**Оглавление**

1. [Обучение с Подкреплением…………………….………………………….3](#obucheniyespodkrepleniem)
   1. [Введение в Обучение с Подкреплением……………………………3](#vvedenievobucheniye)
   2. [Терминологии………………………………………………………..6](#Terminologies)
   3. [Алгоритмы Обучения с Подкреплением…………………………...8](#Algorithmsobucheniya)
2. [В Воздушном Хоккее…………………………………………...………...10](#vvozdushnemhockey)
3. [Q-Learning...…….…………………………………………………..13](#QLearningdocs)
4. [Upper Confidence Bound(UCB)…………………………………….17](#UpperConfidenceBounddocs)
5. [Визуализация, Система Вознаграждения и как QLearn работает..21](#visualizationandsystem)
6. [Заключение………………………………………………………………..25](#Zaklucheniye)
7. [Приложение……………………………………………………………….26](#prilojeniye)
8. [Список Литературы……………………………………………………….27](#spisokliteraturi)

**Обучение с Подкреплением**

**Введение в Обучение с Подкреплением**

Обучение с подкреплением (RL) - это наука о принятии решений. Речь идет об обучении оптимальному поведению в окружающей среде для получения максимального вознаграждения. Этому оптимальному поведению обучаются благодаря взаимодействию с окружающей средой и наблюдениям за тем, как она реагирует, подобно тому, как дети исследуют окружающий мир и учатся действиям, которые помогают им достичь цели.

В отсутствие руководителя учащийся должен самостоятельно определить последовательность действий, которые максимизируют вознаграждение. Этот процесс обнаружения сродни поиску методом проб и ошибок. Качество действий измеряется не только немедленным вознаграждением, которое они приносят, но и отсроченным вознаграждением, которое они могут получить. Поскольку он может обучаться действиям, которые приводят к конечному успеху в невидимой среде, без помощи руководителя, обучение с подкреплением является очень мощным алгоритмом.

Примеры обучения с подкреплением:

* Робототехника: Роботы с запрограммированным поведением полезны в структурированных средах, таких как сборочная линия автомобильного завода, где задача повторяется по своей природе. В реальном мире, где реакция окружающей среды на поведение робота неопределенна, предварительное программирование точных действий практически невозможно. В таких сценариях RL предоставляет эффективный способ создания роботов общего назначения. Это было успешно применено для планирования траектории движения робота, когда робот должен найти короткий, плавный и проходимый путь между двумя точками, без столкновений и совместимый с динамикой робота;
* AlphaGo: Одной из самых сложных стратегических игр является китайская настольная игра Го, которой 3000 лет. Ее сложность обусловлена тем фактом, что существует 10270 возможных комбинаций на доске, что на несколько порядков больше, чем в шахматах. В 2016 году агент по Го из RL под названием AlphaGo победил величайшего игрока-человека в Го. Как и любой игрок-человек, он учился на собственном опыте, сыграв тысячи игр с профессиональными игроками. Новейший Go-агент на базе RL обладает способностью учиться, играя против самого себя, - преимуществом, которого нет у игрока-человека;
* Автономное вождение: Система автономного вождения должна выполнять множество задач восприятия и планирования в условиях неопределенности. Некоторые конкретные задачи, в которых RL находит применение, включают планирование траектории движения транспортного средства и прогнозирование движения. Планирование траектории движения транспортного средства требует нескольких политик низкого и высокого уровня для принятия решений в различных временных и пространственных масштабах. Прогнозирование движения - это задача прогнозирования движения пешеходов и других транспортных средств, чтобы понять, как может развиваться ситуация, исходя из текущего состояния окружающей среды;

Основными элементами системы RL являются:

1. Агент или обучаемый;
2. Среда, с которой взаимодействует агент;
3. Политика, которой агент следует при выполнении действий;
4. Сигнал вознаграждения, который агент наблюдает при выполнении действий;



Рисунок 1: Агент, Среда, Политика и Сигнал вознаграждения [[1](#synopsys)]

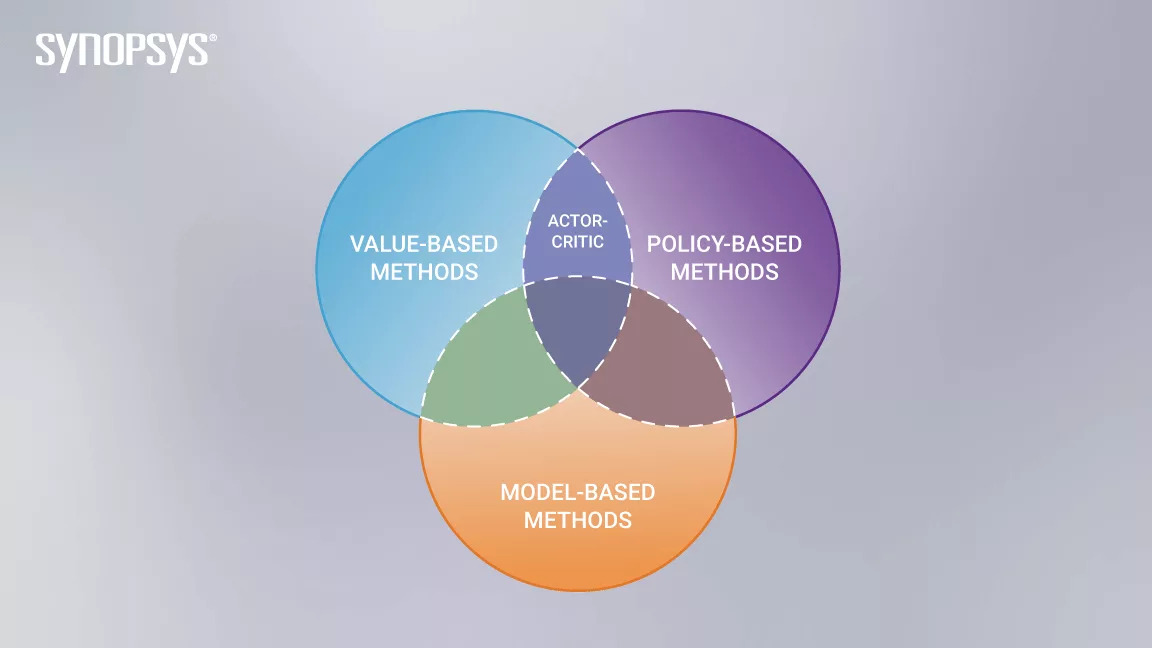


Рисунок 2: Value-based, Model-based и Policy-based[[2](#synopsys)]

