사본

『제곱평균제곱근을 활용한 컴퓨터와의 숫자야구 대결 게임 개발』

2학년 의반 18번 박현우

◎ 초록(Abstract) :

알파고와 이세돌의 대결 이후로 국내에서 인공지능에 대한 관심은 최고조인 상태다. 이에 내가 속해 있는 프로그래밍 동아리 'SPACE'에서는 작년 청심축전 동아리 부스로 '인공지능과의 숫자야구 대결'이라는 콘셉트로 게임 소프트웨어를 제작해 운영하였다. 본 탐구에서는 이때의 소프트웨어에 사용한 알고리즘을 개선하여, 보다 플레이어의 숫자를 빨리 맞추는 훨씬 더 난도 높은 숫자야구 게임 소프트웨어를 제작하기 위한 연구를 진행하였고, 그 결과 제곱평균제곱근의 원리를 이용한 알고리즘을 설계해 소프트웨어에 적용하여 이를 성공하였다.

◎ 목 차(Index) :

위 사본은 원본과 상이없음을 증명함
2018. 09.
첫 원 고 등 학 교 장

I. 머리말

1. 탐구 목적 및 문제제기

작년 청심축전 동아리 부스에서 선보인 게임 소프트웨어는 참여한 학생들로부터 80%가 넘는 승률을 기록하였다. 하지만 이 때 대부분이 숫자야구를 접해보지 못한 초보자였다는 점, 그리고 무엇보다 사용한 컴퓨터의 질문 알고리즘이 가능한 모든 숫자 중에서 랜덤으로 뽑아서 질문하는, 상당히 무식한 방식이었다는 것을 감안하면 컴퓨터의 실제 실력은 위 통계보다 낮을 것으로 보인다. 실제로 기존에 숫자야구를 많이 해본 실력 있는 학생들로 범위를 한정하면 컴퓨터는 절반도 승리하지 못하였다. 따라서 본 탐구에서는 제곱평균제곱근이라는 수학적 개념을 활용해 새로운 컴퓨터의 질문 알고리즘을 구상하고, C#/WPF로 숫자야구 게임 소프트웨어를 다시 제작해 보고자 한다.

2. 탐구의 필요성과 연계성

논리·추리게임의 일종인 숫자야구를 하면서 플레이어는 논리력과 사고력을 비롯한 다양한 능력을 키울 수 있을 것이다. 본 탐구를 통해 제작한 소프트웨어를 활용하면 혼자 있을 때도 2명이서 하는 것처럼 컴퓨터와 양방향으로 숫자야구 게임을 할 수 있을 것이며 2명이서 하는 것처럼 논리력과 추리력을 키울 수 있다. 하지만 이 때 컴퓨터의 실력이 플레이어에 비해 현저히 떨어진다면 재미가 반감될 것이다. 따라서 컴퓨터는 플레이어의 재미를 충족시켜줄 수 있는 실력을 갖추어야 하며 본 탐구를 통해 이를 해결할 수 있으리라 기대된다.

3. 선행연구에 대한 분석 및 연구의 방향 제시

인터넷에서 사전 조사를 해본 결과 컴퓨터의 효율적인 질문 알고리즘은 고사하고 컴퓨터가 플레이어에게 질문하는 알고리즘 자체에 대한 내용이 크게 부족했고, 대부분 플레이어가 일방적으로 컴퓨터의 숫자를 추리해내는 단방향 게임 알고리즘에 관한 자료만 있었다. 실제로 작년에도 대부분의 핵심 알고리즘을 내가 직접 설계했어야 했다. 따라서 제곱평균제곱근을 활용하여 컴퓨터가 플레이어에게 효율적으로 질문을 하는 알고리즘을 구현하고 이를 이용하여 보다 뛰어난 실력의 컴퓨터와 번갈아 질문하는 양방향 게임을 프로그래밍으로 구현하는 것에 관한 연구를 중점적으로 하도록 한다.

