Модульна контрольна робота №1
Черпака Андрія, ІП-01
Варіант 28

1.1. Над даними можна здійснювати багато дій. В рамках курсу “Програмування інтелектуальних інформаційних систем” ми розглядали такі з них:

- консолідація - об’єднання даних з кількох джерел в одне
- профайлинг - процес вивчення, аналізу та створення корисних підсумків даних
- трансформація - перетворення даних одного формату в інший
- очищення - виявлення та виправлення або видалення даних, що є пошкодженими, неповними чи неточними або ще з якоїсь причини не підлягають аналізу
- оптимізація - сукупність дій, спрямованих на пришвидшення та збільшення ефективності процесів отримання, аналізу або вилучення даних
- збагачення - доповнення даних з використанням інших джерел

.1.2. Класифікація інформаційних систем:

- фактографічні, документальні, геоінформаційні;
- за функціональною ознакою: автоматизовані системи, системи підтримки прийняття рішень (інтелектуальні системи), інформаційно-обчислювальні системи, інформаційно-довідкові (інформаційно-пошукові), системи навчання;
- за рівнями управління: системи оперативного рівня, системи спеціалістів, системи тактичного рівня, стратегічні системи;
- за ступенем автоматизації: ручні, автоматичні, автоматизовані;
- за сферою застосування: організаційного управління, управління технологічними процесами, автоматизованого проєктування, інтегровані;
- за способом організації

1.3. Інтелектуальна інформаційна система – це взаємопов’язана сукупність засобів, методів та персоналу, що використовується для зберігання, оброблення та видачі
інформації з метою здійснення підтримки діяльності людини і пошуку інформації в режимі розширеного діалогу на природній мові. Їх також називають системами, заснованими на знаннях.

2.1. Різниця між мінімаксом і A*:
Алгоритм мінімакс зазвичай використовується при розробці ігор та призначений для прийняття рішення в умовах протистояння з одним або кількома суперниками. Він передбачає побудову дерева рішень певної глибини. Вузлами дерева є стани системи, для кожного вузла безпосередніми нащадками є такі стани, яких ми можемо досягнути за

один хід. Якщо на парній глибині хід здійснюється гравцем, то на непарній - суперником, і навпаки. Вважається, що як гравець, так і суперник обирають найвигідніші з доступних варіантів на момент ходу. Звісно, якість прийнятого рішення залежатиме від глибини дерева, але оскільки кількість вузлів залежить від глибини логарифмічно, її варто обмежувати до оптимального рівня.

A* теж найчастіше використовується в геймдеві, але призначений для пошуку оптимального рішення без урахування ходів суперника. Зазвичай використовується для пошуку найкоротшого шляху, але може бути використаний для вирішення великої кількості задач. Наприклад, для розв'язання 8-puzzle. По суті, являє собою пошук у ширину, єдиною відмінністю є те, що на порядок обробки вершин впливає також значення певної евристичної функції. Для пошуку шляху це очікувана відстань до фінішу, а для 8-puzzle - сума відстаней кожної клітинки до свого цільового положення. В ідеалі, алгоритм базується на використанні пріоритетної черги, хоча є й простіші реалізації без використання додаткових структур. Цей алгоритм вважається оптимальним, оскільки працює значно швидше за пошук у ширину і майже для всіх задач знаходить найкращий розв'язок.

Код для A*:

```
public override bool FindPath()
{
    PriorityQueue<Cell, int> queue = new PriorityQueue<Cell, int>();
    Cell current = Field[startX, StartY];
    current.MinDistanceFromStart = 0;
    queue.Enqueue(current, GetPriority(current));
    while (queue.Count>0)
    {
        current = queue.Dequeue();
        if (current.IsPassed) continue;
        current.MakePassed();
        if ((current.Y, current.X) == Field.EndPoint)
        {
            Field.MarkAsSolved();
            return true;
        }
        Cell[] adjacent = GetAdjacentCells(current);
        for (int i = 0; i < adjacent.Length; i++)
        {
            int newAdjacentMinDistance =
                current.MinDistanceFromStart + ((adjacent[i].X == current.X ||
adjacent[i].Y == current.Y) ? 10 : 14);
            if (newAdjacentMinDistance < adjacent[i].MinDistanceFromStart)
            {
                adjacent[i].MinDistanceFromStart = newAdjacentMinDistance;
                adjacent[i].PreviousCellYXCoordinates = (current.Y, current.X);
                queue.Enqueue(adjacent[i], GetPriority(current));
            }
        }
    }
    Field.MarkAsSolved();
    return false;
}
```

```
private int GetPriority(Cell cell)
{
    return (int)(cell.MinDistanceFromStart +
Math.Sqrt(Math.Pow((Field.EndPoint.Item1 - cell.Y) * 10, 2) +

Math.Pow((Field.EndPoint.Item2 - cell.X) * 10, 2)));
}


public RouteModel TraceRoute()
{
    if (!Field.IsSolved) throw new Exception("Impossible to trace route in not
solved field!");
    Stack<Cell> reversedRoute = new Stack<Cell>();
    Cell current;
    (short previousY, short previousX) = Field.EndPoint;
    do
    {
        current = Field[previousY, previousX];
        reversedRoute.Push(current);
        (previousY, previousX) = current.PreviousCellYXCoordinates;
    } while ((previousY, previousX) != (-1, -1));

    (short, short)[] route = new (short, short)[reversedRoute.Count];
    for (short counter = 0; counter<route.Length; counter++)
    {
        current = reversedRoute.Pop();
        route[counter] = (current.Y, current.X);
    }

    return new RouteModel(route, current.MinDistanceFromStart,
        (route[0] == Field.StartPoint && route[^1] == Field.EndPoint));
}
```

Код для Mini-Max:

```
private int MiniMax(Node node, int depth)
{
    if (depth == 0 || node.IsTerminal)
    {
        ratePosition(node);
    }
    else
{
    foreach (Node child in node.PossibleNextMoves)
    {
        int nVal = AlphaBeta(child, depth - 1, α, β);
        if (nVal > node.Value && node.DepthLevel % 2 == 1 || node.Value > nVal)
node.Value = nVal;
    }
}

return node.Value;
}
```