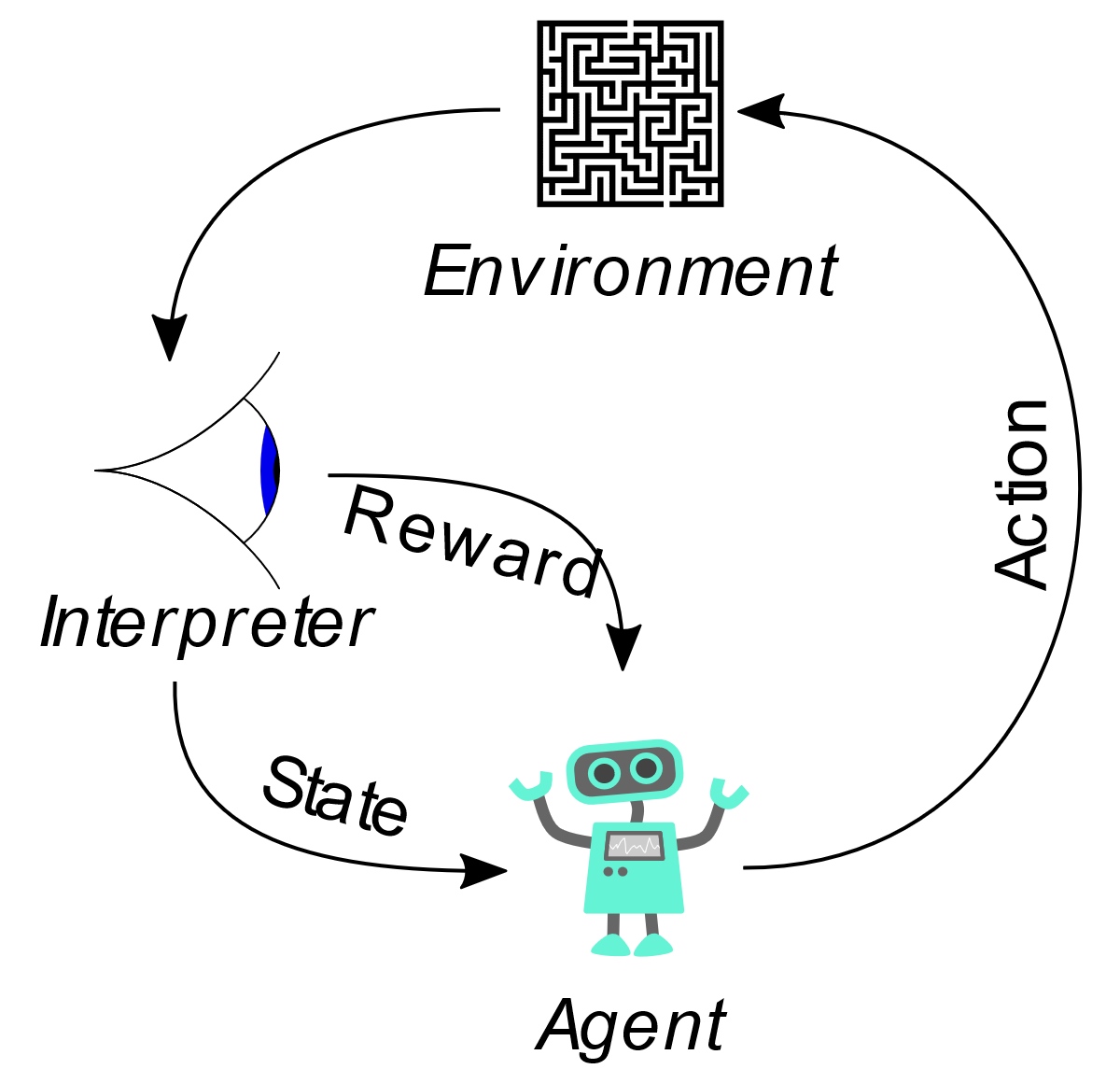


Рисунок 3: Типичная структура обучения с подкреплением [[3](#reinforcementlearningwiki)]

**Терминологии**

* Agent: является единственным лицом, принимающим решения и обучающимся;
* Environment: физический мир, в котором агент учится и принимает решение о действиях, которые необходимо выполнить;
* Action: список действий, которые может выполнить агент;
* State: текущее положение агента в окружающей среде;
* Reward: За каждое выбранное агентом действие среда выдает вознаграждение. Обычно это скалярное значение и ничего, кроме обратной связи от среды;
* Policy: агент разрабатывает стратегию (принятие решений) для сопоставления ситуаций с действиями;
* Value Function: Значение состояния показывает достигнутое вознаграждение, начиная с состояния и до тех пор, пока политика не будет выполнена;
* Model: Каждый агент RL не использует модель своей среды. Представление агента отображает распределения вероятностей пар состояние-действие по состояниям;

Базовое обучение с подкреплением моделируется как марковский процесс принятия решений:

* набор состояний среды и агента, *S*;
* набор действий, *A*, агента;
* *Pa(s, s') = Pr(St+1 = s' | St = s, At = a)*, вероятность перехода (в момент времени *t*) из состояния *s* в состояние *s'* при действии *a*;
* *Ra(s,s')*, немедленное вознаграждение после перехода от *s* к *s'* с действием *a*;

Базовый агент обучения с подкреплением AI взаимодействует со своей средой с дискретными временными шагами. В каждый момент времени *t* агент получает текущее состояние *St* и вознаграждение *Rt*. Затем он выбирает действие *At* из набора доступных действий, которое впоследствии отправляется в среду. Среда переходит в новое состояние *St+1*, и определяется вознаграждение *Rt+1*, связанное с переходом *(St,At,St+1)*. Цель обучающего агента с подкреплением состоит в том, чтобы изучить политику: *π: S × A [0,1]*, *π(s, a) = Pr(At = a, St = s),* которая максимизирует ожидаемое совокупное вознаграждение.

**Алгоритмы Обучения с Подкреплением**

Существует 3 подхода к реализации алгоритмов обучения с подкреплением

* Value-based: Основная цель этого метода - максимизировать функцию значения. Здесь агент с помощью политики ожидает долгосрочного возврата текущих состояний.
* Policy-based: В режиме, основанном на политике, вы можете разработать стратегию, которая поможет получить максимальное вознаграждение в будущем за счет возможных действий, выполняемых в каждом штате. Два типа методов, основанных на политике, - детерминированные и стохастические.
* Model-based: В этом методе нам нужно создать виртуальную модель для агента, которая поможет ему научиться работать в каждой конкретной среде.

