МИНОБРНАУКИ РОССИИ

ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Математические основы искусственного интеллекта»

на тему: «Искусственный интеллект для игры в Сапёра»

|  |  |
| --- | --- |
| Выполнил:  студент гр. Б06-191-1 | Р.И. Мусин |
| Принял:  к.т.н., доцент | А.В. Коробейников |

Ижевск 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. ОБЗОР ИГРЫ 4

1.1. Принцип игры 4

1.2. Варианты 4

1.3. Оценка сложности поля 5

1.4. Рекорды 5

1. ПОСТАНОВКА ЗАДАЧИ 7
2. АЛГОРИТМ 8
   1. Описание алгоритма «обучение с подкреплением» 8
   2. Q-Learning 8
   3. Использованный алгоритм 10

ЗАКЛЮЧЕНИЕ 11

СПИСОК ЛИТЕРАТУРЫ 12

ПРИЛОЖЕНИЕ 1 13

ПРИЛОЖЕНИЕ 2 15

ВВЕДЕНИЕ

История игры "Сапёр" уходит в далекие 50-е года 20-го столетия. В то время это была конечно же не компьютерная игра, т.к. персональные компьютеры появились гораздо позже, а игра в большой картонной коробке.

Игровое поле представляло собой трёхслойный бумажный punchboard. В верхнем слое были отверстия, в которые нужно было ударять молоточком во второй слой, который в свою очередь был защитной пленкой и скрывал под собой третий слой, на котором были изображены мины и цифры в точности под каждым отверстием первого слоя.

Если игрок пробивал все отверстия, под которыми не было мин, то он мог получить от производителя приз - еще одну такую же игру.

Таким образом, "Сапёр" стал очень популярной и увлекательной игрой, а одновременно и прибыльным товаром для производителя.

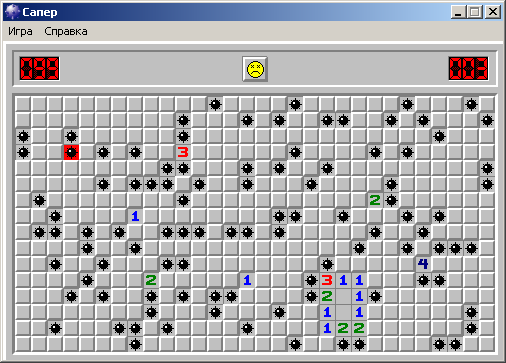
С эрой появления персональных компьютеров игра "Сапёр" обрела свою еще большую популярность.

Самым первым компьютерным прародителем этой игры можно считать игру "Куб", созданную Дэвидом Алом.

Вскоре после этого, в 1985-м году, взяв идею игры "Куб", трое программистов из США "родили" на свет игру Relentless Logic (RLogic), которая работала под операционной системой MS-DOS и все действия игры происходили не в кубе, а на прямоугольной поверхности.

Еще одним вариантом игры послужила игра Mines, разработанная Томом Андерсоном в 1987г., в основе которой была игра RLogic, только с немного измененными правилами.

А с появлением операционной системы Windows "Сапёр" приобрел свою самую большую популярность. Теперь в нее стали играть во всем мире.



В настоящее время популярность этой логической игры нисколько не уменьшилась. Даже наоборот - появились в интернете целые сообщества и форумы, посвященные игре "Сапёр", периодически проводятся чемпионаты внутри многих стран и чемпионаты мира.

1. ОБЗОР ИГРЫ
   1. Принцип игры

Плоское или объёмное игровое поле разделено на смежные ячейки (квадраты, шестиугольники, кубы и т. п.), некоторые из которых «заминированы»; количество «заминированных» ячеек известно. Целью игры является открытие всех ячеек, не содержащих мины.

Игрок открывает ячейки, стараясь не открыть ячейку с миной. Открыв ячейку с миной, он проигрывает. Мины расставляются после первого хода, поэтому проиграть на первом же ходу невозможно. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано» (в каждом варианте игры соседство определяется по-своему); используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры. «Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки, игрок выигрывает.

* 1. Варианты

Существуют варианты игры с полем и/или ячейками непрямоугольной формы, в трёхмерном пространстве, многопользовательские варианты. В некоторых разновидностях цифры, обозначающие количество заминированных ячеек, различаются по цвету. В стандартных вариантах «Сапёра» и «Mines» цифра 1 — синяя, 2 — зелёная, 3 — красная, 4 — тёмно-синяя.

При некоторых расстановках мин вычислить их расположение невозможно, и игрок оказывается перед необходимостью открывать ячейки наугад. Сапёр для Windows имеет интерактивное поле, если все возможные комбинации на поле уже открыты, то взрыва не последует даже при нажатии наугад на любой неоткрытый участок поля.