II. 이론적 배경

1. 숫자야구

가. 개요

먼저 게임에 사용할 숫자의 길이를 3/4자리 중에서 정한다. 각자 3/4자리의 숫자를 임의로 정한 뒤, 서로에게 자신이 추측하는 상대방의 숫자를 질문해서 결과를 확인한다. 그리고 그 결과를 토대로 상대가 생각하는 숫자를 예상한 뒤 맞히는 게임이다.

나. 게임 방식

사용되는 숫자는 0에서 9까지이며 숫자는 맞지만 위치가 틀렸을 때는 볼, 숫자와 위치가 전부 맞으면 스트라이크, 숫자와 위치가 전부 틀리면 아웃이다. 또 무엇이 볼이고 스트라이크인지는 알려주지 않으며 숫자는 중복해서 사용할 수 없다.

다. 예시

상대가 생각한 숫자가 320이라 할 때,

횟수	숫자			판정
1	6	1	4	OUT
2	5	7	3	1B
3	8	3	0	1S 1B
4	3	0	2	1S 2B
5	3	2	0	3S

- 614 - 들어맞는 숫자가 전혀 없으므로 아웃. 말 그대로 1, 4, 6을 예상 숫자에서 아예 "아웃"시킬 수 있다. 즉 현재 예상숫자는 0, 2, 3, 5, 7, 8, 9.
- 573 - 3이 있지만 위치가 다르므로 1볼. 물론 게임상으론 어떤 숫자가 맞는지 알 수 없기에 가장 난감하다.

- 830 - 0이 있고 위치가 맞으며, 3이 있지만 위치가 다르므로 1스트라이크 1볼.
- 302 - 숫자는 전부 맞지만 위치는 3만 맞고 나머지 둘은 다르므로 1스트라이크 2볼.
- 320 - 전부 맞으므로 승리. [1]

2. 제곱평균제곱근

가. 개요

수학에서 제곱평균제곱근, 혹은 이차평균은 변화하는 값의 크기에 대한 통계적 척도로, 명칭 그대로 값들의 제곱에 대한 평균의 제곱근이다.

나. 정의

일련의 값에 대한 제곱평균제곱근은 원래의 값의 제곱들에 대한 산술평균의 제곱근으로, n개의 값들 $\{x_1, x_2, \ldots, x_n\}$에 대한 제곱평균제곱근은 다음과 같이 주어진다. [2]

