**Обзор аналогов и выбор метода решения**

Для выбора метода решения рассмотрим основные методы разработки игрового искусственного интеллекта и некоторые реализации для компьютерных игр жанра стратегии в реальном времени (Real-Time Strategy, RTS), в основном, для StarCraft.

I. Методы разработки игрового искусственного интеллекта:

1. Основанный на правилах (Scripting) [18].

Метод, основанный на правилах, был и остаётся популярным методом разработки искусственного интеллекта, так как он прост в реализации. Суть метода заключается в разработке правил, согласно которым действуют игровые сущности. Часто для этих целей разрабатывается высокоуровневый скриптовый язык, что ещё сильнее упрощает разработку. Помимо простоты, заранее прописанные сценарии быстры при выполнении, так как для запуска определённого поведения требуется лишь несколько игровых фреймов.

Чтобы сделать достаточно сложный ИИ, требуется предусмотреть огромное число игровых ситуаций и для каждой из них выбрать поведение игровых сущностей. Таким образом, усложнение поведения приводит к быстрому увеличению сложности программного кода, поддерживать который становится всё труднее.

2. Конечные автоматы (Final State Machines, FSMs) [20].

Конечные автоматы рассматриваются как один из простейших подходов к реализации поведения ИИ в играх. Причина популярности конечных автоматов заключается в простоте их использования, так как концепция состояний и условных переходов между ними весьма понята.

Поведение искусственного интеллекта зависит от состояния, в котором находится система, переходящая при срабатывании определённых условий из одного состояния в другое. Когда сложность ИИ возрастает, количество состояний и переходов между ними будет быстро увеличиваться, а для добавления состояния потребуется просмотреть все прочие, чтобы добавить все необходимые переходы.

Расширение конечных автоматов, иерархические конечные автоматы (hierarchical final state machine, HFSMs), решает некоторые из этих проблем. Эта модель вводит концепцию модульности, позволяя группам состояний совместно использовать переходы. Её цель состоит в том, чтобы избежать избыточных переходов и получить лучший обзор модели. Используя HFSMs, также становится проще группировать состояния и действия для формирования моделей поведения.

Таким образом, конечные автоматы представляют простой и интуитивно понятный метод создания искусственного интеллекта, но сложны при расширении и изменении, так как добавление или модификация поведения требует пересмотра и возможной корректировки всей модели. Также трудно предусмотреть все последствия, которые эти изменения могут вызвать.

3. Деревья поведения (Behavior Trees, BTs) [19].

Деревья поведения — это относительно новый подход к проектированию ИИ. BTs сочетают в себе элементы как из методов, основанных на правилах, так и из иерархических конечных автоматов, чтобы обеспечить гибкую структуру, доступную как геймдизайнерам, так и программистам, с минимальными сложностями, насколько это возможно. Структура BTs также предназначена для обеспечения более масштабируемого подхода, чем HFSMs, за счет снижения структурной сложности. ИИ принимает решение, основываясь на указанных условиях. У него есть возможность рассматривать сложные условия без учета комбинаций состояний. При использовании модели разработчик заботится об обходе дерева в глубину и слева направо.

В отличие от конечных автоматов, в деревьях поведения логика принятия решений отделена от кода самого состояния. Вся логика переходов между состояниями вынесена в отдельную структуру, поэтому при добавлении нового поведения не нужно пересматривать все переходы и состояния.

BTs предоставляют интуитивно понятную структуру, которая может быть легко применена к игровому сценарию. Структура и семантика BTs позволяют легко отслеживать поток поведения, что полезно как для понимания, так и для модификации и отладки существующего поведения. Модульность, обеспечиваемая деревьями поведения, хорошо масштабируется по мере увеличения вариантов поведения.

Как и в случае с HFSMs, по-прежнему необходимо поддерживать обзор структуры и идентифицировать отдельные части поведения, чтобы разложить его на древовидную структуру. По этой причине BTs может демонстрировать некоторые из тех же проблем, что и HFSMs, где части дерева могут стать очевидными для игрока, тем самым разрушая иллюзию интеллекта.

4. Планировщики (Planning).

В классическом планировании [21] состояния представлены наборами параметров или отношений, связанных между собой действиями, позволяющими менять значения этих параметров и, непосредственно, состояние среды. Действия не представлены в виде простых функций в среде, а вместо этого описываются схемами действий. Схемы действий определяют как действия, так и последствия этих действий. То есть, выбирая действие для плана, мы знаем соответствующий эффект этого действия, где эффект — это результирующее новое состояние. Набор предварительных условий для действия также включен в схему действия. Действие может быть предпринято только в том случае, если эти предварительные условия выполняются.