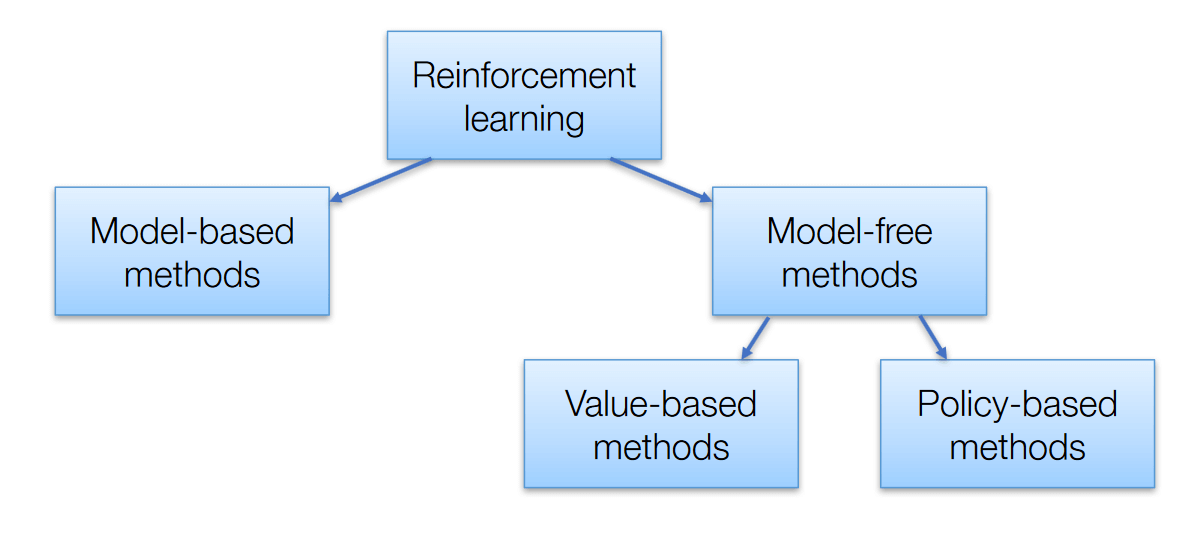


Рисунок 4: Алгоритмы Обучения с Подкреплением[[4](#WhatIsRLandHowDoesItWork)]

Model-based Подход:

При подходе к обучению с подкреплением, основанном на моделях, акцент смещается с непосредственного изучения действий или ценностных функций на понимание и построение модели окружающей среды. Этот подход включает в себя два ключевых этапа: изучение модели и планирование.

* Model Learning: Здесь агент пытается изучить представление о динамике окружающей среды. По сути, он пытается предсказать следующее состояние и потенциальную награду за свои действия. Это предсказание может быть основано на исторических данных или выведено путем взаимодействия с окружающей средой. Модель может быть любой - от простого линейного предиктора до сложной нейронной сети, в зависимости от сложности окружающей среды;
* Planning: Как только у агента есть модель, он использует алгоритмы планирования, чтобы выбрать наилучший курс действий. Планирование включает в себя моделирование различных путей действий в модели для прогнозирования их результатов и выбора пути, который ведет к максимальному совокупному вознаграждению. Это может включать такие методы, как алгоритмы древовидного поиска или даже более простые эвристические методы;

Сила подхода, основанного на моделях, заключается в его эффективности. При построении модели агенту требуется меньше взаимодействий с реальной средой для эффективного обучения, что выгодно в сценариях, где взаимодействия являются дорогостоящими или опасными. Более того, способность моделировать будущие состояния позволяет принимать более обоснованные и потенциально безопасные решения.

Области применения и соображения:

RL на основе моделей особенно эффективен в контролируемых средах, где можно создавать точные модели, например, в робототехнике или игровых сценариях. Однако одной из проблем является точность модели – неточная модель может привести к неоптимальным или даже вредным решениям. Таким образом, сложность окружающей среды и возможность построения надежной модели являются критическими соображениями при выборе этого подхода.

**В Воздушном Хоккее**

Задача управления роботом-противником имеет игровую структуру: ожидается, что агент самостоятельно выберет действия для достижения определенной цели. Поэтому мы рассмотрели эту проблему в контексте обучения с подкреплением, когда агента обучают совершать действия в неизвестной среде, руководствуясь системой вознаграждения, которая вознаграждает или наказывает агента за совершение определенного действия в определенном игровом состоянии.

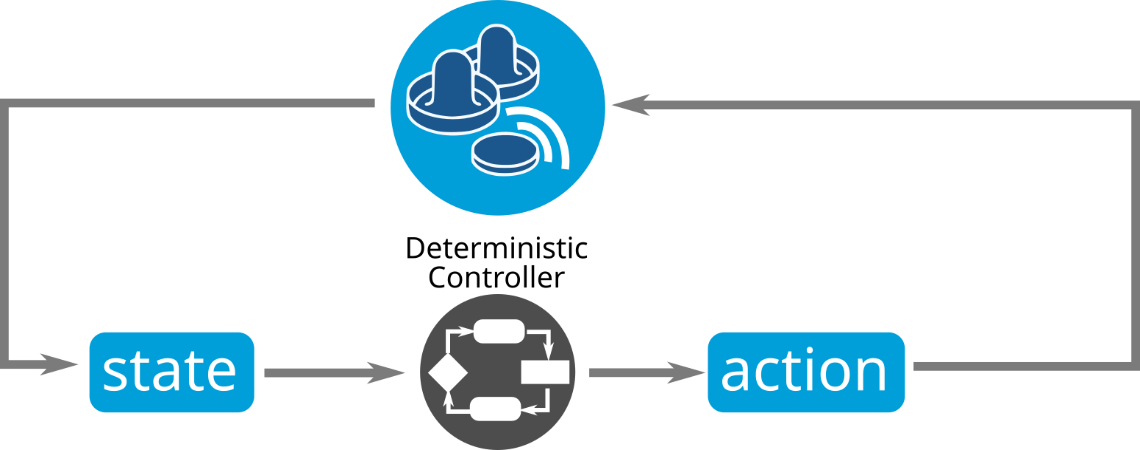


Рисунок 5: Контур управления[[5](#AIplayinAirHockey)]

Учитывая динамику и требования аэрохоккея, рекомендуется внедрять Q-Learning, усиленное методом верхней доверительной границы и временной разницы.