$$x_{\mathrm{s}} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}$$

Ⅲ. 연구방법 및 주요 내용

1. 질문 알고리즘 구상

1) 게임 클래스 설계 및 구현

먼저 숫자야구 게임을 관리할 Game 클래스를 만들고자 한다. 생성자에서는 자릿수를 입력받아 자릿수에 따라 가능한 모든 경우의 수와 대답을 구해 리스트와 배열에 넣는다. 클래스는 가능한 경우의 수가 1 이하일 때 참을 반환하는 IsFinished 메서드, 게임의 진행단계를 반환하는 GetStep 메서드, 컴퓨터가 플레이어에게 할 질문을 정해서 반환하는 GetQuestion 메서드(이 메서드에 어떤 알고리즘을 사용하느냐에 따라 컴퓨터의 실력이 결정된다.), 질문과 숫자를 입력받아 올바른 대답을 반환하는 GetAnswer 메서드, 질문에 대한 플레이어의 대답을 입력하는 PutAnswer 메서드, 플레이어의 대답에 해당하는 경우의 수가 존재하는지 여부를 반환하는 IsCorrect 메서드로 이루어지는 구조로 설계하였다. 다음은 작성한 Game 클래스의 코드이다.

```
public class Game
{
    public enum Digit
    {
        Three = 3,
        Four = 4,
    }

    public int Length { get; }
    public List<string> AllowedNumbers {
get; }

    List<History> _history = new
List<History>();

    int _step;
    string _lastQuestion;

    readonly Answer[]
POTENTIAL_ANSWERS;

    public Game() : this(Digit.Four) { }
    public Game(Digit digit)
    {
        Length = (int)digit;
        AllowedNumbers =
Permutation.Create("1234567890",
Length).ToList();

        switch (digit)
        {
            case Digit.Three:
                POTENTIAL_ANSWERS =
                    (생략)
                break;
            case Digit.Four:
                POTENTIAL_ANSWERS =
                    (생략)
                break;
        }
    }

    public bool IsFinished()
```

```
    {
        return AllowedNumbers.Count <= 1;
    }

    public int GetStep()
    {
        if (IsFinished())
        {
            if (_history.Last().Answer.Bulls
== Length)
                return _step;
            return _step + 1;
        }
        return _step;
    }

    public string GetQuestion()
    {
                (질문 알고리즘)
    }

    public Answer GetAnswer(string
number, string question)
    {
        int bulls = 0, cows = 0;

        for (int i = 0; i < number.Length;
i++)
        {
            if (number[i] == question[i])
                bulls++;
            else if
(number.Contains(question[i]))
                cows++;
        }

        return new Answer(bulls, cows);
    }

    public void PutAnswer(Answer answer)
    {
        History history = new
History(_lastQuestion, answer);
        _history.Add(history);
        AllowedNumbers.RemoveAll(e =>
!IsConsistent(e, history));
    }

    public bool IsConsistent(string number,
History history)
    {
        Answer answer =
GetAnswer(number, history.Question);
        return answer.Bulls ==
history.Answer.Bulls && answer.Cows ==
history.Answer.Cows;
    }

    public void PutAnswer(int bulls, int
cows)
    {
        PutAnswer(new Answer(bulls,
cows));
    }

    public bool IsCorrect(out string result)
    {
        result =
AllowedNumbers.FirstOrDefault(e =>
IsConsistent(e, _history.Last()));
        return result != null;
    }
}
```

코드 1-1. Game.cs

2) 알고리즘 구상 및 구현

이제 본격적으로 GetQuestion 메서드에 들어갈 컴퓨터의 질문 알고리즘을 구현하고자 한다. 사전에 여러 알고리즘을 구상 및 구현하고 시행착오를 겪어본 결과 앞서 말했던 제곱평균제곱근을 활용한 알고리즘을 사용하기로 결론을 내렸다. 알고리즘에 대해 설명하자면 다음과 같다.

먼저 가능한 모든 플레이어의 대답에 반복문을 돌려 현재의 원소가 질문에 대한 플레이어의 대답이라고 가정한다. 그리고 플레이어의 숫자로 가능한 모든 수에 대해 차례차례 시뮬레이션을 해서 질문과 대답에 대해 가능한 경우의 수를 구한다. 이 결과, 즉 질문을 했을 때 남는 경우의 수의 평

균이 최소가 되는 질문이 평균적인 상황에서 경우의 수를 한 번에 제일 크게 줄일 수 있는 가장 효율적인 질문이라고 할 수 있다.

하지만 일반적인 산술평균으로는 이를 비교할 수 없다. 어떤 질문이든 대답에 대한 경우의 수를 모두 더한 것은 전체 경우의 수로 일정하기 때문이다. 여기서 제곱평균제곱근이 사용된다. 경우의 수의 합이 아닌 제곱의 합을 비교함으로써 컴퓨터는 가장 효율적인 질문을 찾아낼 수 있다. 다음은 이를 바탕으로 구현한 GetQuestion 메소드다.

```
if (_step++ == 0)
    return _lastQuestion =
AllowedNumbers.RandomValue();
return _lastQuestion =
AllowedNumbers.Shuffle().AsParallel().MinBy(
question =>
POTENTIAL_ANSWERS.AsParallel().Sum(answ
er =>
Math.Pow(AllowedNumbers.AsParallel().Count
(e => IsConsistent(e, new History(question,
answer))), 2)));
```

코드 1-2. Game.cs -GetQuestion-

코드를 설명하면 다음과 같다. 먼저 제일 처음 질문일 때는 가능한 모든 경우 중에서 랜덤으로 하나를 질문으로 택한다. 하지만 그 다음부터는 가능한 모든 경우의 수를 질문으로 가정하고 모든 대답을 그에 대한 대답으로 가정하여 반복해서 시뮬레이션을 돌린다. 마지막으로 시뮬레이션 결과의 제곱의 합이 최소가 되는 질문을 선택한다.