Проблему планирования последовательности действий можно рассматривать как проблему поиска. Начиная с начального состояния, мы можем просто выполнять поиск в пространстве состояний, переходя по доступным действиям из состояния в состояние, пока не будет достигнуто целевое состояние. Из-за декларативного представления схем действий существует два возможных способа сделать это: прямой поиск из начального состояния и обратный поиск из целевого состояния.

Во время выполнения плана может быть использовано перепланирование, если в результате следования плану будут достигнуты неожиданные состояния. Это приводит к некоторым накладным расходам, поскольку необходимо генерировать новый план. В зависимости от рассматриваемой задачи планирования пространство состояний может стать настолько большим, что простой поиск становится невозможным. Для преодоления этой проблемы используется эвристическая функция.

4.1. Иерархическая сеть задач (Hierarchical Task Networks, HTN).

Подобно классическому планированию, планирование иерархической сети задач использует понятие сопряженных состояний для представления схем для описания действий. Однако, по сравнению с классическим планированием, мы не планируем достижение цели, а вместо этого выполняем некоторый набор задач.

Задача определяется как один или несколько составных элементов последовательных упорядоченных действий или задач. Каждая задача содержит описание различных последовательностей, на которые она может быть разложена. Планирование иерархической сети задач — это процесс рекурсивной декомпозиции этих задач до тех пор, пока не останется только последовательность действий [22].

Планировщики HTN в значительной степени используют имеющиеся знания о том, как можно декомпозировать задачи. То есть всякий раз, когда мы сталкиваемся с задачей, для которой план уже был составлен путем декомпозиции, мы можем использовать этот план вместо того, чтобы искать план повторно. Это может значительно сократить вычислительное время, затрачиваемое на поиск новых планов.

Специалисты по планированию HTN также могут расширить существующие знания с помощью новых декомпозиций задач. Эту базу знаний иногда называют библиотекой планов. Мощь библиотеки планов также является одной из причин, по которой планирование HTN получило более широкое применение в практических приложениях по сравнению с другими методами планирования. Разработчики игрового ИИ, например, могут заранее определить декомпозицию задач высокого уровня, чтобы ИИ действовал так, как ожидалось, при выполнении определенных задач. С другой стороны, игровой ИИ способен самостоятельно конструировать поведение для слабо определенных задач.

4.2. Целеориентированное планирование действий (Goal-Oriented Action Planning, GOAP).

GOAP является расширением STRIPS (аббревиатура от STanford Research Institute Problem Solver) [23], которое аналогично классическому планированию.

Расширение GOAP было предложено Джеффом Оркином из M.I.T. Media Lab с целью разработки A.I. для шутера от первого лица F.E.A.R. от Monolith Productions [24]. GOAP добавляет планировщику STRIPS несколько расширений. Наиболее важными из этих расширений являются включение затрат на действия и добавление процедурных предварительных условий и эффектов к схемам действий. Стоимость действий может быть использована в качестве эвристики при поиске планов в пространстве состояний.

Добавление процедурных предварительных условий расширяет формальный взгляд на мир и позволяет осуществлять дополнительную фильтрацию по состояниям. Последнее расширение, рассмотренное здесь, — это добавление процедурных эффектов к схемам действий. Декларативный характер этих эффектов налагает мгновенное изменение состояния при выполнении действия. В сценариях реального мира действия не обязательно выполняются мгновенно и вместо этого требуют некоторого времени для выполнения. В GOAP используются оба этих понятия состояний. Элемент времени реализуется путем подключения системы планирования к FSMs. При выполнении действия в GOAP процедурные эффекты устанавливают состояние FSM до тех пор, пока действие не будет завершено и среда не перейдет в новое состояние.

5. Обучение с подкреплением (Reinforcement Learning, RL).

Обучение с подкреплением можно описать как изучение того, какие действия следует предпринимать в среде, чтобы максимизировать или минимизировать некоторое кумулятивное вознаграждение [3]. Однако обучение с подкреплением не определяется конкретным методом достижения этого обучения. Вместо этого он определяется как характеризующий набор проблем. Любой метод, который помогает решить эти проблемы, называется методом обучения с подкреплением.

Проблему обучения с подкреплением в целом можно визуализировать на рисунке 1. Здесь представлен агент, который взаимодействует со средой посредством некоторых действий a. Каждое действие приводит к некоторому изменению в среде, представленной агенту в виде состояний, s. Информация об окружающей среде, представленная в состояниях, зависит от наблюдаемости окружающей среды для агента. Требуется, чтобы эта среда была, по крайней мере, частично наблюдаемой для агента.

