Заключение

В данной статье было дано общее описание и актуальность автоматизированного склада, его принципы и сложности в организации. Основная идея заключается в перемещении самих товаров к складским рабочим, а не наоборот. При автоматизации склада, повышается скорость и точность выполняемых операций (доставка и отправка товаров). На данный момент авторами ведется работа по исследованию и разработке алгоритмов, связанных с задачами, которые возникнут при организации такого склада. Описанный в данной статье автоматизированный склад является фактически роботизированным складом.

Несмотря на то, что такие склады организуются, они все еще слишком дорогие. Исходя из опыта существующих проектов в США, можно сказать, что роботизированный склад стоит примерно 0,5-1 миллион долларов на каждую тысячу квадратных метров склада. В эту цену включается стоимость оборудования, монтаж и строительство всех конструктивных элементов внутри здания, запуск в работу, обучение персонала.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Рассел, С. Искусственный интеллект: Современный подход / С. Рассел, П. Норвиг. – 2 изд., 2007. – 86-88 с.

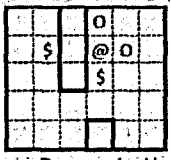
УДК 004.021

Никонович В.Б., Михневич В.А.
Научный руководитель: доцент Дунец А.П.

РЕШЕНИЕ ЗАДАЧИ «SOKOBAN» НА БАЗЕ МЕТОДОВ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА. ПРЕОБРАЗОВАНИЕ КАРТЫ ИГРЫ В АГЕНТНУЮ СИСТЕМУ

«Sokoban» – это логическая игра-головоломка. По своей сути игра сокобан – это виртуальная модель склада, где главный герой (кладовщик) должен расставить все ящики на заранее заданные позиции. Математики-теоретики причисляют эту задачу к классу пр-сложных задач, таких как шахматы, шашки или кубик Рубика. Универсального алгоритма решения этой задачи до сих пор никем предложено не было, несмотря на то, что исследования ведутся уже давно. Актуальность решения этой задачи переоценить невозможно, поскольку она является подклассом более общей задачи поиска пути. Но основным стимулом к разработке описанного ниже подхода послужила перспектива использования этих алгоритмов в автоматизированных складах, обслуживаемых роботами.

Предположим, что имеется следующая карта игры сокобан (рис.1):



O – точка
$ – ящик
@ – толкатель

Рисунок 1 – Начальная карта

Первым делом необходимо узнать, каким образом можно попасть в точку, то есть из каких координат. Для этого требуется рассчитать матрицу *достижимости*. Принцип её расчёта сходен с волновым алгоритмом, но маркер в координату ставится при выполнении определённого условия. Это условие следующее, чтобы поставить маркер в точку i+1, требуется, чтобы точка i+2 также была свободна. Такое требование возникает в результате того, что, чтобы затолкнуть ящик в маркированную точку, сокобан должен стать в противоположной стороне от маркера.

Выглядит матрица глобальной достижимости следующим образом (рис.2):

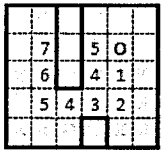


Рисунок 2 – Матрица глобальной достижимости точки

Далее, после вычисления матрицы достижимости требуется вычислить матрицу перемещения ящика. Для вычисления этой матрицы нам требуется иметь возможность вычислять достижимость сокобаном определенной точки. Например, чтобы переместить ящик на ЮГ, сокобан должен толкнуть его с СЕВЕРА. Поэтому перед тем как поставить маркер с юга, нужно проверить достижимость сокобаном северной точки. Предположим, что мы реализовали этот алгоритм, тогда матрица перемещения ящика будет выглядеть следующим образом (рис.3):