1. Q-Learn:
   * Определение: Q-Learning - это основанный на ценности алгоритм обучения с подкреплением. Он используется для поиска оптимальной политики выбора действий в данной среде. Q-Learning работает путем изучения функции (Q-function), которая представляет ожидаемую полезность выполнения данного действия в данном состоянии и последующего следования оптимальной политике;
   * Применение: С практической точки зрения Q-Learning может позволить агенту искусственного интеллекта, такому как робот для аэрохоккея, изучать наилучшие ходы и стратегии, постоянно обновляя значения *Q* на основе результатов вознаграждения за его действия;
2. Upper Confidence Bound(UCB):
   * **Определение**: Алгоритм UCB - это метод балансирования исследования и эксплуатации при обучении с подкреплением. Он помогает в принятии решений, принимая во внимание как среднее вознаграждение, получаемое за действие, так и неопределенность или дисперсию, связанные с этим действием;
   * Применение: В аэрохоккее UCB может помочь агенту искусственного интеллекта решить, стоит ли пробовать новые стратегии (исследование) или усовершенствовать стратегии, которые хорошо работали в прошлом (эксплуатация), тем самым повышая эффективность его обучения;
3. Temporal Difference Method(Метод временных разностей):
   * Определение: Методы временных разностей представляют собой группу алгоритмов обучения с подкреплением, которые обучаются непосредственно на основе сырого опыта, не используя модель динамики окружающей среды. Они обновляют оценки, основываясь частично на других наученных оценках, не дожидаясь окончательного результата;
   * Применение: В аэрохоккее TDM позволяет ИИ агенту учиться оптимальным стратегиям из каждой игры, постепенно улучшая свои навыки, обновляя свою политику после каждой игры или даже во время текущих игр;

Эта комбинация дает несколько преимуществ:

- Сбалансированное обучение: Интегрируя Q-Learning с UCB, агент искусственного интеллекта может эффективно балансировать между изучением новых игровых стратегий и использованием известных успешных тактик, что приводит к более надежному и адаптивному игровому процессу.

- Эффективное обновление стратегии: Использование метода временных различий в Q-Learning позволяет ИИ аэрохоккея обновлять свои стратегии более оперативно и поэтапно. Это особенно полезно в быстро меняющейся среде, такой как аэрохоккей, где адаптивность имеет решающее значение.

- Оптимальное принятие решений: Сочетание этих методов помогает принимать более обоснованные и оптимальные решения во время игрового процесса, поскольку учитывает как краткосрочные, так и долгосрочные результаты действий.

- Снижение потребности в предварительном обучении: Такой подход снижает потребность в обширном предварительном обучении, поскольку искусственный интеллект может обучаться и адаптироваться на ходу, что делает его более практичным и эффективным для реальных приложений.

Использование Q-Learning с UCB и TDM является эффективной стратегией для разработки конкурентоспособного агента искусственного интеллекта в аэрохоккее. Такой подход не только улучшает процесс обучения, но и обеспечивает высокий уровень адаптивности и стратегическую глубину игрового процесса.

**Q-Learning**

Q-learning - это безмодельный алгоритм обучения с подкреплением, позволяющий узнать ценность действия в определенном состоянии. Для него не требуется модель окружающей среды (отсюда и "безмодельный"), и он может решать проблемы со стохастическими переходами и вознаграждениями, не требуя адаптации. Для любого конечного марковского процесса принятия решений Q-learning находит оптимальную политику в смысле максимизации ожидаемого значения общего вознаграждения на любых последовательных этапах, начиная с текущего состояния. Q-обучение может определить оптимальную политику выбора действий для любого заданного конечного марковского процесса принятия решений, учитывая бесконечное время исследования и частично случайную политику. "*Q*" относится к функции, которую вычисляет алгоритм – ожидаемое вознаграждение за действие, предпринятое в данном состоянии.

После того, как *Δt* переместится в будущее, агент примет решение о каком-либо следующем шаге. Вес для этого шага рассчитывается как ***γ***Δt, где гамма (коэффициент дисконтирования) - это число от 0 до 1 (0 ≤ ***γ***  ≤1). Предполагая, что ***γ***  < 1, это приводит к тому, что награды, полученные ранее, оцениваются выше, чем те, которые получены позже (отражая ценность "хорошего начала"). гамма также может интерпретироваться как вероятность успеха (или выживания) на каждом шаге *Δt*.

Таким образом, алгоритм имеет функцию, которая вычисляет качество комбинации состояния и действия:

*Q: S x A → ℝ*

Перед началом обучения *Q* инициализируется, возможно, произвольным фиксированным значением (выбранным программистом). Затем, в каждый момент времени t агент выбирает действие *At*, наблюдает вознаграждение *Rt+1*, переходит в новое состояние *St+1* (которое может зависеть как от предыдущего состояния *St*, так и от выбранного действия), и *Q* обновляется. Ядром алгоритма является уравнение Беллмана в виде простого итерационного обновления значения, использующего средневзвешенное значение текущего значения и новую информацию:

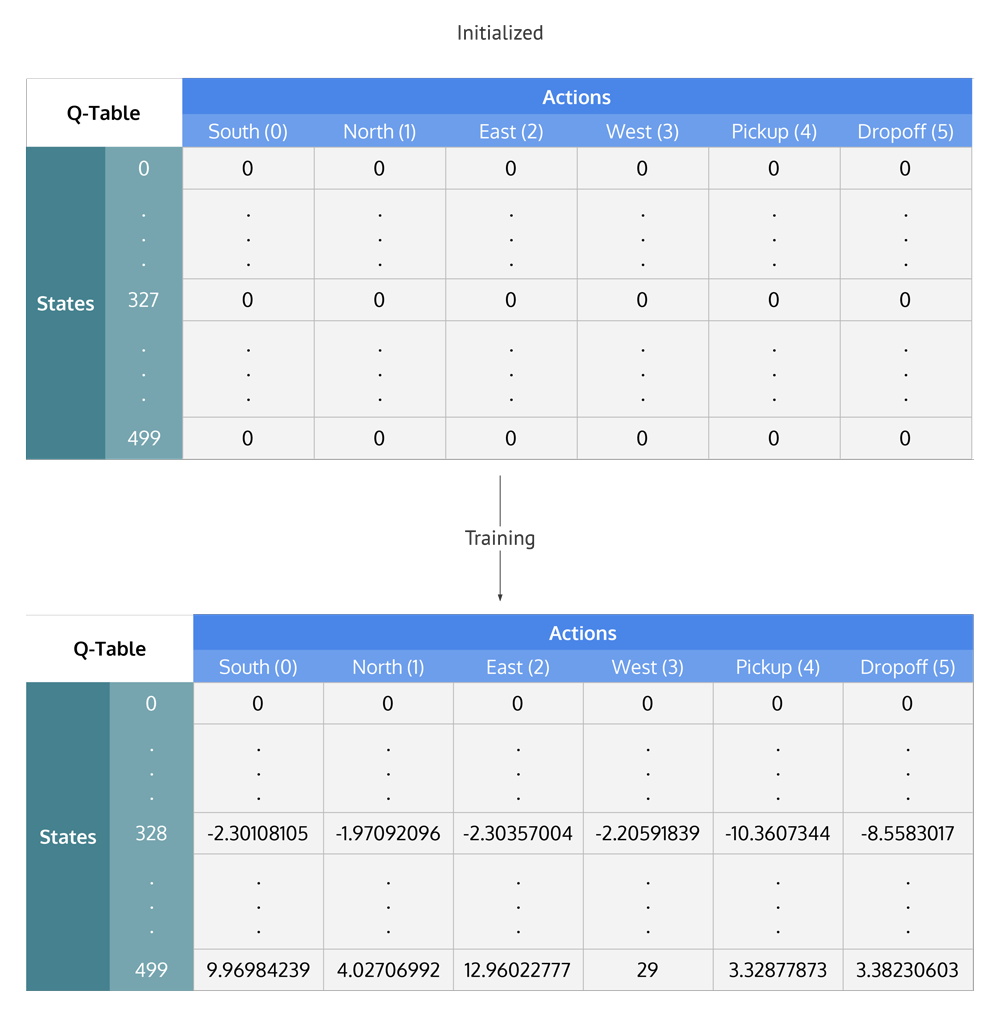


Рисунок 6: Q-Learning таблица состояний по действиям, которая инициализируется нулем, затем каждая ячейка обновляется посредством обучения. [[6](#QLearningWiki)]

Перед началом обучения *Q* инициализируется, возможно, произвольным фиксированным значением (выбранным программистом). Затем, в каждый момент времени t агент выбирает действие *At*, наблюдает вознаграждение *Rt+1*, переходит в новое состояние *St+1* (которое может зависеть как от предыдущего состояния *St*, так и от выбранного действия), и *Q* обновляется. Ядром алгоритма является уравнение Беллмана в виде простого итерационного обновления значения, использующего средневзвешенное значение текущего значения и новую информацию:

*Qnew (St, At) ← (1 - α) ⋅ Q(St, At) + α ⋅ (Rt+1 + γ ⋅ maxQ(St+1, a))*

*α*

*α* - скорость обучения

*Q(St, At)* - текущее значение

*Rt+1* – вознаграждение

*γ* - коэффициент дисконтирования

*maxQ(St+1, a))* - оценка оптимальной будущей стоимости

*α*

*(Rt+1 + γ ⋅ maxQ(St+1, a))* - новое значение (целевое значение временной разницы)

где *Rt+1* - вознаграждение, полученное при переходе из состояния St в состояние *St+1*, а *α* - скорость обучения (0 < *α* < 1). Обратите, что *Qnew (St, At)* является суммой трех факторов:

* *(1 – α) ⋅ Q(St, At)* - текущее значение (взвешенное на единицу минус скорость обучения);
* *αRt+1*: вознаграждение *Rt+1*, которое можно получить, если выполнить действие *At*, находясь в состоянии *St* (взвешенное по скорости обучения);
* *αγmaxQ(St+1, a)*: максимальное вознаграждение, которое можно получить из состояния *St+1* (взвешенное по скорости обучения и коэффициенту дисконтирования);

Эпизод алгоритма заканчивается, когда состояние *St+1* является конечным. Однако Q-learning может также обучаться в неэпизодических задачах (в результате свойства сходящихся бесконечных рядов). Если коэффициент дисконтирования меньше 1, значения действия конечны, даже если задача может содержать бесконечные циклы.

Для всех конечных состояний *sf Q(sf,a)* никогда не обновляется, но устанавливается на значение вознаграждения r, наблюдаемое для состояния *sf*. В большинстве случаев *Q(sf,a)* можно принять равным нулю.

*α*(скорость обучения) : Скорость обучения или размер шага определяет, в какой степени новая полученная информация переопределяет старую. Коэффициент, равный 0, заставляет агента ничему не учиться (исключительно используя предшествующие знания), в то время как коэффициент, равный 1, заставляет агента учитывать только самую свежую информацию (игнорируя предшествующие знания для изучения возможностей). В полностью детерминированных средах скорость обучения, равная *α* = 1}, является оптимальной. Когда задача стохастическая, алгоритм сходится при некоторых технических условиях к скорости обучения, которые требуют ее снижения до нуля. На практике часто используется постоянная скорость обучения, например *αt*=0.1 для всех *t*.

*γ*(коэффициент дисконтирования) : *γ* коэффициентов дисконтирования определяет важность будущих вознаграждений. Коэффициент, равный 0, сделает агента "близоруким" (или недальновидным), учитывая только текущие вознаграждения, т.е. *rt* (в правиле обновления выше), в то время как коэффициент, приближающийся к 1, заставит его стремиться к долгосрочному высокому вознаграждению. Если коэффициент дисконтирования равен или превышает 1, значения действия могут отличаться. При *γ* =1, без конечного состояния, или если агент никогда не достигает его, все истории среды становятся бесконечно длинными, а утилиты с аддитивными, недисконтированными вознаграждениями обычно становятся бесконечными. Даже при коэффициенте дисконтирования лишь немного меньшем, чем 1, обучение Q-функции приводит к распространению ошибок и нестабильности, когда функция значений аппроксимируется искусственной нейронной сетью. В этом случае, начиная с более низкого коэффициента дисконтирования и увеличивая его до конечного значения, ускоряется обучение.

Начальные условия (*Q0*) : Поскольку Q-learning является итеративным алгоритмом, он неявно предполагает начальное условие до того, как произойдет первое обновление. Высокие начальные значения, также известные как "оптимистичные начальные условия", могут стимулировать исследование: независимо от того, какое действие выбрано, правило обновления приведет к тому, что оно будет иметь более низкие значения, чем другая альтернатива, тем самым увеличивая вероятность их выбора. Первое вознаграждение r может быть использовано для сброса начальных условий. Согласно этой идее, при первом выполнении действия вознаграждение используется для установки значения *Q*. Это позволяет немедленно обучаться в случае фиксированных детерминированных вознаграждений. Ожидается, что модель, включающая сброс начальных условий (RIC), будет предсказывать поведение участников лучше, чем модель, предполагающая любое произвольное начальное условие (AIC). RIC, по-видимому, согласуется с поведением человека в повторяющихся экспериментах с бинарным выбором.

**Upper Confidence Bound(UCB)**

Задача о многоруком бандите:

В обучении с подкреплением мы используем задачу о многоруком бандите, чтобы формализовать понятие принятия решений в условиях неопределенности, используя *k*-вооруженных бандитов. Лицо, принимающее решение, или агент присутствует в задаче о многоруком бандите, чтобы выбрать между *k*-различными действиями, и получает вознаграждение в зависимости от выбранного действия. Задача о бандите используется для описания фундаментальных концепций обучения с подкреплением, таких как вознаграждения, временные интервалы и ценности.



Рисунок 7: игровой автомат, также известный как бандит с двумя рычагами[[7](#UpperConfidenceBound)]

Распределение вероятности получения вознаграждения, соответствующего каждому рычагу, различно и неизвестно игроку (лицу, принимающему решение). Следовательно, цель здесь состоит в том, чтобы определить, за какой рычаг потянуть, чтобы получить максимальное вознаграждение после заданного набора испытаний.