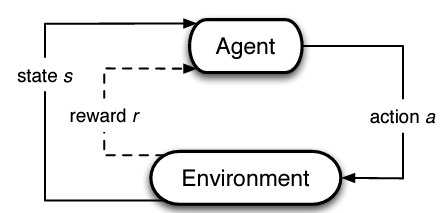


Рисунок 1. Взаимодействие среды и агента в обучении с подкреплением.

Изменения в среде могут вызвать некоторое вознаграждение r для агента, как отрицательное, так и положительное. Это вознаграждение дается функцией подкрепления, которая сопоставляет пары состояний и действий с вознаграждениями. Обучающая часть проблемы заключается в определении пар состояний и действий, которые приводят к максимальному или минимальному совокупному вознаграждению, пока не будет достигнуто некоторое конечное состояние.

Вознаграждение за действия агента может быть не немедленным, а вместо этого зависеть от рассматриваемого способа обучения с подкреплением. Кроме того, цель максимизации или минимизации совокупного вознаграждения также зависит от подхода.

Поиск функции оптимального значения является ключевой проблемой обучения с подкреплением, потому что, начиная с любого состояния, мы будем знать не только значение этого состояния, но и значения всех достижимых состояний с помощью доступных действий. Простое принятие мер, которые приводят к состояниям с максимумом, равносильно следованию оптимальному поведению. Таким образом, совокупное вознаграждение или подкрепление будет максимизировано, что и было целью обучения с подкреплением. Найти функцию оптимального значения сложно, и обычно это делается путем аппроксимации с помощью динамического программирования.

6. Заключение

В данном разделе были рассмотрены основные методы разработки искусственного интеллекта для игр.

Наиболее используемыми в индустрии остаются метод, основанный на правилах, и конечные автоматы, оба из которых сложны в поддержке и расширении поведения.

Конечные автоматы предоставляет понятную структуру для дизайнеров, которая не допускает повторного использования логики. Это приводит к созданию конструкции с увеличивающимся числом повторяющихся состояний, что делает ее излишне сложной. Проблема повышенной сложности проектирования была решена созданием иерархических конечных автоматов путем введения суперсостояний, но метод по-прежнему страдает от отсутствия возможности повторного использования и повторяющихся состояний.

Деревья поведения были разработаны, чтобы компенсировать отсутствие возможности повторного использования в иерархических конечных автоматах, позволяя каждому узлу инкапсулировать свою собственную логику. Делая это, узел будет содержать некоторое поведение, основанное на нем самом и его дочерних элементах. Это поведение можно рассматривать как независимое от остального контекста и может быть повторно использовано там, где это необходимо. Эта конкретная функция делает деревья поведения очень масштабируемыми по сравнению с HFSMs, и создание сложных моделей поведения будет менее утомительным.

Все три рассмотренных метода имеют некоторые общие недостатки. Поведение, созданное на основе правил, с помощью HFSMs или BTs, будет реагировать в соответствии с предопределенным поведением. Ни один из методов не предусматривает какой-либо техники для планирования наперед. Таким образом, поведение будет несколько статичным, поскольку методы не предусматривают непредсказуемых игровых событий. Если бы игрок-человек видел модель насквозь, он мог бы легко принять контрмеры, приводящие к уверенной победе.

Планирование очень важно при рассмотрении человекоподобного поведения в играх. Человеческое поведение в целом в значительной степени зависит от заблаговременного планирования не только во время игр, но и в повседневной жизни. По этой причине планирование прекрасно вписывается в концепцию стратегического мышления, необходимого в играх RTS. Где мы ставим перед собой некоторые цели и планируем способ их достижения. Для сравнения, FSMs и BTs полагаются на реактивное планирование. То есть они только планируют следующее действие, которое необходимо предпринять, исходя из текущего контекста мира.

Обучение, как и планирование, также важно при рассмотрении поведения людей. При выполнении задач мы получаем опыт, будь то в результате успеха или неудачи, и этот опыт вспоминается в следующий раз, когда нам придется выполнять аналогичную задачу. Было рассмотрено обучение с подкреплением. Этот подход, однако, попытается найти оптимальную последовательность действий для достижения конечного состояния. Чтобы включить это в игру RTS, награда, получаемая за действие, должна быть определена таким образом, чтобы последовательность действий не приводила к сверхчеловеческому поведению. Вместо этого вознаграждение должно основываться на игровом опыте, который является переменной, которую мы хотим максимизировать. Это, пожалуй, самая большая проблема стимулирования обучения в такой игре, как StarCraft, поскольку игровой опыт трудно поддается определению.