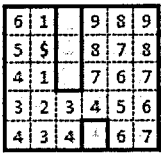


Рисунок 3 – Матрица глобального перемещения ящика

После вычисления всех матриц достижимости и матриц перемещения ящиков требуется вычислить приоритет точек. Это нужно сделать для того, чтобы после установки ящика на точку, этот ящик не заблокировал подходы к другим точкам и тем самым не завёл игру в тупик. Вычисляется приоритет следующим образом. Делим все ящики на группы. Количество групп равно количеству точек, и каждая точка соответствует одной из групп. Ящик будет находиться в группе, если он находится в зоне достижимости точки, соответствующей этой группе. Для рассматриваемого случая эти группы будут иметь следующий вид:

- *Группа1(0,3) = {(1,1), (2,3)}*
- *Группа2(1,4) = {(1,1), (2,3)}*

В данном примере оба ящика могут попасть в любую точку, поэтому тут нет ограничения как на соответствие ящика точке, так и на порядок помещения ящиков на точки.

Алгоритм поиска гипотез и приоритетов следующий. Первым делом требуется найти самую маленькую группу. После нахождения такой группы берём первый ящик из группы и проверяем эту пару на соответствие необходимым условиям. Если условия выполнены, то присуждаем этой точке высший приоритет № 1 и удаляем эту группу из списка групп, а также удаляем этот ящик из всех групп. Затем увеличиваем счётчик приоритета до № 2 и повторяем описанную выше процедуру снова. Так продолжается до тех пор, пока список групп не станет пустым. Если же необходимые условия не выполнены, то переходим к следующему ящику в этой группе и делаем проверку на необходимые условия с новым ящиком. Если точка не удовлетворяет условиям ни с одним ящиком, то её приоритет не равен № 1, поэтому переходим к следующей по размеру группе и проводим описанную выше процедуру.

Выше упоминалось соответствие необходимым условиям. Проверяется возможность соответствия точки ящику с определенным приоритетом. Для этого нужно удалить проверяемый ящик с карты игры и поставить на проверяемую точку стену(непроходимое препятствие). Затем вычислить матрицы достижимости для всех оставшихся точек и матрицы перемещения для всех оставшихся ящиков. И проанализировать эти матрицы следующим образом:

- *Если все точки имеют в зоне достижимости хотя бы один ящик.*
- *Если каждый ящик может попасть хотя бы в одну точку.*

Если эти два условия выполняются, то считается, что данная пара ящик-точка удовлетворяет условию, поэтому она может считаться успешной и получить соответствую-

щий приоритет. Для рассматриваемого примера приоритеты и гипотезы будут следующими:

Priority\Hypo: {[(1, 4), (1, 1)], [(0, 3), (2, 3)]}

После того как появились гипотезы о соответствии ящика точке, можно начинать превращать ящик в агента, для которого точка будет являться локальной целью.

Выполнять превращение начинаем с вычисления локальных правил перемещения агента. Осуществлено это будет следующим образом. Нужно пересечь матрицу достижимости точки и матрицу перемещения соответствующего точке ящика. На выходе получаем матрицу перемещения агента. Осуществляется это очень просто, из матрицы перемещения ящика удаляем все маркеры, находящиеся за пределами зоны достижимости точки.

Выглядит это следующим образом:

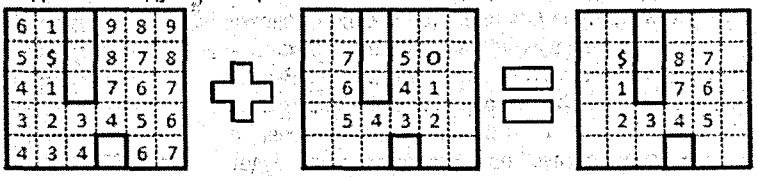


Рисунок 4 – Вычисление матрицы перемещения агента

Теперь можно из карты-ситуации игры сокобан получить мультиагентную систему. Остаётся только реализовать прозрачный интерфейс управления толкателем через призму мультиагентного подхода.