* 1. Оценка сложности поля

Часто сложность поля оценивают с помощью величины 3BV (Bechtel’s Board Benchmark Value). Эта величина численно равна минимальному количеству непосредственных открытий ячеек (в стандартном варианте «Сапёра» открытия ячеек только левой кнопкой мыши, без использования флагов и двойных кликов), необходимому для открытия всего поля. Следует отметить, что эта величина отображает лишь количество определённых действий в идеальном случае при определенной манере игры, а вовсе не трудность расстановки для конкретного игрока.

* 1. Рекорды

В большинстве вариантов игры подсчитывается время решения головоломки, поэтому регистрируются рекорды для стандартных уровней сложности игры. Для серьёзных соревнований используются версии игры, фиксирующие время прохождения с точностью до миллисекунд.

Результат сильно зависит от расположения мин. Теоретически при любых игровых параметрах есть вероятность прохождения одним щелчком. Но практическая реализация генератора случайных комбинаций не позволяет получить слишком простую расстановку на больших досках. Поэтому результаты на уровнях сложности Intermediate и Expert хорошо отражают уровень игрока. В официальных программах установлены ограничители для простых досок по 3bv. В настоящее время они составляют 2 для уровня сложности Beginner, 30 — Intermediate и 100 — Expert. См. также исследование достаточности ныне установленного ограничения для «Профессионала», вызванное результатом Яна Фрейзера..

Также летом 2016 года японец прошел "Сапёра" с 777 бомбами на что ушло у него 12 часов. А предпринимал попытки он в течение 10 лет.

1. ПОСТАНОВКА ЗАДАЧИ

Требуется написать игру Сапёр с искусственным интеллектом, основанным на алгоритме обучения с подкреплением с использованием алгоритма Q-Learning. На каждом ходу игрок выбирает клетку, в которой предположительно нет мины, и открывает её. Если в клетке оказалась мина, игрок проигрывает, иначе игра продолжается. Как только на поле стали открыты все клетки, не содержащие мину, игра считается выигранной.

Программа разработана на языке Java.

3. АЛГОРИТМ

3.1. Описание алгоритма «обучение с подкреплением»

Обучение с подкреплением (англ. reinforcement learning) — один из способов машинного обучения, в ходе которого испытуемая система (агент) обучается, взаимодействуя с некоторой средой. С точки зрения кибернетики, является одним из видов кибернетического эксперимента. Откликом среды (а не специальной системы управления подкреплением, как это происходит в обучении с учителем) на принятые решения являются сигналы подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или её модель. Также нужно иметь в виду, что некоторые правила подкрепления базируются на неявных учителях, например, в случае искусственной нейронной среды, на одновременной активности формальных нейронов, из-за чего их можно отнести к обучению без учителя.

* 1. Q-Learning

Q-обучение (Q-learning) — метод, применяемый в искусственном интеллекте при агентном подходе. Относится к экспериментам вида oбучение с подкреплением. На основе получаемого от среды вознаграждения агент формирует функцию полезности Q, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-обучения — то, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды. Применяется для ситуаций, которые можно представить в виде марковского процесса принятия решений.

Алгоритм:

1. Init (Инициализация):
   1. for each s and a do Q[s, a] = RND // 0..1
2. Observe (Наблюдение):
   1. s' = s // Запомнить предыдущие состояния
   2. a' = a // Запомнить предыдущие действия
   3. s = FROM\_SENSOR // Получить текущие состояния с сенсора
   4. r = FROM\_SENSOR // Получить вознаграждение за предыдущее действие
3. Update (Обновление ценности):
   1. Q[s',a'] = Q[s',a'] + LF \* (r + DF \* MAX(Q,s) — Q[s',a'])
4. Decision (Выбор действия):
   1. a = ARGMAX(Q, s)
   2. TO\_ACTIVATOR = a
5. GO TO 2

Обозначения

* LF — это фактор обучения. Чем он выше, тем сильнее агент доверяет новой информации.
* DF — это фактор дисконтирования. Чем он меньше, тем меньше агент задумывается о выгоде от будущих своих действий.

