Документация

Описание классов

AMyAIVehicle

Класс описывает параметры и методы для восприятия окружающей среды и для управления транспортным средством.

Параметры

UBoxComponent* collisionBox - куб, вмещающий в себя трансопртное средство. Создается автоматически при запуске. **FVector collisionBoxSize** - Размеры куба. Задается до старта. **FVector collisionBoxShift** - Смещение куба. На поворотах куб поворачивается вместе с машиной, из-за этого может быть пересечение куба и земли. В своей реализации я чуть поднимаю. Задается до старта. **float lineTraceLenght** - Длина луча. Задается до старта. **bool showDebug** - Отладочный параметр. Если *true*, отрисовывает лучи и показывает их результат. Задается до старта. **int countOverlaps** - Количество пересечений куба с другими объемами. Доступно только для чтения.

Методы

TArray GetNormalizedLineTraceResult() - Испускает лучи из точек, которые основаны на размере и положении куба (солнышком во все стороны, 16 штук). Возвращает расстояние до столкновения деленное на длину луча *lineTraceLenght* для каждого луча. Если столкновения нет, то 1.
bool HaveOverlap() - возвращает *true*, если количество пересечений не равно 0. Иначе *false*.

AMyAIController

Класс описывает методы для использования данных об окружающей среде и обеспечивает связь между транспортным средством, behaviorTree и нейронной сетью.

Методы

void MoveVehicleUsingNN(TArray inputs, float currentRevard) - Переводит входящие параметры в нужный тип. Запрашивает у нейронной сети результат. Награждает ее, в соответствии текущих обстоятельств.
int GetNNResult(vector inputs, float currentRevard) - возвращает результат нейронной сети в качестве номера действия.
void SetVehicleMovement(float ThrottleInput, float SteeringInput, bool HandbrakeInput) - Передвигает управляемое транспортное средство с заданными параметрами.

DQNAgent

Класс позволяет создавать нейронную сеть определенной архитектуры и обучает в соответствии с QDN-алгоритмом.

Методы

void AddWorldAnswer(vector inputs, double reward) - Принимает текущее состояние и награду. Если уже есть объект *DQNMemory*, то добавляет в него эти параметры и сохраняет их в объекте класса *DQNReplayMemory*. **int GetAction(vector inputs)** - Создает новый объект *DQNMemory*, сохраняет в него *inputs* и результат нейронной сети в виде номера действия. Возвращает номер действия.

DQNRaplayMemory

Класс реализует собой инструмент для хранения и воспроизведения событий.

Методы

int Lenght() - возвращает длину вектора, хранящего состояния, награды и действия
*DQNReplayMemory AddAgentAction(vector inState, int inAction)** - Создает новый экземпляр ячейки памяти. Сохраняет в него текущее состояние и действие. *DQNReplayMemory AddWorldAnswer(vector inNextState, double inReward)** - В созданную ячейку памяти добавляет награду и следующее состояние. Добавляет ячейку памяти в вектор.
*vector<AgentMemory> SampleMemory(int size)** - возвращает набор ячеек памяти. Очищает вектор.

DQNMemory

Просто хранит в себе параметры.

Параметры

vector inState - Текущее состояние.
int inAction - Действие, которое было принято в текущем состоянии.
vector inNextState - Состояние, в которое мы в итоге пришли.
double inReward - Награда.

Описание всего и вся

В начале я думал как заставить машину кататься. Ответ пришел быстро, "по рельсам". Построить кривую, и по ней ориентироваться. Однако я не знаю как, используя кривую, объезжать препятствия. Был вариант вести перед собой коробку, для определения столкновений, и в случае, если столкновение будет, поворачивать, сползать с рельс. И крутиться вокруг этого столкновения, пока снова не встричу рельсы. Но тут было столько "НО", которые я не знаю как обработать. Поэтому я решил использовать то, в чем более-менее уверен. Нейронные сети.

Я решил разобраться в алгоритме DQN. Я его даже закодил.

Сейчас я буду описывать алгоритм своими словами. Если Вы его знаете, смело проматывайте.

Q

Для описания DQN нужно понимать, как работает Q алгоритм.

Представим, что у Вас есть решетка, как та, что в судоку, 9 на 9. У вас есть хомяк, он находится в какой-то клетке. Так же клетки могут содержать в себе мышеловку или семечко. Если хомяк заходи в мышеловку, он умирает, игра перезапускается. Сам хомяк ничего не знает о мире. Он не знает, что наступать в мышеловки - плохо, есть семечко - хорошо. И Вы хотите, чтобы хомяк нашел кратчайший путь до семечка, при этом не заходя на клетки с мышеловками.

Q - обучение предлагает ввести награду. Нашел семечку - условные +10. Наступил в мышеловку - условные -5. В таком случае у хомяка есть две вещи, от которых он может отталкиваться - текущее состояние (номер клетки), и действия (двинуться вправо, влево, вверх, вниз).

