소프트웨어실습3

HW3

2013726003

컴퓨터소프트웨어학과

임우재

받을 수 있겠다고

생각하는 점수 : 10

구현한 기능

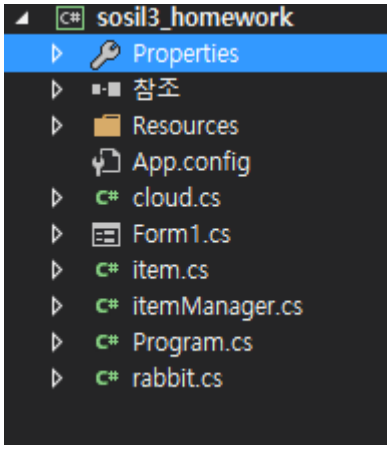
1. 기본 UI + 매뉴얼 1
2. 토끼가 좌우로 움직이는 키보드 이벤트 구현 1
3. 구름이 일정 범위를 랜덤하게 이동 1
4. 구름이 이동하는 도중 특정 확률로 물체를 떨어뜨림 1
5. 토끼가 물체를 먹는 기능 1.5
6. 레벨에 따른 난이도 차등 구현 1.5
7. 모든 것을 구현하고 예시 프로그램처럼 돌아감 1
8. 하이 스코어를 보여주는 기능(파일을 통해 구현) 1
9. 유저가 쓸만하다고 생각하는 추가 요소 1

    9-1. 점프 기능 추가

    9-2. 점프와 무적 모드 사용시 사운드 추가

    9-3 기타 효과음 추가

코드를 설명하기에 앞서 제가 짯던 구조 입니다.



각각 세분화 된 클래스로 하였고 각각에 대해 설명하면

Cloud 는 구름을 구현한 클래스입니다.

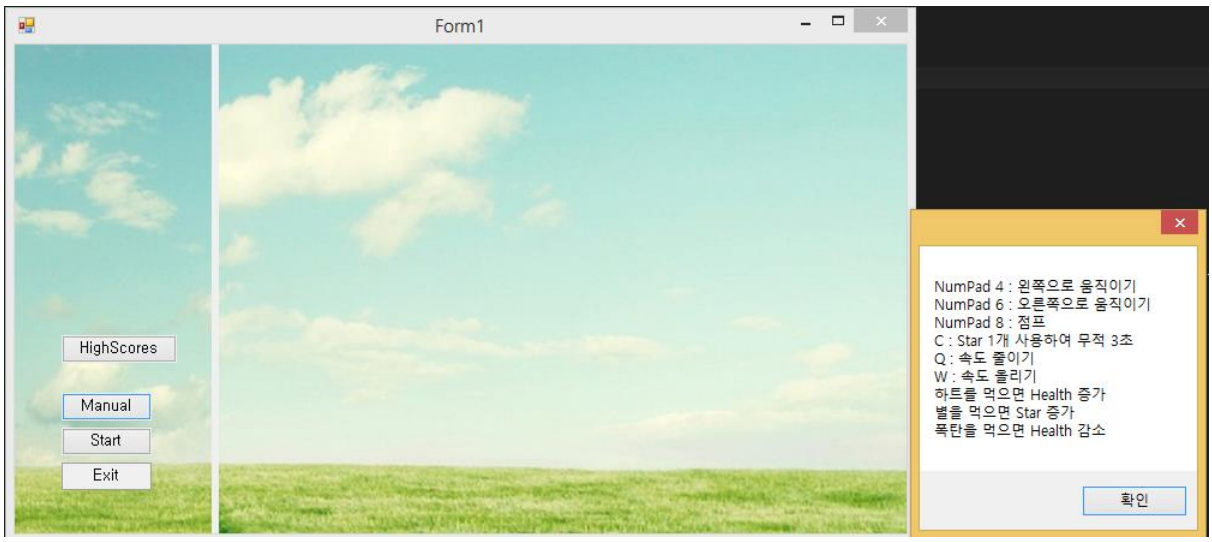
Item 클래스는 떨어지는 아이템들을 구현한 클래스입니다. 이 클래스들이 하나씩 생성되면 아이템이 하나씩 떨어지게 됩니다.

ItemManager 는 아이템의 management를 담당합니다. 시간 텀을 두고 아이템을 하나 생성하고 죽은 아이템들을 삭제해줍니다.

