение честной игры в PvP; инфляция виртуальных экономик; фарм-фермы и эксплуатация. Положительная сторона тех же технологий: ассистенты доступности, автоматизированное тестирование, научные исследования, обучение студентов CV и RL.

Принципы: использовать методы только в single-player, offline, собственных или research-средах; не мешать другим игрокам; не распространять готовые обходы античита; при публикации подчёркивать образовательный контекст. Техническая возможность не равна этическому праву.

ГЛАВА 5. Основы компьютерного зрения для игр

Компьютерное зрение — основной способ vision-бота понять экран. Преимущества: не нужно лезть в память; бот видит то же, что игрок; подходит для любой игры с GUI. Цена — освещение, перекрытия, разрешения, эффекты, антиалиасинг.

Задачи: object detection (враги, NPC, предметы, кнопки); classification (меню/бой/смерть); segmentation; tracking; OCR; template matching; иногда pose estimation.

Пайплайн: захват → предобработка → ROI → perception → постобработка → символическое состояние → решение.

Игровые кадры: высокая контрастность, плотный UI, частицы, вспышки, разные aspect ratio и DPI. Разрыв perception→decision закрывают слоем интерпретации и памятью о недавних событиях. Метрики: precision/recall, mAP, IoU, FPS, latency, устойчивость к смене условий.

Зрение избыточно при хорошем API, недостаточно при необходимости скрытой информации, проблематично при сверхжёсткой latency на слабом железе.

ГЛАВА 6. Захват экрана и предобработка изображений

Требования к захвату: высокий FPS (30–60+), низкая latency, захват окна/ROI, multi-monitor и DPI, малая нагрузка.

Windows: BitBlt (просто/медленно), Desktop Duplication API (быстро), mss (удобный кросс-платформенный), dxcam (высокая производительность), PIL (просто/медленно).

Пример:

```
import mss, numpy as np, cv2
with mss.mss() as sct:
    mon = sct.monitors[1]
    img = np.array(sct.grab(mon))
    frame = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
```

Окно: HWND → client rect с DPI → region. Exclusive fullscreen часто хуже windowed/borderless.

Предобработка: resize, BGR/RGB/HSV/gray, ROI, blur, CLAHE, threshold, morphology, inRange. Оптимизация: не захватывать лишнее; тяжёлую обработку не каждый кадр; GPU для сетей; профилировать. Проблемы: чёрный экран, смещение координат, низкий FPS — решаются выбором API и DPI-awareness.

Стартовый стек: Python, mss или dxcam, OpenCV, numpy.

ГЛАВА 7. Распознавание объектов: от Template Matching до YOLO

Template Matching (matchTemplate) — для статичного UI и иконок; плох при смене масштаба и сильных эффектах. Цвет + контуры (HSV, inRange, findContours) — быстрый без обучения. Haar/HOG устарели для большинства игровых задач.

YOLO (v5–v12) — практический стандарт: скорость, точность, ultralytics, экспорт в TensorRT/ONNX. Альтернативы: RT-DETR, EfficientDet.

Обучение: сбор кадров → разметка (LabelImg, CVAT, Roboflow) → аугментации → fine-tune → оценка на видео → экспорт → встройка. Постобработка: NMS, confidence, размер, трекинг (ByteTrack, Kalman), стабильные ID. Комбинируйте YOLO для динамики и template/цвет для UI. Ошибки: однообразные данные, игнор разрешений, плохие пороги, нет трекинга.

ГЛАВА 8. OCR и чтение игрового интерфейса

OCR нужен для HP, таймеров, золота, названий, квестов. Движки: Tesseract, EasyOCR, PaddleOCR; для цифр шрифта игры часто лучше свой классификатор или template глифов.

Предобработка: плотный ROI, upscale, gray, контраст, бинаризация, морфология. Числа: whitelist, резка цифр, длина полосы HP. Декоративные шрифты ломают общий OCR — нужен свой датасет. Вызывайте редко, кэшируйте, фильтруйте медианой, имейте fallback.

ГЛАВА 9. Симуляция ввода: мышь, клавиатура, геймпад

Уровни эмуляции: pyautogui/pydirectinput/pyinput → WinAPI → виртуальные HID/драйверы → Arduino. Ниже уровень — натуральнее для ОС, сложнее и рискованнее.

Мышь: абсолютные/относительные координаты, Безье и шум, ускорение, задержки, калибровка под sens и raw input. Клавиатура: press/release, удержание, тайминги. Геймпад: vgamepad.

Человечность: не идеальные прямые, не мгновенные 180°, не сверхреакция, не нулевые промахи. Аварийный стоп, логирование, абстракция InputController.