II. Аналоги.

В отбираемые аналоги вошли существующие программы реализации искусственного интеллекта для игр жанра RTS (далее - Программы), преимущественно для StarCraft. Многие из них используют как классические подходы, описанные выше, так и более уникальные, чаще всего совмещающие несколько подходов.

Большинство аналогов были взяты из статьи 2014 года «A review of real-time strategy game AI» [1]. В ней авторы сгруппировали используемые методы на три группы: Tactical-decision making (принятие тактических решений, микроменеджмент), Strategic-decision making (принятие стратегических решений, макроменеджмент) и Plan recognition and Learning (распознавание плана и обучение).

1) Возможные варианты решения:

1. Sarsa [2] (Обучение с подкреплением)

2. Alpha-Beta Considering Durations [4] (Деревья решений)

3. Bayesian model [5] (Байесовская вероятностная модель с использованием обучения с подкреплением)

4. rtNEAT [6] (Нейронные сети)

5. Darmok [7] (Планирование с использованием машинного обучения)

6. Generalized Sequential Patterns [9] (Генерация иерархической сети задач на основе игровых записей)

7. EISBot [10] (Планирование с использованием машинного обучения)

8. Soar [12] (Когнитивная архитектура)

9. Build order optimization [13] (Целеориентированное планирование сборки)

10. HICOR [14] (Дедуктивное распознавание планов в виде дерева решений)

2) Описание аналогов

1. Tactical-decision making

Тактические решения принимаются при краткосрочном контроле над отдельными подразделениями.

1.1. Sarsa [2]

Алгоритм Sarsa использует обучение с подкреплением - область машинного обучения, в которой агент должен методом проб и ошибок научиться оптимальным действиям в конкретных ситуациях, чтобы максимизировать общую ценность вознаграждения [3]. Sarsa используется для обучения управлению юнитами в небольших перестрелках. Авторы использовали искусственные нейронные сети, чтобы узнать ожидаемую награду за атаку или бегство с конкретным подразделением в данном состоянии, и выбрали действие с наибольшей ожидаемой наградой в игре. Система научилась побеждать встроенные сценарии искусственного интеллекта StarCraft в среднем только в небольших перестрелках из трех единиц, при этом ни одна из вариаций не научилась побеждать встроенные сценарии в среднем в перестрелках из шести единиц.

1.2. ABCD (Alpha-Beta Considering Durations) [4]

Алгоритм ABCD заключается в поиске по дереву решений, но с ограничением глубины поиска. В конечных узлах оценивается максимальная полезность путем вызова функции оценки, и это значение распространяется вверх по дереву, на основе чего и создаётся последовательность действий.

Алгоритм разрабатывался для SparCraft - симуляции игры StarCraft c возможностью произвольно манипулировать состоянием (https://code.google.com/archive/p/sparcraft/).

Основным ограничением этой системы является то, что из-за комбинаторного взрыва возможных действий и состояний по мере увеличения количества юнитов, количества возможных действий в StarCraft и ограничения по времени в 5 мс на игровой кадр поиск позволит использовать только до восьми юнитов с каждой стороны в битве двух игроков. С другой стороны, лучшие результаты могут быть достигнуты с помощью моделирования противника, поскольку поиск может включать известные действия противника вместо поиска по всем возможным действиям противника. Когда это было протестировано на стратегиях с идеальной моделью каждого противника, поиск смог достичь по крайней мере 95%-ного коэффициента выигрыша против каждого из сценариев в симуляции.

1.3. Bayesian model (Байесовская модель) [5]

Программа, основанная на Байесовской модели, учитывает цели, возможности и угрозы подразделений, чтобы решить, в каком направлении перемещать подразделения в бою. Модель рассматривает каждый из своих сенсорных входных сигналов как часть уравнения вероятности, которое может быть решено, учитывая данные (потенциально полученные с помощью RL) о распределении входных сигналов относительно направления перемещения, чтобы определить вероятность того, что единица должна двигаться в каждом возможном направлении. Можно выбрать наилучшее направление или перебрать вероятности направления, чтобы избежать перемещения двух единиц в одно и то же место. Эта байесовская модель работает в паре с иерархическим конечным автоматом для выбора различных наборов поведения, когда подразделения вступают в бой с вражескими силами или избегают их, или ведут разведку. Созданная Программа была очень эффективна против встроенного искусственного интеллекта StarCraft.