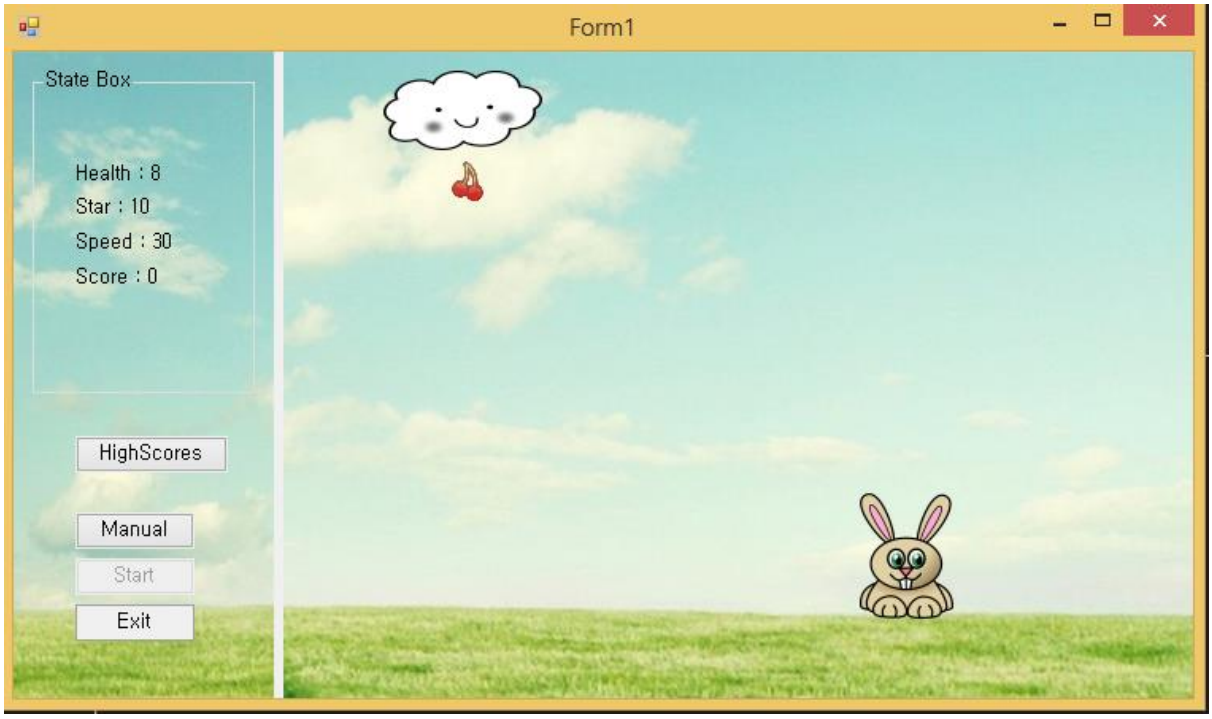
Rabbit 클래스는 주인공인 토끼를 구현한 클래스입니다.

1. 기본 UI + 매뉴얼

매뉴얼



Start 버튼을 누른 후



코드 설명

```
private void btn_Manual_Click(object sender, EventArgs e)
{
    Sound_Message.Play();
    MessageBox.Show("NumPad 4 : 왼쪽으로 움직이기₩n" + "NumPad 6 : 오른쪽으로 움직이기₩n" + "NumPad 8 : 점프₩n" + "C : Star 1개 사용하여 무적 3초₩n" + "Q : 속도 줄이기₩n" + "W : 속도
        + "하트를 먹으면 Health 증가₩n" + "별을 먹으면 Star 증가₩n" + "폭탄을 먹으면 Health 감소₩n");
}
```

```
groupBox1.Parent = pictureBox_left;

pictureBox_nomal_rabbit.BackColor = Color.Transparent;
pictureBox_nomal_rabbit.Parent = pictureBox_background;

pictureBox_cloud.BackColor = Color.Transparent;
pictureBox_cloud.Parent = pictureBox_background;
```

픽쳐박스와 그룹박스의 배경 색을 투명으로 하고 그들의 parent를 지정해 주었다.

2. 토끼의 키보드 입력 이벤트 구현

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    my_rabbit.move(e, pictureBox_background.Width, this);
}
```

Form1_keydown 이벤트가 발생하면 rabbit 클래스 안에 있는 move 메소드를 호출합니다.