2. 게임 구현 및 테스트

1) 게임 소프트웨어 구현

이제 구현한 Game 클래스를 이용하여 플레이어와 컴퓨터가 대결하는 게임을 만들고자 한다. 먼저 랜덤 클래스를 이용해서 턴을 정해 변수에 저장한다. 플레이어의 질문 메서드와 대답 메서드를 작성하고, 턴 변수의 값에 따라 실행 여부를 정한다. 플레이어의 턴일 때 플레이어가 질문을 입력하면 질문 메서드가 실행되고 컴퓨터의 숫자에 대한 컴퓨터의 대답이 화면에 뜬다. 이 후 컴퓨터의 턴으로 자동 전환되며, 컴퓨터는 앞서 구현한 GetQuestion 메서드의 결과를 질문으로 플레이어에게 제시한다. 플레이어가 이에 대한 대답을 입력하면 대답 메서드가 실행되어 컴퓨터에게 전달되고 다시 플레이어의 턴으로 자동 전환된다. 이를 게임이 끝날 때까지 반복한다. 다음은 이를 구현한 코드의 일부이다.

```
private async void GuessClick()
{
    if (await
_window.GetCurrentDialogAsync<BaseMetroD
ialog>() != null)
        return;

    if (Turn == GameTurn.Player &&
Guess?.Length == _game.Length &&
Guess.All(e => "1234567890".Contains(e)))
    {
        PlayerLog.Add(new History(Guess,
_game.GetAnswer(Number, Guess)));

        if (Number == Guess)
        {
                (생략)
            Message = "앗, 제 암호를
맞추셨군요...\n당신의 암호는 무엇인가요?";

            string number = await
_window.ShowInputAsync("앗, 제 암호를
맞추셨군요...", "당신의 암호는 무엇인가요?");
            CheckConsistent(number);
        }
        else
        {
            Turn = GameTurn.Computer;
        }

        Guess = "";
    }
}

private async void AnswerClick()
```

```
{
    if (Turn == GameTurn.Computer &&
SelectedStrike + SelectedBall <=
_game?.Length)
    {
        int count =
_game.AllowedNumbers.Count;

        ComputerLog.Add(new
History(_question, new
Answer(SelectedStrike, SelectedBall)));
        _game.PutAnswer(SelectedStrike,
SelectedBall);

        if (SelectedStrike == _game.Length)
        {
                (생략)
            Message = string.Format("제가
이겼습니다!\n저의 암호는 {0}입니다.",
Number);
            Repeat();
            return;
        }

        if (_game.AllowedNumbers.Count ==
0)
        {
                (생략)
            Message = "조건에 맞는 암호가
존재하지 않습니다.\n당신이 생각한 암호는
무엇인가요?";

            string number = await
_window.ShowInputAsync("조건에 맞는 암호가
존재하지 않습니다.", "당신이 생각한 암호는
무엇인가요?");
            CheckConsistent(number);
            return;
        }

        double reduction = 1 -
_game.AllowedNumbers.Count /
(double)count;
                (생략)
        Message = "그렇군요, 알겠습니다.";
        Turn = GameTurn.Player;
    }
}
```