1.4. rtNEAT [6]

Нейроэволюция — это метод, который использует эволюционный алгоритм для создания или обучения искусственной нейронной сети. rtNEAT - нейроэволюционный алгоритм для разработки как топологии, так и весов соединений нейронных сетей для управления отдельными подразделениями в StarCraft. В этом алгоритме каждый юнит имеет свою собственную нейронную сеть, которая получает входные данные из источников окружающей среды (таких как близлежащие юниты или препятствия) и абстракции, определенные вручную (такие как количество, тип и “качество” близлежащих юнитов), и выдает решение атаковать, отступать или двигаться влево или вправо. Во время игры производительность юнитов оценивается с помощью разработанной вручную функции пригодности, а плохо работающие агенты юнитов заменяются комбинациями наиболее эффективных агентов. rtNEAT тестировался в очень простых сценариях 12 против 12 юнитов на квадратной арене, где все юниты с каждой стороны являются либо единицами рукопашного боя, либо единицами дальнего боя. В таких ситуациях он учится побеждать встроенный в StarCraft искусственный интеллект и некоторые другие Программы. Однако остается неясным, насколько хорошо он справится с большим количеством единиц или набором различных типов единиц.

2. Strategic-decision making

Принятие стратегических решений происходит на высоком уровне с учётом долгосрочных целей. Для совершения разумных действий на стратегическом уровне многие исследователи создали системы планирования. Эти системы способны определять последовательность действий, которые необходимо предпринять в конкретной ситуации для достижения определенных целей. Это сложная проблема из-за неполной доступной информации – “туман войны” скрывает области поля боя, которые находятся вне поля зрения дружественных подразделений, - а также огромные пространства состояний и действий и множество одновременных неиерархических целей. С помощью систем планирования исследователи надеются позволить ИИ играть на человеческом уровне, одновременно сокращая затраты на разработку по сравнению с методами на основе правил, обычно используемыми в промышленности. Основными методами, используемыми для систем планирования, являются планирование на основе конкретных случаев (CBP), автономия, основанная на достижении целей (GDA), и иерархическое планирование.

2.1. Darmok [7]

Darmok был разработан для игры в WARGUS, реализацию знаменитой Warcraft II с открытым исходным кодом. Использует метод Case-Based Planning (CBP, планирование на основе конкретных случаев). Случаи изучаются из игровых записей с комментариями человека, причём в каждом случае подробно описываются цели, которых человек пытался достичь с помощью определенных последовательностей действий в определенном состоянии. Эти случаи затем могут быть адаптированы и применены в игре, чтобы попытаться изменить игровое состояние. Рассуждая о дереве целей и подцелей, которые необходимо выполнить, можно выбрать кейсы и связать их вместе в план для достижения общей цели - победы в игре.

Системы CBP могут проявлять низкую реактивность на стратегическом уровне и чрезмерную реактивность на уровне действий, не реагируя на изменения ситуации на высоком уровне до тех пор, пока действие на низком уровне не завершится неудачей, или отказываясь от всего плана из-за неудачи одного действия [8].

2.2. Generalized Sequential Patterns [9]

Generalized Sequential Patterns - алгоритм последовательного анализа шаблонов. GSP работает, выполняя серию сканирований последовательностей данных, каждый раз совершая поиск частых шаблонов на один элемент длиннее, чем при предыдущем сканировании. Таким образом, обучаясь на игровых записях, система извлекает шаблоны поведения в разных игровых ситуациях. Генерируемые закономерности создаются как на микро-, так и на макроуровне. Полученные шаблоны в последствии можно использовать для генерации сложных задач для модели Hierarchical Task Network (HTN, Иерархическая сеть задач), которую никогда напрямую не использовали в области RTS AI.

2.3. EISBot [10]

EISBot использует модель Goal-Driven Autonomy (GDA) для одновременного анализа в нескольких масштабах. Используемый язык поведения способен формировать планы с ожиданиями относительно результата. Если возникает неожиданная ситуация или событие, система может записать это как несоответствие, сгенерировать объяснение, почему это произошло, и сформировать новую цель для пересмотра плана, что позволяет системе соответствующим образом реагировать на непредвиденные события. Изначально было невозможно изучить цели, ожидания или стратегии, поэтому эти знания приходилось вводить и обновлять вручную, но более поздние усовершенствования позволили извлечь их из демонстрации [11].

2.4. Soar [12]

Soar - когнитивная архитектура, использующая пространственную визуальную систему Spatial Vision System (SVS) для ведения разведки и определения пути, а также рабочую память для хранения воспринимаемой и аргументированной информации о состоянии. Однако в настоящее время она ограничена частичной игрой в StarCraft, используя только базовые казармы и подразделения пехоты для ведения боя и используя жестко заданные локации для размещения зданий.