ГЛАВА 10. Простые боты на Python + OpenCV

Стек: Python + OpenCV + mss + pydirectinput. Каркас: capture → vision → decisions → input. Состояние экрана: хеши ROI, template UI, цвет, лёгкий классификатор.

Примеры: сбор ресурсов; рыбалка. State machine прозрачен и отлаживаем. Отладка: overlay боксов, логи, dry-run. Усложняйте до YOLO/RL/LLM, когда классика упирается в потолок.

ГЛАВА 11. Машинное обучение и нейронные сети в ботах

ML: детекция, классификация, сегментация, OCR, траектории, behavioral cloning, RL. Behavioral cloning прост в данных, страдает compounding errors.

Практика: YOLO nano/small, 300–2000 кадров, fine-tune, TensorRT. Данные важнее архитектуры. Инференс: лёгкие модели, FP16/INT8, отдельные потоки. Не решайте нейросетью то, что закрывается OpenCV.

ГЛАВА 12. Reinforcement Learning для игровых агентов

RL: state, action, reward, policy, value, episode. DQN-семейство; PPO/A2C/SAC; model-based (MuZero). PPO — практический стандарт.

Сложности: огромные пространства, sparse rewards, partial observability. Нужны reward shaping, curriculum, иерархия; опасен reward hacking. Среды: Gymnasium, ViZDoom, ML-Agents, rusec2. Старт с SB3 на простых средах. RL оправдан при огромном пространстве решений и отсутствии хороших демонстраций.

ГЛАВА 13. Платформы и фреймворки (ViZDoom, ML-Agents, Gym)

Gymnasium — единый API сред. ViZDoom — быстрый FPS-полигон. ML-Agents — Unity, curriculum, imitation, self-play. StarCraft (pysc2) — макро/микро, туман войны. Библиотеки: SB3, CleanRL, RLlib, Sample Factory. Путь: Gym → Atari → ViZDoom → ML-Agents → свои обёртки.

ГЛАВА 14. ИИ-боты для FPS, ММО и стратегий

FPS: YOLO → трекер → lead → сглаженное наведение → отдача; критична latency. ММО: UI, SM/BT, маршруты, инвентарь, recovery; perception = CV + OCR; high-level — LLM. МОБА — очень сложно. RTS — API предпочтительнее vision. Sandbox — иерархия и LLM. Принципы: частота решений, разделение слоёв, recovery, телеметрия, перехват человеком.

ГЛАВА 15. Античит-системы и методы детекции

Античиты: клиентские (BE, EAC, Vanguard), серверные (статистика, ML), гибриды. Детектируют сигнатуры, память, DLL, хуки, невозможные действия, аномалии поведения, injected input. Vision без инжекта невидим для memory-проверок, уязвим к behavioral. Исследователям — только offline/research.

ГЛАВА 16. Vision-Language Models и LLM-агенты в играх

VLM: изображение + текст → текст (описание, план, команды). Прямой агент универсален, но дорог. Гибрид: локально 95–98 %, LLM при застревании и планировании. Память, инструменты, ReAct. SIMA — generalist по инструкциям. Жёсткий формат ответа, лимит частоты, логи, YOLO для координат.

ГЛАВА 17. Кейс-стади: реальные проекты и архитектуры

Кейсы: OpenCV fishing; YOLO-aim; МАА-ассистенты; Mineflayer (API-first); Peek-a-bot (RL in UE5); SIMA; гибридный LLM-RPG. Уроки: узкое — надёжно; общее — дорого; гибриды выигрывают; важна поддержка после патчей.

ГЛАВА 18. Инструменты, библиотеки и экосистема

Стек: Python; OpenCV, ultralytics, ONNX/TensorRT, EasyOCR; mss/dxcam, pydirectinput; PyTorch, Gymnasium, SB3; Roboflow, CVAT. Минимум: mss+OpenCV+pydirectinput+ultralytics. Для продакшена — TensorRT, трекер, конфиги, логи.

ГЛАВА 19. Будущее ИИ-ботов в гейминге

Тренды: generalist-агенты; agentic AI в геймдизайне; sample efficiency; связь с робототехникой; ужесточение античитов; этика и регуляция. Практика: крепкий perception, мультимодальность, модульность, этика.

ГЛАВА 20. Заключение и практические рекомендации

Итоги: vision универсален; гибриды практичнее end-to-end; данные и инженерия важнее хайпа; этика обязательна. Дорожная карта: OpenCV → SM → YOLO → Gym/PPO → LLM-

гибриды. Не начинайте с онлайн-шутера; логируйте; работайте offline. Игры — лучшая песочница для ИИ при ответственном подходе.