코드 2-1. MainWindowViewModel.cs

2) 알고리즘 성능 테스트

이제 구현한 알고리즘의 성능을 테스트해보고자 한다. 새로 콘솔 프로젝트를 만들어서 반복문을 돌려 랜덤으로 숫자를 정하고 컴퓨터가 몇 번의 질문으로 이를 맞추는지 확인하는 것을 반복하여 통계를 낸다. 다음은 이를 구현한 테스트 코드의 일부이다.

```
int stepSum = 0;
int stepMin = int.MaxValue, stepMax =
int.MinValue;
int[] distribution = new int[9];

string[] numbers =
Permutation.Create("1234567890",
4).Shuffle().ToArray();

for (int n = 0; n < numbers.Length; n++)
{
    string number = numbers[n];

    game = new Game(Game.Digit.Four);

    while (!game.IsFinished())

game.PutAnswer(game.GetAnswer(number,
game.GetQuestion()));

    string result;

    if (!game.IsCorrect(out result))
    {
        Console.WriteLine("Error");
        Console.ReadLine();
    }

    int step = game.GetStep();
```

```
        stepSum += step;
        stepMin = Math.Min(stepMin, step);
        stepMax = Math.Max(stepMax, step);
        distribution[step]++;

        Console.WriteLine("#{0} num: {1} avg: {2}
best: {3} worst: {4}", n + 1, number,
stepSum / (double)(n + 1), stepMin,
stepMax);
}

for (int i = 1; i <= 9; i++)
    Console.WriteLine(distribution[i]);
```

코드 2-2. Program.cs

Ⅳ. 맺음 말

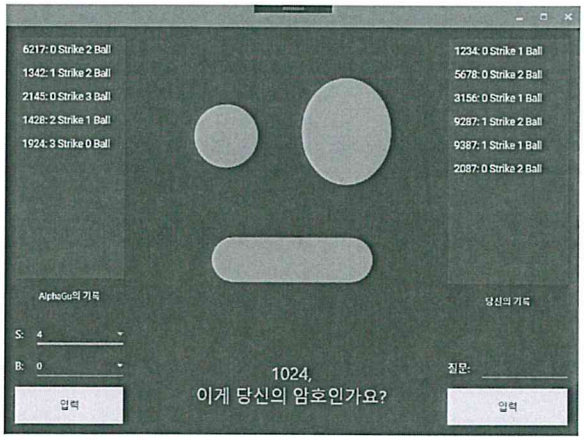

그림 1. 최종적으로 완성한 게임 소프트웨어의 화면

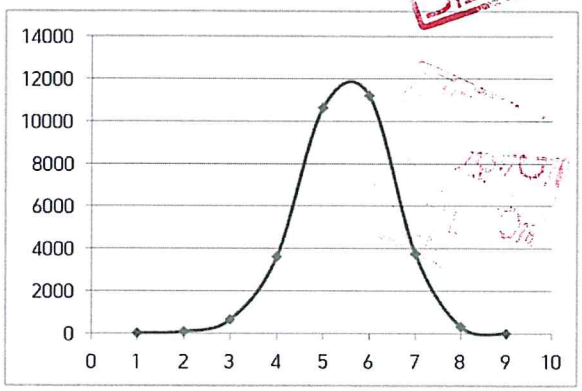

차트 1. 랜덤 알고리즘 질문 횟수 분포

1	7	0.02%
2	79	0.26%
3	642	2.12%
4	3608	11.93%
5	10634	35.17%
6	11208	37.06%
7	3753	12.41%
8	306	1.01%
9	3	0.01%
평균	5.479	

표 1. 랜덤 알고리즘 질문 횟수 분포

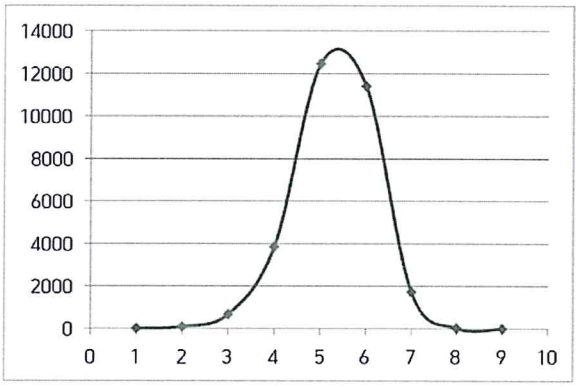

차트 2. 제곱평균제곱근 알고리즘 질문 횟수 분포

1	7	0.02%
2	88	0.29%
3	671	2.22%
4	3853	12.74%
5	12457	41.19%
6	11407	37.72%
7	1732	5.73%
8	25	0.08%
9	0	0.00%
평균	5.313	