2.5. Build order optimization [13]

Алгоритм оптимизации сборки использует автоматизированный подход поиска в пространстве состояний для планирования заказов на сборку в играх RTS. Алгоритм сосредоточен на одной цели: найти план строительства желаемого набора единиц и зданий за минимальное время. Домен RTS упрощается за счет абстрагирования сбора ресурсов от нормы дохода на одного работника, предполагая, что размещение зданий и перемещение юнитов занимает постоянное количество времени, и полностью игнорируя противников. Игнорирование оппонентов довольно разумно для начала игры, поскольку, как правило, взаимодействие с оппонентами невелико, и это означает, что планировщику не приходится иметь дело с неопределенностью и внешними воздействиями на состояние. Данный метод требует экспертных знаний, чтобы обеспечить целевое состояние для их достижения.

Из-за вычислительных затрат на планирование позже в игре планирование было ограничено 120 секундами вперед, с перепланировкой каждые 30 секунд. Это привело к созданию более коротких или равных по длине планов для игроков-людей в начале игры и планов аналогичной длины в среднем (с большей дисперсией) позже в игре. Еще предстоит выяснить, насколько хорошо этот метод будет работать на более поздних этапах игры, поскольку оценивались только первые 500 секунд, а во второй половине поиск занял значительно больше времени. Тем не менее, это, по-видимому, эффективный способ получения почти оптимальных заказов на сборку, по крайней мере, для раннего и среднего этапов игры StarCraft.

3. Plan Recognition

Некоторые исследователи сосредоточились на подзадаче определения стратегии противника, которая особенно сложна в играх RTS из-за неполной информации о действиях противника, скрытой “туманом войны”. Большинство методов распознавания планов используют существующую библиотеку планов для сопоставления при попытке распознавания стратегии, но некоторые методы позволяют распознавать планы без каких-либо предопределенных планов. Часто данные извлекаются из широкодоступных файлов повторов опытных игроков-людей.

Одним из примеров решения задачи распознавания плана является Дедуктивное распознавание плана, идентифицирующее план путем сравнения ситуации с гипотезами ожидаемого поведения для различных известных планов. Наблюдая за конкретным поведением, можно сделать вывод о предпринимаемом плане, даже если полное знание недоступно. Система HICOR[14] выполняет распознавание плана в StarCraft путем сопоставления наблюдений за своим противником со всеми известными стратегиями, которые могли бы создать данную ситуацию. Затем она моделирует возможные планы, чтобы определить ожидаемые будущие действия своего противника, оценивая вероятность планов на основе новых наблюдений и отбрасывая планы, которые не совпадают. Используемый метод требует значительных человеческих усилий для описания всех возможных планов в структуре типа дерева решений.

3) Критерии сравнения аналогов.

1. Существует ли Программа, реализовывающая описанный метод ИИ или представлен только алгоритм.

2. Способна ли Программа провести полную игру от начала до конца, или же обеспечивает только ограниченный функционал для определённой области игры.

3. Обучаемость. Может ли рассматриваемый алгоритм обучаться на игровых записях.

4. Зависимость от экспертных знаний и многократных повторов.

Может ли предложенный метод искусственного интеллекта быть использован при отсутствии игровых записей. Например, можно ли будет применить эту технику при создании искусственного интеллекта для новой игры или не такой популярной, как StarCraft.

5. Принятие решений на стратегическом уровне.

Учитывает ли метод ИИ цели на стратегическом уровне при выборе действий или планировании.

4) Таблица сравнения аналогов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Sarsa | ABCD | Bayesian model | Darmok | GSP |
| Реализована Программа | + | + | + | + | - |
| Программа полноценная | - | - | - | + | - |
| Обучаемость | + | + | + | + | + |
| Отсутствие зависимости от экспертных знаний | + | + | + | - | - |
| Принятие решений на стратегическом уровне | - | - | - | + | + |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | EISBot | Soar | BOO | HICOR |
| Реализована Программа | + | + | + | + |
| Программа полноценная | + | - | - | + |
| Обучаемость | + | + | + | + |
| Отсутствие зависимости от экспертных знаний | - | - | + | - |
| Принятие решений на стратегическом уровне | + | + | + | + |

III. Заключение

Мало кто из исследователей создал полноценную Программу, поэтому сложно судить об эффективности этих подходов.

Большое количество Программ, участвующих в соревнованиях[26] среди реализаций искусственного интеллекта для StarCraft, рассмотрено в [15]-[16]. Большинство из них использует метод, основанный на правилах, с применением машинного обучения, стремясь предусмотреть наибольшее количество игровых ситуаций. Это объясняется тем, что участники соревнований стремятся просто победить, а для этого не нужно разрабатывать новые инновационные методы. К сожалению, к Программам нет достаточно подробного описания, кроме их успехов в соревнованиях, поэтому они не рассматривались в данном обзоре.