Функция MAX(Q,s)

1. max = minValue
2. for each a of ACTIONS(s) do
   1. if Q[s, a] > max then max = Q[s, a]
3. return max

Функция ARGMAX(Q,s)

1. amax = First of ACTION(s)
2. for each a of ACTION(s) do
   1. if Q[s, a] > Q[s, amax] then amax = a
3. return amax
   1. Использованный алгоритм

Каждый ход делается независимо от предыдущих ходов и основывается только на текущем состоянии игрового поля. Поиск клетки для открывания выполняется следующим образом:

* 1. Берутся все подматрицы игрового поля размера 3х3.
  2. Каждая матрица поворачивается на 90 градусов до тех пор, пока она не примет лексикографически минимальное состояние.
  3. Для всех матриц берётся вероятность успеха при открытии любой из клеток, принадлежащих ей.
  4. Из всех выбранных вариантов выбирается тот, у которого вероятность успеха максимальна.
  5. Открывается выбранная клетка.
  6. Вес выбранного хода на подматрице, которая ему принадлежит, изменяется в зависимости от успеха сделанного хода.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы был создан программный продукт, удовлетворяющий всем поставленным задачам. Получены навыки разработки программного обеспечения с использованием искусственного интеллекта. В результате использования языка Java данный программный продукт легко расширяем и модифицируем.

СПИСОК ЛИТЕРАТУРЫ

* 1. Левитин А.В. Алгоритмы: введение в разработку и анализ. М.: Вильямс, 2006. 576 с.
  2. Волков Е.А. Численные методы. М.: Наука, 1987. 248 с.
  3. Брауде Э. Дж. Технология разработки программного обеспечения. - СПб.: Питер, 2004. - 655с.

ПРИЛОЖЕНИЕ 1

Исходный код программы

**package** rustam.minesweeper.ai.impl.qlearning;  
  
**import** lombok.Value;  
**import** rustam.minesweeper.ai.OpenChoices;  
**import** rustam.minesweeper.commons.CellPosition;  
**import** rustam.minesweeper.commons.GameField;  
**import** rustam.minesweeper.commons.GameStatus;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
**import** java.util.function.Supplier;  
  
**import static** java.util.Collections.*singletonList*;  
  
**public class** QLearningAIPlayer **extends** QLearningAIPlayerBase {  
  
 **private final** Map<State, Map<CellPosition, Double>> **chance** = **new** HashMap<>(); *//[state, at] = chance* **public** QLearningAIPlayer(**int** squareSide) {  
 **super**(squareSide);  
 }  
  
 @Override  
 **public** OpenChoices selectOpenChoices() {  
 **return** selectOpenChoiceWithCenterCell().toOpenChoices();  
 }  
  
 **private** OpenChoiceInternal selectOpenChoiceWithCenterCell() {  
 State bestState = **null**;  
 CellPosition bestUpLeftPosition = **null**;  
 CellPosition bestChoosePosition = **null**;  
 **double** bestSuccessProb = -1;  
 **for** (**int** row = 0; row + **squareSide** <= **gameField**.getFieldSize().getRowCount(); row++) {  
 **for** (**int** column = 0; column + **squareSide** <= **gameField**.getFieldSize().getColumnCount(); column++) {  
 State curState = StateFactory.*createState*(**gameField**, **new** CellPosition(row, column), **squareSide**);  
 **for** (Map.Entry<CellPosition, Double> e : getTurns(curState).entrySet()) {  
 CellPosition position = e.getKey();  
 Double successProb = e.getValue();  
 **if** (successProb > bestSuccessProb && curState.getCells()[position.getRow()][position.getColumn()] == State.***UNKNOWN\_CELL***) {  
 bestState = curState;  
 bestUpLeftPosition = **new** CellPosition(row, column);  
 bestChoosePosition = position;  
 bestSuccessProb = successProb;  
 }  
 }  
 }  
 }  
 CellEntity[][] cellsAround = StateFactory.*getSortedCellsAround*(**gameField**, bestUpLeftPosition, **squareSide**);  
 **return new** OpenChoiceInternal(bestUpLeftPosition, bestChoosePosition, cellsAround, bestSuccessProb, bestState);  
 }  
  
 **private** Map<CellPosition, Double> getTurns(State state) {  
 **if** (!**chance**.containsKey(state)) {  
 Map<CellPosition, Double> turns = **new** HashMap<>();  
 **for** (**int** i = 0; i < **squareSide**; i++)  
 **for** (**int** j = 0; j < **squareSide**; j++)  
 turns.put(**new** CellPosition(i, j), 20.0);  
 **chance**.put(state, turns);  
 }  
 **return chance**.get(state);  
 }  
  
 **private void** updateSuccessProb(OpenChoiceInternal openChoice, **double** k, **boolean** success) {  
 **boolean** allUnknown = **true**;  
 **for** (CellEntity[] row : openChoice.**cellsAround**) {  
 **for** (CellEntity pos : row) {  
 allUnknown &= pos.getValue() == State.***UNKNOWN\_CELL***;  
 }  
 }  
 **if** (allUnknown) {  
 **for** (**int** i = 0; i < **squareSide**; i++)  
 **for** (**int** j = 0; j < **squareSide**; j++)  
 **chance**.get(openChoice.**state**).put(**new** CellPosition(i, j), 0.0);  
 **return**;  
 }  
  