```
if (e.KeyCode == Keys.NumPad4)
{
    (new Thread(() =>
    {
        form.Invoke(new MethodInvoker(delegate ()
        {
            location.X = location.X - speed;
            if (location.X <= 0)
            {
                location.X = 1;
            }
            myrabbit.SetBounds(location.X, location.Y, myrabbit.Width, myrabbit.Height);

        }));
    })).Start();

}
else if (e.KeyCode == Keys.NumPad6)
{
    (new Thread(() =>
     {
         form.Invoke(new MethodInvoker(delegate ()
         {
             location.X = location.X + speed;
             if (location.X >= Width_limit - 60)
             {
                 location.X = Width_limit - 60;
             }
             myrabbit.Location = location;

         }));

     })).Start();
```

좌우로 움직이는 부분입니다. 보시면 speed 값 만큼 움직이게 되어있고 양쪽 벽 이상 못 나가게 처리해두었습니다. 여기서 Thread를 람다식으로 생성한 이유는 ui 스레드에서 키보드 입력까지 처리하면 성능 저하를 불러올까 싶어서 각각의 경우,

ui 스레드가 처리하는 경우와 새로운 스레드를 만들어서 처리하는 경우를 비교해 봤는데

크게 차이는 나지 않았지만 이쪽이 살짝 나은거 같아서 이렇게 구현했습니다.

새로운 스레드를 만들어서 움직이게 할 경우 invoke를 호출해야 하는데 이 부분에서 상당히 골치가 아팠습니다. Invoke는 form에만 구현되어 있는데 다른 클래스에서 불러오는 방법이 조금 많이 어려웠습니다.

3. 구름이 일정하게 랜덤하게 범위를 이동

```
private void Cloud_move()
{
    int speed = 100;
    cloud.Wait(100);
    bool p = true;
    Random r = new Random();
    int move_range;
    while (true)
    {
        move_range = r.Next(20, 150);
        for (int i = 0; i < move_range; i++)
        {
            if (pictureBox_cloud.Location.X <= 2)
            {
                p = true;
            }
            else if (pictureBox_cloud.Location.X >= pictureBox_background.Width - pictureBox_cloud.Width - 2)
                p = false;
            this.Invoke(new MethodInvoker(delegate ()
            {
                IntPtr x;
                if (!pictureBox_cloud.IsHandleCreated)
                    x = pictureBox_cloud.Handle;
                pictureBox_cloud.Location = my_cloud.move(p);
            }));
            if (speed > 10) speed = 100 - (my_rabbit.score / 5);
            cloud.Wait(speed);
        }
        p = !p;
    }
}
```

Form1 안에 구현되어 있는 cloud_move() 메소드 입니다. 이 메소드는

```
Task cloud;
```

```
cloud = new Task(new Action(Cloud_move));
cloud.Start();
```

Ui 쓰레드의 과부하를 막기 위해서 task가 실행하도록 했습니다.

Move_range 만큼 이동하고 만약 구름의 위치가 벽 끝에 닿으면 p 값을 이용해서 반대로 가도록 설정하였습니다. 여기서 task가 wait 하는 시간이 길면 길수록 cloud는 천천히 움직이고 짧을수록 빨리 움직입니다.

4. 구름이 이동하는 도중 특정 확률로 물체를 떨어뜨림

```
}
private void makeitem(Form1 Form1, PictureBox background, PictureBox Cloud, PictureBox prabbit, rabbit rabbit, List<item> itemList, Label lab
{
    int a = 1200;
    Thread.Sleep(3000);
    while(true)
    {
        if(a > 500) a = 2000 - (rabbit.score/100) * 100;
        itemList.Add(new item(Form1, background, Cloud, prabbit, rabbit, label_health, label_Star, label_score));
        for(int i = itemList.Count - 1; i >= 0; i--)
        {
            if(itemList[i].state == false)
            itemList.RemoveAt(i);
        }
        Thread.Sleep(a);
    }
}
```

아이템매니저 안에 있는 makeitem 메소드 이다. 주기적으로 깨어나서 아이템을 추가한다.