Также каждому состоянию мы придаем значение (в начале 0 у всех), зависящее от полученных наград и состояний ближайших клеток. А теперь говорим хомяку "делай то, что принесет большую пользу".

Так вот Q-функция говорит куда ему двигаться, номер действия. В начале хомяк будет действовать рандомно, может встанет в мышеловку, может в семечко. Суть в том, что со временем, клетки с мышеловкой получат плохую оценку, а с семечкой - хорошую. Клетки рядом с семечками получат большую оценку и наоборот, и тд. Через какое-то количество итераций хомяк твердо запомнит маршрут по сбору семечек.

Еще следует рассказать про еплисон-функцию. На каждой итерации есть какой-то шанс, что хомяк сделает не то, что ему говорит голос разума, а рандомное действие. Это помогает расшатывать устаявшийся не оптимальный путь.

DQN

Собственно такие же идеи, но с нейронками. Проблема в том, что мы не можем для каждого входа в нейронную сеть создать состояние (ну, можем, но их очень и очень много) Поэтому создается две нейронки и хранитель памяти. Суть в том, что если раньше у нас была сетка и Q-функция отталкивалась от нее, то сейчас мы с помощью нейронной сети аппроксимируем Q-функцию. В хранителе памяти заложены несколько важных числел - вместимость хранилеща, при достижения которого происходит обучение, и номер итерации, при котором значения одной нейронки копируются в другую.

Моя реализация

При помощи OpenNN (c++ библиотеки для нейронок) реализовал DQN. Сама нейронная сеть - просто перцептрон (просто вершины, без ухищрений). На вход приходит 20 параметров. 16 из них - значения от 0 до 1, которые мы получаем от AMyAIVehicle. 3 - UpVector машинки. Ее заносит на углах, поэтому лучи пересекают пол, и чтобы нейронка не думала что это стена и это опасно, даю ей это вектор, значения от -1 до 1. И последний - угол между направлением машинки и углом до цели. Угол делиться на 180, значения, соответственно от -1 до 1. На выходе 18 значений. Выбираем индекс максимального - этот индекс и есть номер действия. 18 их, по соображения, что если человеку хватает WASD и пробела, чтобы ехать, то и ей наверное хватит. Собственно 3 варианта поворота - право, лево, прямо. Три варианта езды - вперед, назад, никуда. Ну и ручник - вкл, выкл. 3 на 3 на 2 = 18

Также я решил поставить четыре машинки в разных местах. Четыре, потому что в 4 раза быстрее достигается предел вместимости хранилища. Разные, у одной стена рядом, другая развернута, чтобы данные не повторялись. Награду я высчитываю в разнице расстояний до цели на прошлой итерации и сейчас. Если было столкновение, то -5.

И создал behaviorTree, который выдает случайную точку для машины.

Если коротко, то я надеялся вместо кривой использовать просто точки, а вместо постройки алгоритма по нахождению пути научить нейронку делать это самостоятельно.

Резюме

У алгоритма есть важный плюс - ModelFree. Это значит, что его можно использовать без обработки окружающего мира. Просто нужно не радикально (-1000000 за столкновение и +1 за достижение цели не надо) настроить награды.
Это все таки обучение, что заставляет достигать максимума.
Также плюс - скорость вычислений.

Из минусов, для моей реализации важен размер. Ну, тут я скорее не уверен, если условная ветка - преграда, то нужно или что-то другое использовать, или использовать не лучи а плоскости или объемы (я не знаю, можно ли так в UE, выпустить не из точки, а из отрезка или грани). В случае гонок, мне кажется, наименьшим объемом будет какая-нибудь бочка, а это вполне можно обойти небольшим увеличением количеством лучей и их обработкой.

Так же к минусу стоит отнести то, что алгоритм, после обучения, не будет работать хорошо на всех машинах, кататься они будут, но он как бы привыкнет к той машинке, к ее параметрам. Однако, думаю, что повторное обучение обучение на уснове уже созданной нейронки все исправит.

В моем случае я не смог доделать, так как не знаю как справиться с этой ошибкой:
undefined symbol: std::__cxx11::
Вроде и статическую библиотеку сделал, но, видно, что-то не учел. Так же потратил много времени на использование OpenNN. Библиотека подразумевает, что у вас уже есть все данные для обучения (там это класс DataSet), поэтому я переписал оптимизатор.
В дальнейшем, думаю, стоит или написать свой код для нейронок (там не так уж и много, алгоритмы для СЛАУ - публичны и быстры, а требуется не так много, как есть в OpenNN, несколько классов слоев, несколько функций активации, сама нейронка, пара оптимизаторов и пара функций ошибок).
А еще в период нашего собеседования я перешел с винды на линукс, что тоже внесло моментов для изучения.

Спасибо за тестовое, жду ответа.