ПРИЛОЖЕНИЕ А. Примеры кода и шаблоны проектирования

А.1. Каркас главного цикла бота

```
while running:
    frame = capture.get_frame()
    if frame is None:
        continue
    state = vision.analyze(frame)
    actions = decisions.decide(state)
    input_ctrl.execute(actions)
    logger.tick(state, actions)
    if emergency_stop_pressed():
        break
```

А.2. Рекомендации по структуре проекта

```
bot/
main.py
capture.py
vision/
detector.py
ocr.py
state.py
decisions/
state_machine.py
policies.py
input_controller.py
config.yaml
templates/
weights/
logs/
```

А.3. Чек-лист перед запуском на реальной игре

- Dry-run vision без ввода
- Совпадение координат клика с картинкой
- Аварийный стоп работает
- Окно игры в ожидаемом режиме (windowed/borderless)
- Логи пишутся
- Нет работы на свёрнутом окне без явного флага

ПРИЛОЖЕНИЕ В. Глоссарий ключевых терминов

Agent, Behavioral cloning, Bounding box, Curriculum learning, DPI awareness, End-to-end, Gymnasium, IoU, Latency, mAP, NMS, OCR, Partial observability,

Policy, Reward shaping, ROI, State machine, Template matching, VLM, YOLO, PPO, DQN, TensorRT, ONNX, ByteTrack, Kalman filter, ReAct, SIMA.

ПРИЛОЖЕНИЕ С. Рекомендуемые ресурсы

Документация: OpenCV, ultralytics YOLO, Gymnasium, Stable-Baselines3, ViZDoom, Unity ML-Agents, mss, dxcam.

Книги: Sutton & Barto — Reinforcement Learning: An Introduction.

Статьи: DQN, AlphaGo, AlphaStar, OpenAI Five, SIMA (DeepMind).

Принцип: изучать открытый код критически и применять только в этических рамках.

КОНЕЦ КНИГИ

Книга подготовлена в образовательных и исследовательских целях.
Используйте знания ответственно. Не применяйте описанные методы для нарушения правил онлайн-игр и создания нечестного преимущества.

РАСШИРЕННЫЕ ПРАКТИЧЕСКИЕ МАТЕРИАЛЫ И УГЛУБЛЁННЫЕ РАЗДЕЛЫ

Углубление: проектирование state machine

Конечный автомат остаётся одним из самых надёжных способов организации поведения бота. Состояния должны быть взаимоисключающими и покрывать все разумные режимы: инициализация, ожидание, поиск цели, сближение, взаимодействие, бой, отступление, восстановление, обработка смерти, возврат к маршруту, авария.

Переходы задаются предикатами над состоянием мира (то, что вернул perception). Избегайте «скрытого состояния» в глобальных переменных — всё, от чего зависит переход, должно быть явно в state object. Для отладки логируйте каждую смену состояния с timestamp и причиной перехода. При росте сложности рассмотрите hierarchical state machines или behaviour trees.

Углубление: сбор и разметка датасета для YOLO

Качество детекции на 80 % определяется данными. Снимайте геймплей в разных локациях, при разном освещении и погоде, с разными эффектами. Включайте частично перекрытые объекты, крайние ракурсы, моменты вспышек. Размечайте аккуратно: бокс должен плотно охватывать объект, класс — быть согласованным. Используйте active learning: прогоняйте текущую модель по новым видео и доразмечайте кадры с низкой уверенностью или ошибками. Балансируйте классы. Документируйте версию игры и настроек графики, на которых собран датасет.

Углубление: снижение latency пайплайна

В динамичных жанрах задержка от появления врага до движения мыши критична. Профилируйте каждый этап: захват, сору на GPU, инференс, трекинг, расчёт движения, отправка ввода. Часто узкое место — не модель, а лишние конвертации цветов или захват всего стола вместо окна. Используйте FP16/INT8, TensorRT, фиксированный input size, асинхронность. Детекцию можно делать не на каждом кадре, опираясь на трекер между запусками.

Углубление: reward shaping без reward hacking

Промежуточные награды ускоряют обучение, но агент начинает оптимизировать прокси. Если награждать за урон, агент может «фармить» слабых врагов вместо цели эпизода. Если штрафовать смерть слишком сильно — станет трусливым. Проверяйте выученную политику визуально и по вторичным метрикам (успех задачи, время, разнообразие поведения). Curriculum и иерархия часто безопаснее сложных ручных формул награды.

Углубление: гибрид LLM + классический CV

Паттерн: perception всегда локальный и быстрый. LLM получает структурированное описание (список объектов, OCR-поля, текущее SM-состояние, последние события) плюс редкий скриншот. Ответ — строгий JSON с командой или планом. Парсер отвергает любой неразбираемый вывод. LLM вызывается по триггерам: застревание, новый квест, неизвестный UI, конец эпизода. Так сохраняются скорость, стоимость и предсказуемость.