Многорукого бандита (сокращенно: бандит или ЧЕЛОВЕК) можно рассматривать как набор реальных распределений *B = {R1,…,RK}*, каждое распределение связано с вознаграждениями, предоставляемыми одним из *K∈ℕ+* рычагов. Пусть *μ1,…,μK* - средние значения, связанные с этими распределениями вознаграждений. Игрок последовательно разыгрывает один рычаг за раунд и наблюдает за соответствующим вознаграждением. Цель состоит в том, чтобы максимизировать сумму собранных вознаграждений. Горизонт *H* - это количество раундов, которые осталось сыграть. Задача о бандитах формально эквивалентна марковскому процессу принятия решений с одним состоянием. Сожаление ***ρ*** после раунд *T* определяется как ожидаемая разница между суммой вознаграждения, связанной с оптимальной стратегией, и суммой собранных вознаграждений:

Action-Values: Чтобы рекламодатель мог решить, какое действие является наилучшим, мы должны определить ценность совершения каждого действия. Мы определяем эти значения с помощью функции action-value, используя язык вероятности. Ценность выбора действия *q\*(a)* определяется как ожидаемое вознаграждение *Rt*, которое мы получаем при выполнении действия a из возможного набора действий.

# *q\*(a) = E[Rt | A₁ = a]*

Цель агента состоит в том, чтобы максимизировать ожидаемое вознаграждение, выбрав действие, имеющее наибольшую ценность.

# *Qt(a) =*

Разведка против эксплуатации:

* Жадное действие: Когда агент выбирает действие, которое в данный момент имеет наибольшее оценочное значение. Агент использует свои текущие знания, выбирая жадное действие;
* Нежадное действие: Когда агент не выбирает наибольшее оценочное значение и жертвует немедленным вознаграждением в надежде получить больше информации о других действиях;
* Исследование: Это позволяет агенту улучшить свои знания о каждом действии. Надеемся, что это приведет к долгосрочной выгоде;
* Эксплуатация: Это позволяет агенту выбрать жадное действие, чтобы попытаться получить наибольшее вознаграждение за краткосрочную выгоду. Чисто жадный выбор действия может привести к неоптимальному поведению;

Выбор действия с верхней границей достоверности:

Выбор действия с верхней границей достоверности использует неопределенность в оценках ценности действия для сбалансирования разведки и эксплуатации. Поскольку существует неотъемлемая неопределенность в точности оценок ценности действия, когда мы используем выборочный набор вознаграждений, UCB использует неопределенность в оценках для стимулирования разведки.

# *At = argmaxa (Qt(a) + c)*

# *Qt(a)* – эксплойт

*c* – исследование

*t* – временные интервалы

*Nt(a)* - количество раз, когда выполняется действие (*a*)

*Qt(a)* здесь представляет текущую оценку для действия a в момент времени *t*. Мы выбираем действие, которое имеет наибольшее оценочное значение действия плюс срок исследования с верхней доверительной границей.

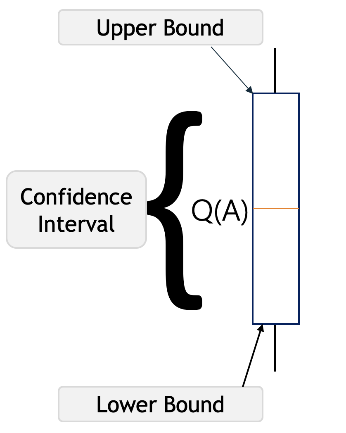


Рисунок 8: текущая оценка стоимости действия для действия A[[8](#UpperConfidenceBound)]

Нижняя скобка называется нижней границей, а верхняя скобка - это верхняя граница. Область между скобками - это доверительный интервал, который представляет неопределенность в оценках. Если область очень мала, то мы становимся абсолютно уверены, что фактическое значение действия A близко к нашему расчетному значению. С другой стороны, если регион большой, то мы становимся неуверенными в том, что значение действия A близко к нашему расчетному значению.

Верхняя доверительная граница соответствует принципу оптимизма перед лицом неопределенности, который подразумевает, что если мы не уверены в каком-либо действии, мы должны оптимистично предполагать, что это правильное действие. Например, предположим, что у нас есть эти четыре действия с соответствующими неопределенностями на рисунке ниже, наш агент понятия не имеет, какое действие является лучшим. Таким образом, согласно алгоритму UCB, он оптимистично выберет действие, имеющее наивысшую верхнюю границу, т.е. A. Делая это, либо оно будет иметь наивысшую ценность и получит наивысшую награду, либо, приняв это, мы узнаем о действии, о котором знаем меньше всего.

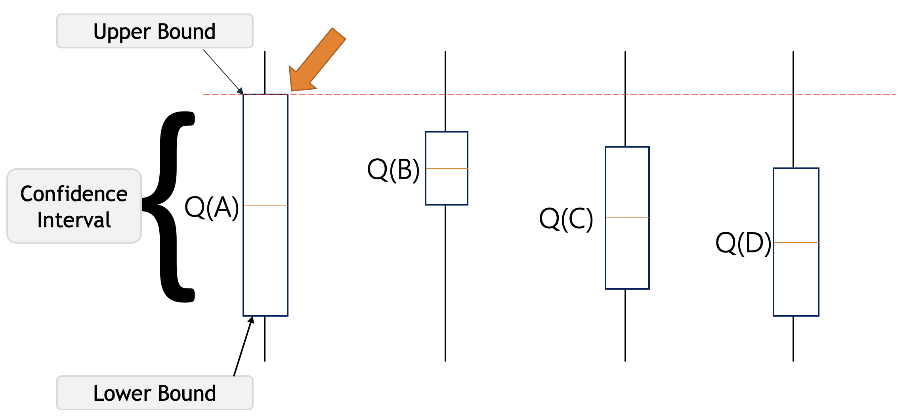


Рисунок 9: верхняя граница, доверительный интервал, нижняя граница[[9](#UpperConfidenceBound)]

Давайте предположим, что после выбора действия A мы попадаем в состояние, изображенное на рисунке ниже. На этот раз UCB выберет действие B, поскольку *Q(B)* имеет самую высокую верхнюю доверительную границу, поскольку оценка значения этого действия является самой высокой, хотя доверительный интервал мал.