Преобразование движения агента в движение толкателя выполняется в два этапа:

- *Выполнить трекинг агента до пункта назначения.*
- *Выполнить трекинг толкателя до каждого шага агента.*

Сам треклист содержит последовательность действий для объекта, в случае с игрой сокобан, объект может совершить только четыре вида действия:

- *Движение на ЮГ (Ю)*
- *Движение на СЕВЕР (С)*
- *Движение на ВОСТОК (В)*
- *Движение на ЗАПАД (З)*

Для вычисления трекинга пути агента пользуемся следующим алгоритмом:

1. Вычислить матрицу локальной достижимости до цели агента.

2. Найти области вокруг агента маркер с наименьшим номером.

3. Записать направление, по которому найден маркер в конец треклиста.

4. Переместить агента на место маркера.

5. Проверить, не достигнута ли цель. Если достигнута, то завершить формирование пути. Если еще не достигнута, то вернуться к пункту (2).

Например:

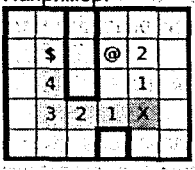


Рисунок 5 – Путь агента к цели

На картинке выше символом «X» обозначена цель агента. А серым цветом выделен его путь. Треклист агента в данной ситуации будет выглядеть следующим образом:

Tracklist = {Ю, Ю, В, В, В}

После того как закончили вычисление пути агента, требуется выполнить трекинг толкателя. Алгоритм используется

тот же что и для трекинга агента. Разница лишь в том, что цель для задаётся на основе треклиста агента.

Выглядит это так:

1. Из треклиста агента выбирается очередное направление.

2. Выполняется вычисление и занесение в треклист пути толкателя до точки, находящейся в противоположном направлении от агента.

3. К треклисту толкателя добавляется выбранное направление агента.

4. Агент перемещается по выбранному направлению.

5. Из треклиста агента удаляется выполненное действие.

6. Если треклист агента пуст, то заканчиваем формирование треклиста толкателя. Иначе возвращаемся к пункту (1)

В результате в треклист запишется путь, который должен пройти толкатель для того, чтобы агент попал в заданную цель.

Например: (треклист для первого хода агента)

1	X		12	11	12
2	$		@	10	11
3	4		8	9	10
4	5	6	7	8	9
5	6	7		9	10

Рисунок 6 – Путь толкателя к агенту

На рисунке выше символом «X» выделена цель толкателя, а серым отмечены позиции, через которые он должен пройти. Треклист толкателя в данной ситуации будет выглядеть следующим образом:

Tracklist = {Ю,Ю,3,3,С,3,С,С,В,Ю}

В результате имеем алгоритмы преобразования игры сокобан в агентную систему, а также имеем алгоритмы для прозрачного управления игрой через сформированную агентную систему. Тем самым образовался более высокий уровень абстракции. Он позволит применять популярные мультагентные подходы к задаче сокобан. Планируется провести имитационное моделирование и проверить эффективность разработанных алгоритмов.

УДК 004.896:621.865

Склипус Д. Б.

Научные руководители: доцент Дунец А.П., к.т.н., доцент Костюк Д.А.

РЕШЕНИЕ ТИПОВЫХ НАВИГАЦИОННЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ АВТОНОМНОГО МОБИЛЬНОГО РОБОТА

В последнее время наблюдается увеличение доли бытового использования автономных мобильных роботов; благодаря широкому распространению дешевых и простых в реализации средств связи и портативных вычислительных систем для данного направления робототехники прогнозируется скачкообразный рост. Ведущие профильные вузы реагируют на наличествующий и ожидаемый в будущем спрос введением дополнительных учебных курсов.

Соответствующие лабораторные практикумы чаще всего строятся на базе различных виртуальных сред моделирования. Изучение программирования электронных устройств невозможно без практических экспериментов, и это особенно важно в случае управления автономной мобильной системой. Такой аппарат в своей физической реализации взаимодействует с широким спектром факторов и воздействий окружающей среды, подвержен ряду физических закономерностей, учет которых нехарактерен для систем мо-