```
Random r = new Random((int)DateTime.Now.Ticks);
int i = r.Next(0, 13 + (score / 300));
switch (i)
{
    case 0:
        return sosil3_homework.Properties.Resources.f_carrot;
    case 1:
        return sosil3_homework.Properties.Resources.f_apple;
    case 2:
        type = 2;
        return sosil3_homework.Properties.Resources.f_bomb;
    case 3:
        return sosil3_homework.Properties.Resources.f_cherry;
    case 4:
        return sosil3_homework.Properties.Resources.f_grapes;
    case 5:
        type = 1;
        return sosil3_homework.Properties.Resources.f_heart;
    case 6:
        return sosil3_homework.Properties.Resources.f_kiwi;
    case 7:
        return sosil3_homework.Properties.Resources.f_orange;
    case 8:
        return sosil3_homework.Properties.Resources.f_pear;
    case 9:
        return sosil3_homework.Properties.Resources.f_pineapple;
    case 10:
        type = 3;
        return sosil3_homework.Properties.Resources.f_star;
    case 11:
        return sosil3_homework.Properties.Resources.f_strawberry;
    case 12:
        return sosil3_homework.Properties.Resources.f_watermelon;
}
type = 2;
```

0부터 12까지가 기본 확률이고 난이도가 높아질수록 폭탄이 나올 확률이 증가한다.

```
for (int i = 0; i < 2 + a / 5; i++)
{
    location.Y += 3;
    my_item.SetBounds(location.X, location.Y, my_item.Width, my_item.Height);
}
```

```
            }
            if (pic_rabbit.Bounds.IntersectsWith(getMyBound()))
            {
                if (type == 1) myrabbit.health++;
                else if (type == 2 && myrabbit.over == false)
                {
                    myrabbit.health--;
                    if (myrabbit.health < 1)
                    {
                        Application.Restart();
                    }
                }
                else if (type == 3) myrabbit.star++;
                else myrabbit.score = myrabbit.score + 10;
                health_label.Text = "Health : " + myrabbit.health.ToString();
                Score_label.Text = "Score : " + myrabbit.score.ToString();
                Star_label.Text = "Star : " + myrabbit.star.ToString();
                state = false;
                my_item.Dispose();
            }
        }
        a++;
```

item 안에 있는 move 메소드 중 일부이다. a값이 가속도를 구현한 부분이다.

```
            if (location.Y > 375)
            {
                if (type == 0 && myrabbit.over == false)
                {
                    myrabbit.health--;
                    if (myrabbit.health < 1)
                    {
                        Application.Restart();
                    }
                }
                health_label.Text = "Health : " + myrabbit.health.ToString();
                state = false;
                if (my_item.IsDisposed == false)
                {
                    my_item.Dispose();

                }
            }
```

위치가 땅 아래 부분 이상을 넘어갈 경우 picturebox를 dispose 해주었다.

```
public item(Form1 Form1, PictureBox parent, PictureBox Cloud
{
    pic_rabbit = pictureboxrabbit;
    Bitmap my_image = getMyImage(rabbit.score);
    state = true;
    location = Cloud.Location;
    location.X = location.X + 35;
    location.Y = location.Y + 50;
```

Item 생성자의 일부이다. Location 을 구름의 location을 받아와서 계산한다.

5. 토끼가 물체를 먹는 기능 구현

사실 이 부분은 충돌 판정을 하는 manager를 구현하여 거기서 처리하려 했으나 그렇게 할 경우 리소스도 많이 사용하게 되고 굳이 그렇게 하지 않아도 되어서 item이 떨어지면서 판정하게 하였다.

아래 코드는 item move 안에 있는 코드이다. move 메소드는 계속 실행되므로 item각자 알아서 충돌판정을 하게 되는 것이다.

```
if (pic_rabbit.Bounds.IntersectsWith(getMyBound()))
{
    if (type == 1) myrabbit.health++;
    else if (type == 2 && myrabbit.over == false)
    {
        myrabbit.health--;
        if (myrabbit.health < 1)
        {
            Application.Restart();
        }
    }
    else if (type == 3) myrabbit.star++;
    else myrabbit.score = myrabbit.score + 10;
    health_label.Text = "Health : " + myrabbit.health.ToString();
    Score_label.Text = "Score : " + myrabbit.score.ToString();
    Star_label.Text = "Star : " + myrabbit.star.ToString();
    state = false;
    my_item.Dispose();
}
```

래빗 픽쳐박스의 바운드에 item 바운드가 들어가게 되면 아이템의 타입에 따라 효과가나뉘게 되고 그 결과 값이 바로 label에 반영되고 아이템의 picturebox 는 dispose 된다. 위에서

Type 2 는 폭탄인데 만약 내가 오버 모드가 아닐 시에만 health 값이 줄어들게 된다.

