МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по практической работе №2
по дисциплине «Интеллектуальные системы»
Тема: «Движение игрока по маршруту»

Студент гр. 8382 _____ Мирончик П.Д.

Преподаватель _____ Беляев С.А.

Санкт-Петербург
2022

ЗАДАНИЕ

Целью работы является решение задачи целенаправленного движения агента с использованием видимых ориентиров.

Для достижения поставленной цели необходимо решить следующие задачи:

– подключиться к серверу игры и передать начальные координаты игрока;

– проанализировать ответы сервера;

– определить направление движения, дать соответствующие команды серверу.

ХОД РАБОТЫ

1. Описание алгоритмов

В ходе выполнения работы были разработаны несколько алгоритмов, позволяющих игроку выполнять простейшие действия.

1.1. Требуемые данные о цели

Для полноценной работы алгоритмов требуется знать некоторую информацию о цели. В частности это:

– координаты цели (`cords`);

– направление цели (относительно направления взгляда) (`direction`);

– расстояние до цели (`distance`).

При этом рассмотрены ситуации нехватки данных:

1. Отсутствуют координаты. Координаты цели можно вычислить, зная текущие координаты игрока, направление до цели и расстояние до цели – вычисление координат осуществляется с использованием написанного в первой работе кода.

2. Отсутствует направление. Направление цели можно вычислить, зная координаты игрока и направление его взгляда. Для этого ищется нормированный вектор $\overline{PT}$, где P – игрок, T – цель, и находится угол между этим вектором и нулевым вектором (направлением взгляда).

3. Отсутствует расстояние. Расстояние можно вычислить, используя координаты объекта. Если координаты неизвестны и имеется только направление, то используется условное расстояние в 100 единиц, и по нему вычисляются другие необходимые параметры.

Таким образом, единственный случай, когда данных о цели недостаточно – отсутствие одновременно и координат цели, и ее направления.

1.2. Движение к точке

Движение к цели `target` осуществляется с заданной точностью `presision` и скоростью `power`. Обработка выполняется по шагам:

1. Если `target.distance < presision`, цель достигнута.

2. Если `target.direction > 10`, т.е. угол отклонения вектора взгляда от вектора к цели более 10 градусов, осуществляется поворот к цели.

3. В противном случае осуществляется движение вперед со скоростью `power`.

1.3. Движение к точке с мячом

Задачей движения с мячом является доставка мяча в указанную точку с указанной точностью `presision` и скоростью движения игрока `power`. Процесс движения:

1. Если мяч не виден, осуществляется поиск мяча поворотом игрока по часовой стрелке. Если известно предполагаемое направления мяча, то осуществляется поворот к этому направлению.

2. Если мяч уже находится в требуемой точке, цель достигнута.

3. Если расстояние до мяча меньше 0.5, т.е. мяч игрок может пнуть мяч, осуществляется несильная передача в направлении цели. При этом

определяется ожидаемое направление мяча как направление удара – мяч может быть отправлен в том числе за спину игроку.

4. В противном случае осуществляется движение к мячу по алгоритму п. 1.2 с точностью 0.5.

1.4. Забивание гола

Цель этой задачи очевидна – забить гол в ворота противника. Для этого выполняется:

1. Если мяч не виден, осуществляется поиск мяча аналогично п. 1.3.1.

2. Если расстояние между мячом и воротами (их центральной точкой) более 30, выполняется алгоритм движения к точке с мячом из п. 1.3, где целью являются ворота.

3. Если игрок находится далеко от мяча (расстояние до мяча более 0.5) выполняется движение к мячу.

4. Игрок выполняет удар по мячу в направлении ворот с максимальной силой.

2. Подключение алгоритмов в программе

Описанные в п. 1 алгоритмы были реализованы в классе Controller. Он хранит список целей и указатель на текущую цель, например:

```
[
  { coords: { x: -10, y: -10 }, t: 'move' },
  { coords: { x: 10, y: 10 }, t: 'dribble' },
  { coords: { x: 10, y: -10 }, t: 'move' },
  { coords: { x: -10, y: 10 }, t: 'dribble' },
  { t: 'goal' },
]
```

Класс Agent при инициализации создает экземпляр Controller и связывает с собой.

В конце каждого такта контроллер выполняет текущую операцию. Если операция выполнена, то он переходит к следующей. При этом если статус игры, полученный через команду head, отличается он play_on, контроллер

сбрасывает текущую операцию к первой в списке и не выполняет никаких команд, ожидая начала игры.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы была реализована программа, выполняющая движение агента в симуляторе RoboCup Soccer Simulation и достигающая простейших целей, таких как перемещение к объекту, ведение мяча и забивание гола.