Все рассмотренные в разделе II методы поддерживают машинное обучение, но многие из них зависят от наличия экспертных знаний, то есть игровых записей. Это может вызывать ряд проблем, таких как переобучение, невозможность внедрения ИИ при отсутствии игровых записей, например, в новых или менее популярных играх.

Таким образом, разрабатываемое решение должно удовлетворять следующим свойствам:

1. Решение должно быть представлено в виде полноценной Программы для игры жанра стратегия в реальном времени StarCraft без упрощений в виде сокращения игрового пространства, доступных для Программы и встроенного ИИ действий и прочих;

2. Разрабатываемая Программа должна обладать возможностью обучаться на игровых записях, что позволит разнообразить её поведение, сделать более человекоподобным и уменьшить время на разработку новых вариантов поведения. Но при этом Программа должна воспроизводить полноценную игру и без машинного обучения, чтобы разработанную систему можно было применить и для других игр на момент выпуска, а также во избежание таких проблем машинного обучения, как переобучение.

3. Программа должна принимать решения как на тактическом (на уровне действий отдельных юнитов или подразделений юнитов), так и на стратегическом (выбор стратегии поведения на высоком уровне) уровнях.

Рассмотрим основные методы разработки игрового искусственного интеллекта, описанные в разделе I, и обсудим их соответствие этим критериям.

Метод, основанный на правилах, очень прост в реализации при использовании машинного обучения, но при отсутствии экспертных знаний учёт всех возможных игровых ситуаций и разработка соответствующего поведения – очень трудоёмкая задача. Также с усложнением поведения становится всё труднее поддерживать код реализации.

Конечные автоматы позволяют описать поведение как на тактическом, так и на стратегическом уровне, но системы на их основе не обучаемы. Поэтому на всех этапах разработки все состояния и переходы пришлось бы описывать вручную. Также конечные автоматы становится гораздо сложнее поддерживать при добавлении или изменении поведения.

Деревья поведения обладают теми же преимуществами, что и конечные автоматы, но при этом обучаемы [25] и намного проще в поддержке и расширении. К тому же неправильно усвоенное поведение можно будет изменить вручную, а также добавить собственное для конкретных сценариев.

Использование планировщиков требует детального представления состояния среды, что в играх жанра RTS проблематично из-за тумана войны и невозможности учитывать действия соперника, пока он вне поля зрения. Среди описанных выше аналогов есть использующие планирование, но, в основном, в ограниченном масштабе (без реализации в виде Программы или только для некоторого аспекта игры, как, например, порядок сборки) и с использованием обучения.

Иерархическая сеть задач упоминалась в разделе II.2.2.2 в методе Generalized Sequential Patterns, который на основе шаблонов способен генерировать такую сеть, но не обходится без обучения и пока не реализован в виде Программы, способной принимать на основе полученной сети задач какие-либо решения.

Похожий на целеориентированное планирование действий (GOAP) метод упоминался в разделе II.2.2.5, где описывается Build order optimization, который генерирует оптимальный порядок сборки построек и юнитов на ранней стадии игры. Даже будучи ограниченным лишь такими задачами стратегии в реальном времени, как постройка, найм юнитов и сбор ресурсов, системе пришлось устанавливать ограничение по времени поиска. Таким образом, метод поиска последовательности действий в пространстве состояний не подходит для создания полноценной Программы, а может использоваться только в качестве оптимизации некоторых игровых задач, так как количество возможных действий на более поздних этапах игры будет настолько большим, что система не будет успевать генерировать актуальный план.

Обучение с подкреплением используется системой Sarsa (раздел II.2.1.1), отвечающей за действия подразделений юнитов на тактическом уровне. Использование этого метода на стратегическом уровне будет проблематично из-за сильной отсрочки получаемого вознаграждения.

Исходя из всего вышесказанного, можно сделать вывод, что наиболее подходящим решением в реализации искусственного интеллекта являются деревья поведения, которые, тем не менее, обладают большим недостатком. При выполнении определённых условий, предусмотренных разработчиком или усвоенных в процессе обучения, поведение агента всегда будет одинаковым, без учёта прочих факторов или непредвиденных событий. Поведение агента реактивно, так как выбирает одно действие и совершает его без продумывания наперёд. Тем не менее, Программа, основанная на деревьях поведения, будет отвечать всем требованиям, описанным выше.

Чтобы сделать поведение Программы более человекоподобным, требуется использование планировщиков, но слишком большое количество доступных действий в стратегической игре не позволит генерировать актуальный план. Поэтому логичнее было бы планировать поведение не поиском среди атомарных игровых действий, а поиском среди некоторых стратегий, соединяющих эти действия в некоторые подпоследовательности. Поэтому следующим шагом после создания деревьев поведения может быть добавление планировщика, который бы сделал переходы по деревьям поведения недетерминированными.