```
else if (e.KeyCode == Keys.E)
{
    if (over != true && star != 0)
    {
        (new Thread(() =>
        {
            star--;
            Sound_anger.Play();
            over = true;
            form.Invoke(new MethodInvoker(delegate ()
            {
                myrabbit.Image = sosi13_homework.Properties.Resources.angryrabbit;
                my_starlabel.Text = "Star : " + star.ToString();
            }));
            Thread.Sleep(3000);
            form.Invoke(new MethodInvoker(delegate ()
            {
                myrabbit.Image = sosi13_homework.Properties.Resources.rabbit;
                over = false;
            }));
        })).Start();
    }
}
```

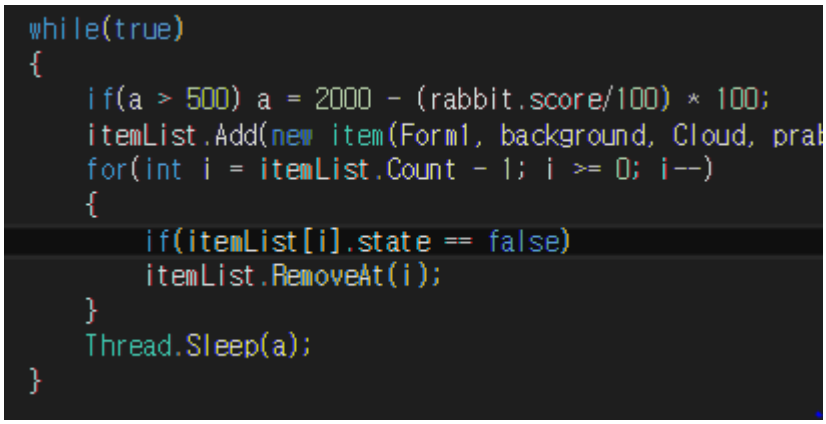
아까 있었던 form1_keydown 이벤트에서 호출되는 rabbit 클래스 move 메소드 안에 있는 코드이다. e 가 눌렸을 경우 현재 내가 가지고 있는 star 의 개수가 충분하고 지금 상태가 무적 모드가 아닐시에 새로운 스레드를 생성한다.

그후 star 한 개 감소시키고 over 값을 true로 변경하고 이미지를 angryrabbit 으로 변경한 뒤 3초간 잠든다. 그 후 다시 이미지를 변경시키고 over 값을 false로 변경한다.

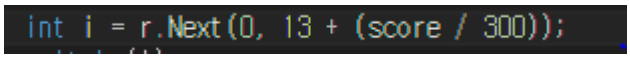
6. .레벨에 따른 난이도 차등 구현

레벨은 score / 100 + 1 로 스코어가 100점씩 쌓일 때 마다 레벨이 1씩 올라가게 해두었고

레벨이 올라갈수록 달라지는 점은 다음과 같다.



위는 아이템 매니저의 코드로 레벨이 올라갈수록 아이템 매니저가 잠들어 있는 주기가 짧아져 아이템이 더욱 자주 생성되며



위는 아이템 생성할 때 나오는 난수 값인데 13이 넘어가면 폭탄이 생성된다. 레벨이 3씩 올라 갈 때마다 폭탄이 생성될 확률이 증가한다.