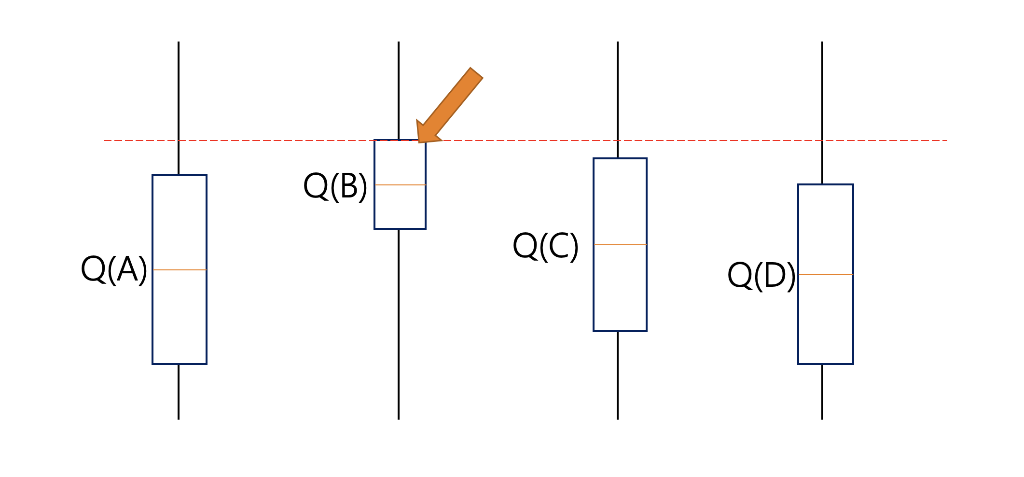


Рисунок 10: Если *Q(B)* верхняя граница[[10](#UpperConfidenceBound)]

Изначально UCB исследует больше, чтобы систематически снижать неопределенность, но со временем его исследование сокращается. Таким образом, мы можем сказать, что UCB в среднем получает большее вознаграждение, чем другие алгоритмы, такие как эпсилон-жадный, оптимистичные начальные значения и т.д.

**Визуализация, Система Вознаграждения и Как QLearn работает**

Здесь мы используем библиотеку, которая называется "Raylib". Эта библиотека является простой библиотекой с открытым исходным кодом для создания игрового окна.

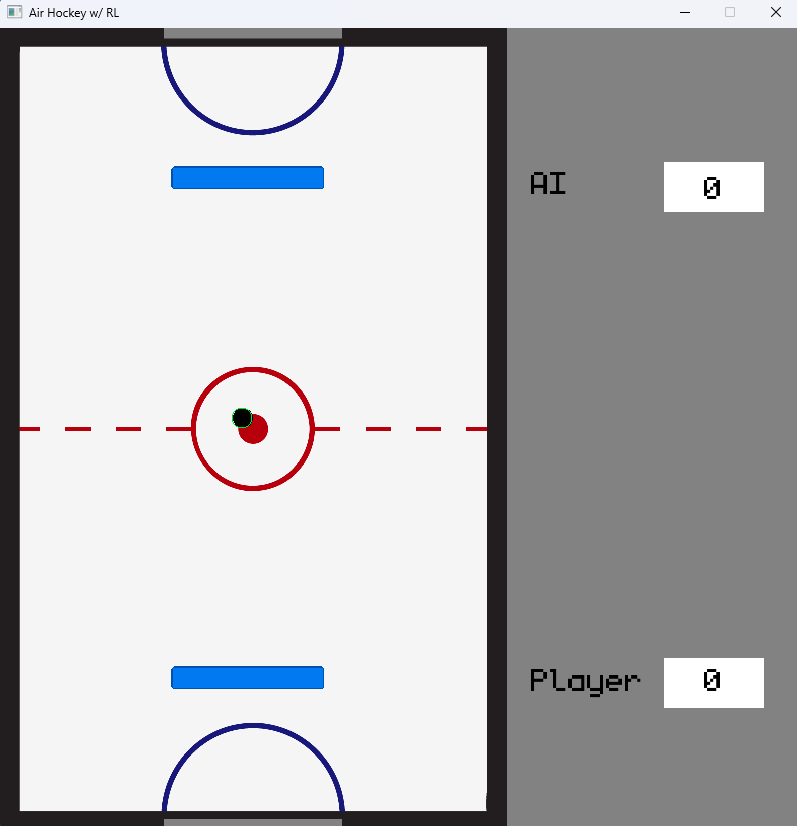


Рисунок 11: Окно для аэрохоккея

Здесь мы рассматриваем окружающую среду (игровую площадку), шайбу, агента и игрока. Игрок и агент будут двигаться только влево и вправо. И шайба будет двигаться в соответствии с координатами *x* и *y* экрана. Если шайба ударит в стену, игрока или агента, она отскочит.

|  |  |  |  |
| --- | --- | --- | --- |
| Ширина Экрана | Высота экрана | Высота фона | Ширина фона |
| 800 | 600 | 800 | 508 |

Таблица 1: Параметры Экрана и Фона

Когда гол забивает игрок или AI, награда будет значительной, чтобы помочь AI учиться. Когда мяч попадает в AI, AI получает 2 очка. И если AI промахивается при попадании по шайбе, он теряет 2 очка.

|  |  |  |
| --- | --- | --- |
| Случай | Полученная награда | Дальнейшее описание |
| AI забивает гол | +10 | Положение шайбы на определенном расстоянии близко к воротам соперника |
| Соперник забивает гол | -10 | Положение шайбы на определенном расстоянии близко к воротам AI |
| Шайба попадает в AI | +2 | Шайба касается AI |
| AI промахивается, чтобы попасть по шайбе | -2 | AI не смог попасть по шайбе |

Таблица 2: Система вознаграждения

Предоставленный код[[11](#prilojeniye)] на C++ реализует алгоритм Q-обучения со стратегией исследования с верхней доверительной границей (UCB) для игры в аэрохоккей. Такой подход к обучению с подкреплением (RL) позволяет агенту искусственного интеллекта со временем обучаться оптимальным действиям посредством взаимодействия с окружающей средой, балансируя между использованием известных хороших действий и изучением новых.