**Источники и литература**

1. Robertson G., Watson I. A review of real-time strategy game AI //Ai Magazine. – 2014. – Т. 35. – №. 4. – С. 75-104

2. Shantia, A.; Begue, E.; and Wiering, M. 2011. Connectionist reinforcement learning for intelligent unit micro managementinStarCraft. Presented at the International Joint Conference on Neural Networks

3. Sutton, R. S., and Barto, A. G. 1998. Reinforcement learning: An introduction. Cambridge Massachusetts: MIT Press

4. Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. In Proceedings of the AIIDE Conference, 112–117

5. Synnaeve, G., and Bessiere, P. 2011b. A bayesian model for RTS units control applied to StarCraft. In Proceedings of the IEEE Conference on Computational Intelligence and Games, 190–196

6. Gabriel, I.; Negru, V.; and Zaharie, D. 2012. Neuroevolution based multi-agent system for micromanagement in real-time strategy games. In Proceedings of the Fifth Balkan Conference in Informatics, 32–39. ACM.

7. Ontanón S. et al. Case-based planning and execution for real-time strategy games //International Conference on Case-Based Reasoning. – Springer, Berlin, Heidelberg, 2007. – С. 164-178.

8. Palma R. et al. Combining expert knowledge and learning from demonstration in real-time strategy games //International Conference on Case-Based Reasoning. – Springer, Berlin, Heidelberg, 2011. – С. 181-195.

9. Leece M. A., Jhala A. Sequential pattern mining in Starcraft: Brood War for short and long-term goals //Tenth Artificial Intelligence and Interactive Digital Entertainment Conference. – 2014.

10. Weber B., Mateas M., Jhala A. Applying goal-driven autonomy to starcraft //Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2010. – Т. 6. – №. 1. – С. 101-106.

11. Weber, B.; Mateas, M.; and Jhala, A. 2012. Learning from demonstration for goal-driven autonomy. In Proceedings of the AAAI Conference on AI, 1176–1182

12. Soar-SC [Электронный ресурс] URL: <https://github.com/bluechill/Soar-SC> (Дата обращения: 30.11.2022)

13. Churchill, D., and Buro, M. 2011. Build order optimization in StarCraft. In Proceedings of the AIIDE Conference, 14–19.

14. Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A.; and Irandoust, H. 2010. Opponent behaviour recognition for real-time strategy games. In Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition.

15. Čertický M. et al. Starcraft AI competitions, bots, and tournament manager software //IEEE Transactions on Games. – 2018. – Т. 11. – №. 3. – С. 227-237.

16. Ontanón S. et al. A survey of real-time strategy game AI research and competition in StarCraft //IEEE Transactions on Computational Intelligence and AI in games. – 2013. – Т. 5. – №. 4. – С. 293-311.

17. Liquipedia [Электронный ресурс] URL: <https://liquipedia.net/> (Дата обращения: 10.12.2022)

18. Sweetser P., Wiles J. Scripting versus emergence: issues for game developers and players in game environment design //International Journal of Intelligent Games and Simulations. – 2005. – Т. 4. – №. 1. – С. 1-9.

19. Weber B.G., Mateas M., Jhala A. Building human-level AI for real-time strategy games, Proc. AIIDE Fall Symp. Adv. Cogn. Syst. - 2011. - С. 329-336.

20. Frutos-Pascual M., Zapirain B. G. Review of the use of AI techniques in serious games: Decision making and machine learning //IEEE Transactions on Computational Intelligence and AI in Games. – 2015. – Т. 9. – №. 2. – С. 133-152.

21. Russell S. J. Artificial intelligence a modern approach. – Pearson Education, Inc., 2010.

22. P. T. Mallik Challab, Dana Nau. Automatic Planning- Theory and Practice. Number ISBN: 1-55860-856-7 in 1st Edition. Morgan Kaufmann, 2004

23. Fikes R. E., Nilsson N. J. STRIPS: A new approach to the application of theorem proving to problem solving //Artificial intelligence. – 1971. – Т. 2. – №. 3-4. – С. 189-208

24. Orkin J. Three states and a plan: the AI of FEAR //Game developers conference. – SanJose, California : CMP Game Group, 2006. – Т. 2006. – С. 4.

25. Robertson G., Watson I. Building behavior trees from observations in real-time strategy games //2015 International symposium on innovations in intelligent systems and applications (INISTA). – IEEE, 2015. – С. 1-7.

26. Buro M., Churchill D. Real-time strategy game competitions //Ai Magazine. – 2012. – Т. 33. – №. 3. – С. 106-106.