Глубокий разбор: vision-aim в FPS

Полный конвейер: захват с минимальной latency → YOLO (nano/small + TensorRT) → NMS → ассоциация с треками → оценка скорости цели (Kalman или конечно-разностная) → расчёт точки упреждения с учётом скорости пули и расстояния → перевод в дельту мыши через чувствительность и raw input → движение по сглаженной траектории → выстрел с учётом разброса и режима огня → компенсация отдачи по пресету оружия. Каждый блок тестируйте отдельно. Человечность: распределение времени реакции, шум траектории, редкие «промахи», запрет на мгновенные развороты. Никогда не используйте такой пайплайн в рейтинговых онлайн-режимах.

Глубокий разбор: фарм-бот MMO на vision

Модули: навигация по точкам/миникарте; бой (агро, ротация способностей, зелья); лут; инвентарь и продажа; квестовые маркеры; обработка смерти и возврат; анти-AFK (редкие осмысленные действия). Perception: детектор мобов/NPC, цвет/template для маркеров, OCR

для HP и количества предметов. SM или behaviour tree связывает модули. Логируйте причины простоев — обычно это ложные срабатывания UI или застревание в геометрии.

Глубокий разбор: обучение PPO в ViZDoom

Обёртка среды в Gymnasium, выбор observation (пиксели vs game variables), frame skip, reward shaping под сценарий (навигация, сражение). Stable-Baselines3 PPO: n_steps, batch_size, learning_rate, ent_coef, gamma. Смотрите кривые reward и episode length. Усложняйте сценарий curriculum-ом. Перенос в реальный шутер прямым копированием весов почти никогда не работает; переносятся идеи архитектуры и shaping.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: С чего начать, если я умею только базовый Python?

О: Установите OpenCV и mss. Сделайте скрипт, который показывает FPS захвата. Затем template matching по кнопке в любой single-player игре и клик через pydirectinput. Это уже бот.

В: Нужна ли видеокарта?

О: Для классического OpenCV — нет. Для YOLO в реальном времени желательна NVIDIA с CUDA/TensorRT. Для обучения детектора GPU сильно ускоряет процесс.

В: Можно ли полностью обойтись без нейросетей?

О: Да, для многих задач UI, цветов и стабильных объектов. Нейросети нужны при сильной вариативности внешнего вида и сложных сценах.

В: Как не получить бан?

О: Не запускайте ботов в онлайн-играх с античитом и ToS-запретом. Работайте в single-player, offline, собственных или research-средах.

В: Чем LLM лучше state machine?

О: Гибкостью в нестандартных ситуациях и способностью понимать текст/контекст. Хуже скоростью, стоимостью и предсказуемостью. Используйте вместе.

В: Сколько данных нужно для YOLO под одну игру?

О: Часто достаточно 300–1500 хорошо размеченных кадров при сильном предобучении и аугментациях, если домен узкий. Для сложных сцен — больше.

В: Что читать дальше?

О: Документацию ultralytics и Gymnasium; Sutton & Barto; блоги и статьи DeepMind по игровым агентам; исходники аккуратных open-source ассистентов.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.

- Задержка реакции «человечная»: не ниже $\sim 0.15\text{--}0.2$ с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже $\sim 0.15\text{--}0.2$ с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже $\sim 0.15\text{--}0.2$ с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже ~0.15–0.2 с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже ~0.15–0.2 с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже ~0.15–0.2 с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.
- Задержка реакции «человечная»: не ниже ~0.15–0.2 с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.
- Confidence YOLO: начинайте с 0.35–0.5 и подстраивайте.
- IoU NMS: 0.45–0.6 типичный диапазон.

- Задержка реакции «человечная»: не ниже $\sim 0.15\text{--}0.2$ с на визуальный стимул в среднем.
- Размер ROI: минимально достаточный; меньше ROI — быстрее и меньше false positives.
- Логи храните хотя бы несколько часов работы для разбора сбоев.
- Версионите веса моделей и конфиги вместе.
- Перед обновлением игры сохраняйте старые шаблоны и датасеты.
- Тестируйте на том же разрешении и настройках графики, на которых собирали данные.
- Имейте profile на игру: пути, ROI, hotkeys, sens, пороги.

Дополнение: при работе с несколькими мониторами всегда явно указывайте monitor index и проверяйте координаты клика на целевом дисплее.

Дополнение: для отладки OCR сохраняйте предобработанные ROI на диск — так легче понять, почему движок ошибся.

Дополнение: при imitation learning перемешивайте демонстрации разных сессий и игроков, если стиль должен быть общим, а не копией одного человека.

Справочный блок: типичные параметры и рекомендации

- Целевой FPS perception в спокойных играх: 10–20; в FPS: 30–60+.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.