Ключевые компоненты класса QLearn

1. Переменные класса:

* альфа: Скорость обучения, определяет степень, в которой новая информация переопределяет старую;
* гамма: Коэффициент дисконтирования, уравновешивает важность немедленных и будущих вознаграждений;
* эпсилон: Скорость изучения, определяет, как часто изучаются новые действия;
* QTable: Двумерный вектор, хранящий значения состояния-действия, представляющий ожидаемую полезность выполнения действия в данном состоянии;
* totalPlays: Отслеживает общее количество воспроизведений, используемое при расчете UCB;
* playCounts: вектор, отслеживающий количество раз, когда было выполнено каждое действие;

1. Конструктор (QLearn):

* Инициализирует параметры Q-Learning и устанавливает Q-таблицу и playCounts с соответствующими измерениями для состояний и действий;

QLearn::QLearn(double alpha, double gamma, double epsilon, int numStates, int numActions)

: alpha(alpha), gamma(gamma), epsilon(epsilon), totalPlays(0) {

qTable.resize(numStates, std::vector<double>(numActions, 0.0));

playCounts.resize(numActions, 0);

}

1. Выбор действия (chooseAction):

* Принимает решение о действии на основе текущего состояния;
* Использует эпсилон-жадную стратегию: с помощью probability epsilon он выбирает случайное действие для исследования; в противном случае он выбирает наилучшее действие на основе значения UCB;

int QLearn::chooseAction(int state)

1. Функция обновления (update):

* Обновляет Q-таблицу после каждого действия;
* Применяет правило обновления Q-Learning, используя полученное вознаграждение и максимальное будущее Q-значение нового состояния;
* Увеличивает количество воспроизведений для выбранного действия;

void QLearn::update(int state, int action, int reward, int newState)

1. Затухание эпсилона (decreaseEpsilon):

* Постепенно уменьшает значение эпсилона с течением времени, снижая скорость исследования;

void QLearn::decreaseEpsilon(double decayRate)

1. Функция сброса (reset):

* Сбрасывает количество воспроизведений и общее количество воспроизведений, что полезно для начала нового этапа обучения;

void QLearn::reset()

1. Вычисление UCB (getUCBValue):

* Вычисляет значение UCB для заданной пары состояние-действие;
* Объединяет эксплуатацию (на основе текущего значения Q) и исследование (на основе того, как часто выбиралось действие);

double QLearn::getUCBValue(int state, int action)

Применение в аэрохоккее

1. Инициализация:

* Объект QLearn инициализируется с заданными параметрами обучения и размерами для Q-таблицы, соответствующими состояниям и действиям в игре в аэрохоккей;

1. Во время игры:

* Для каждого состояния игры вызывается метод chooseAction для выбора действия (например, перемещения весла влево или вправо);
* Действия выбираются на основе стратегии epsilon-greedy, при этом на исследование влияет значение UCB;

1. После каждого действия:

* Игровая среда предоставляет вознаграждение в зависимости от результата (например, забитый или пропущенный гол);
* Метод обновления корректирует значения Q-таблицы на основе этого вознаграждения и предполагаемых будущих вознаграждений;

1. Адаптация стратегии с течением времени:

* Метод decreaseEpsilon используется для снижения скорости изучения по ходу игры, заставляя агента больше полагаться на свою изученную стратегию;
* Компонент UCB гарантирует, что даже менее часто выбираемые действия изучаются в достаточной степени, чтобы оценить их полезность;

1. Сброс для новых сеансов:

* Функцию сброса можно использовать для очистки исторических игровых данных, что полезно при запуске новой учебной сессии или изменении динамики игры;

**Заключение**

Алгоритм Q-Learning с UCB, реализованный в воздушном хоккее, демонстрирует передовой подход к машинному обучению в играх. Он позволяет агенту искусственного интеллекта динамически изучать эффективные стратегии, балансируя между необходимостью использовать известные стратегии и исследовать новые. Со временем эта система адаптируется, становясь более квалифицированной по мере накопления опыта игры.

**Приложение**

1. Файл “QLearn.h”:

#pragma once

#include <vector>

class QLearn {

public:

double alpha;

double gamma;

double epsilon;

std::vector<std::vector<double>> qTable;

double totalPlays;

std::vector<double> playCounts;

public:

QLearn(double alpha, double gamma, double epsilon, int numStates, int numActions);

~QLearn();

int chooseAction(int state);

void update(int state, int action, int reward, int newState);

void decreaseEpsilon(double decayRate);

void reset();

double getUCBValue(int state, int action);

};

1. Файл “QLearn.cpp”:

#include "QLearn.h"

#include <math.h>

#include <stdlib.h>

#include <algorithm>

QLearn::QLearn(double alpha, double gamma, double epsilon, int numStates, int numActions)

: alpha(alpha), gamma(gamma), epsilon(epsilon), totalPlays(0) {

qTable.resize(numStates, std::vector<double>(numActions, 0.0));

playCounts.resize(numActions, 0);

}

QLearn::~QLearn() {}

int QLearn::chooseAction(int state) {

totalPlays++;

if ((double)rand() / RAND\_MAX < epsilon) {

return rand() % qTable[state].size();

}

else {

double bestValue = -INFINITY;

int bestAction = 0;

for (int a = 0; a < qTable[state].size(); a++) {

double ucbValue = getUCBValue(state, a);

if (ucbValue > bestValue) {

bestValue = ucbValue;

bestAction = a;

}

}

return bestAction;

}

}

void QLearn::update(int state, int action, int reward, int newState) {

double bestFutureQ = \*std::max\_element(qTable[newState].begin(), qTable[newState].end());

qTable[state][action] += alpha \* (reward + gamma \* bestFutureQ - qTable[state][action]);

playCounts[action]++;

}

void QLearn::decreaseEpsilon(double decayRate) {

epsilon \*= decayRate;

}

void QLearn::reset() {

std::fill(playCounts.begin(), playCounts.end(), 0);

totalPlays = 0;

}

double QLearn::getUCBValue(int state, int action) {

if (playCounts[action] == 0) return INFINITY;

double exploitation = qTable[state][action];

double exploration = sqrt(2 \* log(totalPlays) / playCounts[action]);

return exploitation + exploration;

}

**Список Литературы**

1. AI playing Air Hockey – URL:

<https://www.barkhauseninstitut.org/research/lab-1/our-blog/posts/airhockey-and-ai>

1. What is Reinforcement Learning? – URL:

<https://www.synopsys.com/ai/what-is-reinforcement-learning.html>

1. What is Reinforcement Learning and How Does It Work? – URL:

<https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners/>

1. Application of Machine Learning in Air Hockey Interactive Control System – URL:

<https://www.mdpi.com/1424-8220/20/24/7233>

1. Upper Confidence Bound in Reinforcement Learning – URL:

<https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/>

1. Q-Learning Algorithm: From Explanation to Implementation – URL:

<https://medium.com/m/global-identity-2?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2Fq-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187>

1. Q-Learning – URL:

<https://en.wikipedia.org/wiki/Q-learning>

1. Reinforcement Learning – URL:

<https://en.wikipedia.org/wiki/Reinforcement_learning>

1. Обучение с подкреплением – URL:

<https://disk.yandex.ru/i/mWpic_8A3TT5V6>

1. Машинное обучение. Обучение с подкреплением. – URL:

<https://www.youtube.com/watch?v=iEUrX_eEWNY&list=PLJOzdkh8T5krxc4HsHbB8g8f0hu7973fK&index=20>

1. Многорукие бандиты: Введение и алгоритм UCB1 – URL:

<https://habr.com/en/companies/surfingbird/articles/168611/>