표 2. 제곱평균제곱근 알고리즘 질문 횟수 분포

보다시피 랜덤으로 질문을 선택한 이전과 달리 제곱평균제곱근을 사용해서 질문을 선택한 게임에서 컴퓨터가 플레이어의 숫자를 맞추는데 걸린 질문 횟수가 현저히 적다는 것을 알 수 있고, 랜덤으로 질문한 1번을 제외한 2-6번의 질문 횟수는 증가한 반면 7-9번의 횟수는 크게 감소했으며 심지어 9번 질문한 경우는 아예 사라진 것을 볼 수 있다. 실제로 새 알고리즘을 적용한 게임을 플레이해보니 이전보다 컴퓨터의 실력이 훨씬 높아진 것이 느껴질 정도였다.

결과만 보면 알고리즘 구상이 쉬워보일지 몰라도 사실 앞서 설명했다시피 엄청난 노력과 시행착오를 거쳐 수많은 알고리즘을 고안하고 폐기했다. 뛰어난 성능의 알고리즘은 대부분 탐색을 많이 하기 때문에 시간복잡도가 커 정상적인 게임 진행이 불가능했고, 시간복잡도가 작은 것은 랜덤 알고리즘만도 못하는 성능을 보였다. 이 시간복잡도가 적당하면서도 좋은 성능을 보이는 알고리즘을 구상해내는 쉽지 않았다. 작년에 동아리 부스에서 한계점을 느낀 이후, 1년 동안 인터넷을 뒤져가며 연구했고 또 연구했다.

하지만 정작 만족스러운 해답은 인터넷이 아니라, 소프트웨어 개발에 별로 도움도 되지 않는다고 생각한 학교 수업에 있었다. 어느 날 물리2 수업을 듣던 중 선생님께서 기체 분자의 평균 속력을 구하는데 사용된다며 제곱평균제곱근에 대해 설명해주셨다. 이를 듣는 순간 머릿속에 아이디어가 떠올랐다. 경우의 수의 제곱평균제곱근을 비교하면 내가 원하는 알고리즘이 나올지도 모른다고 말이다.

지금까지 난 컴퓨터 외적의 지식은 소프트웨어 개발에 직접적으로는 거의 도움이 안 된다고 생각하였다. 간접적으로 소프트웨어 아이디어를 내는 데는 유용할지 몰라도 개발에서 부닥치는 어려움을 해결하는 데는 쓸모가 없다고 여긴 것이다. 하지만 이번 탐구를 통해서 컴퓨터 외적인 지식들도 개발에서 문제를 해결하는데 유용하게 쓰일 수 있다는 것을 느꼈고, 앞으로는 컴퓨터 외적인 지식들도 무시하지 말고 다양한 학문을 접해봐야겠다고 다짐하게 되었다.

본 보고서에는 소프트웨어의 소스코드 전부가 아닌 핵심적인 기능을 담당하는 일부만을 담았다. 대신 깃허브에 프로젝트의 소스코드를 전부 업로드 하였으니, https://github.com/Winrobrine/AlphaGu2에 들어가면 확인할 수 있다.

< 참고 문헌 >

[1] 나무위키. “숫자야구”
https://namu.wiki/w/%EC%88%AB%EC%9E%90%EC%95%BC%EA%B5%AC
[2] Wikipedia. “제곱평균제곱근”
https://ko.wikipedia.org/wiki/%EC%A0%9C%EA%B3%B1%ED%8F%89%EA%B7%A0%EC%A0%9C%EA%B3%B1%EA%B7%BC