 **double** res = success ? 1 : 0;  
 **chance**.get(openChoice.**state**).merge(openChoice.**chosenPosition**, res, (prev, cur) -> prev \* k + cur \* (1 - k));  
 }  
  
 **public void** warmUp(**int** count, Supplier<GameField> gameFieldFactory) {  
 **for** (**int** i = 0; i < count; i++) {  
 **if** (**gameField** == **null** || **gameField**.isGameFinished())  
 setGameField(gameFieldFactory.get());  
 OpenChoiceInternal openChoiceWithCenterCell = selectOpenChoiceWithCenterCell();  
 OpenChoices openChoices = openChoiceWithCenterCell.toOpenChoices();  
 CellPosition chosenCell = openChoices.getPositions().get(0);  
 **gameField**.open(chosenCell);  
 **double** k = Math.*exp*(-(i + 1) / 1000.0);  
 GameStatus gameStatus = **gameField**.getGameStatus();  
 updateSuccessProb(openChoiceWithCenterCell, k, gameStatus != GameStatus.***LOST***);  
 System.***err***.println(**"WARMUP "** + i);  
 }  
 }  
  
 @Value  
 **private static class** OpenChoiceInternal {  
 CellPosition **upLeft**;  
 CellPosition **chosenPosition**;  
 CellEntity[][] **cellsAround**;  
 **double successProb**;  
 State **state**;  
  
 OpenChoices toOpenChoices() {  
 CellPosition pos = **cellsAround**[**chosenPosition**.getRow()][**chosenPosition**.getColumn()].getPosition();  
 **return new** OpenChoices(**successProb**, *singletonList*(pos));  
 }  
 }  
}

ПРИЛОЖЕНИЕ 2

Описание работы программы

В программе нужно выбрать размер игрового поля и желаемое количество мин, затем можно создать случайное поле и начать игру.

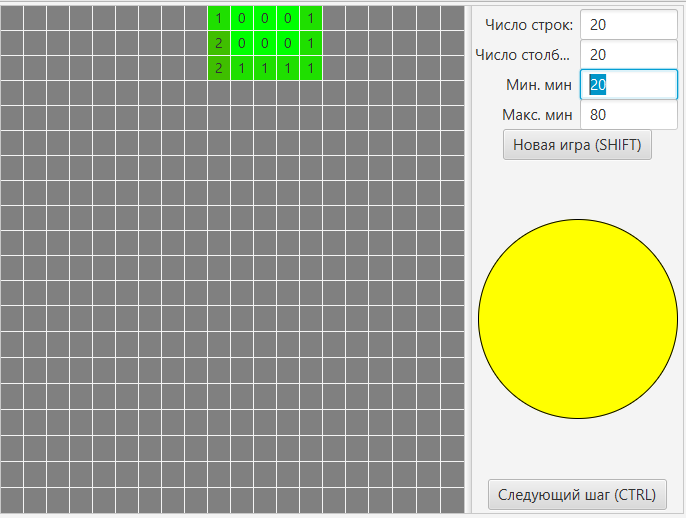


Рисунок П2.1. Окно начала игры

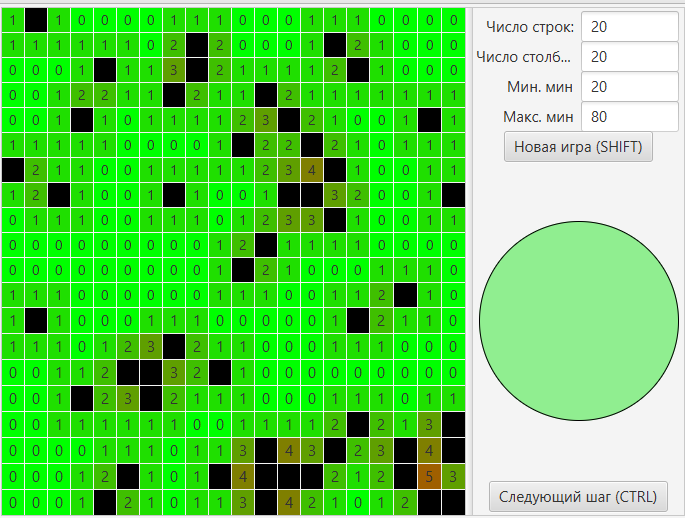


Рисунок П2.1. Выигранная игра