```
private void Cloud_move()
{
    int speed = 100;
    cloud.Wait(100);
    bool p = true;
    Random r = new Random();
    int move_range;
    while (true)
    {
        move_range = r.Next(20, 150);
        for (int i = 0; i < move_range; i++)
        {
            if (pictureBox_cloud.Location.X <= 2)
            {
                p = true;
            }
            else if (pictureBox_cloud.Location.X >= pictureBox_background.Widt
                p = false;
            this.Invoke(new MethodInvoker(delegate ()
            {
                IntPtr x;
                if (!pictureBox_cloud.IsHandleCreated)
                    x = pictureBox_cloud.Handle;
                pictureBox_cloud.Location = my_cloud.move(p,my_rabbit.score);
            }));
            if (speed > 50) speed = 100 - ((my_rabbit.score / 100) * 20);
            cloud.Wait(speed);
        }
        p = !p;
    }
}
```

위는 cloud_move() 메소드이다 아래에서 5번째 줄을 보면 레벨이 올라갈수록 잠자고 있는 시간이 줄어든다. 이는 곧 구름이 빠르게 움직이는 것을 의미한다.

정리

1. 레벨이 올라갈수록 아이템이 빠르게 생성됨
2. 레벨이 올라갈수록 폭탄 아이템이 생성될 확률이 올라감
3. 레벨이 올라갈수록 구름이 빠르게 움직임

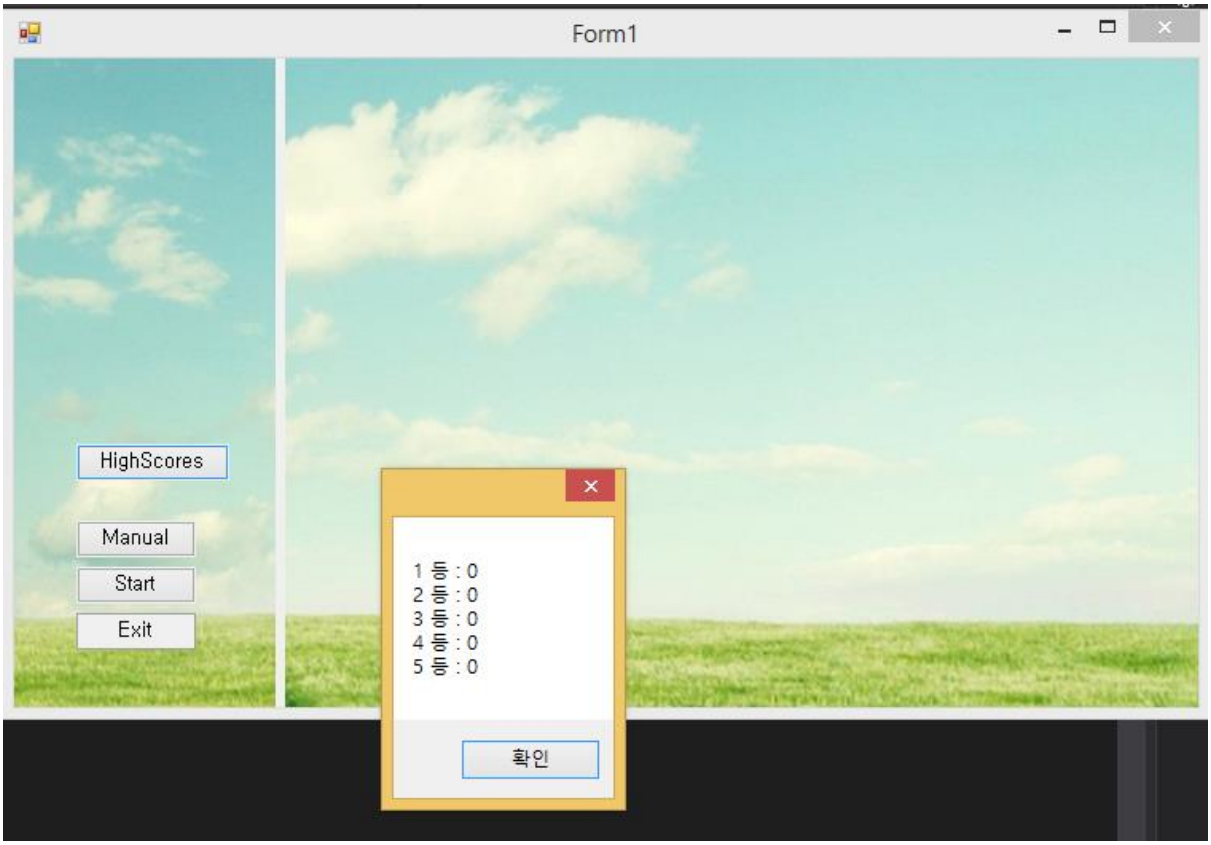
7. 모든 것을 구현하고 예시 프로그램처럼 돌아감

   게임 끝나고 재시작 기능

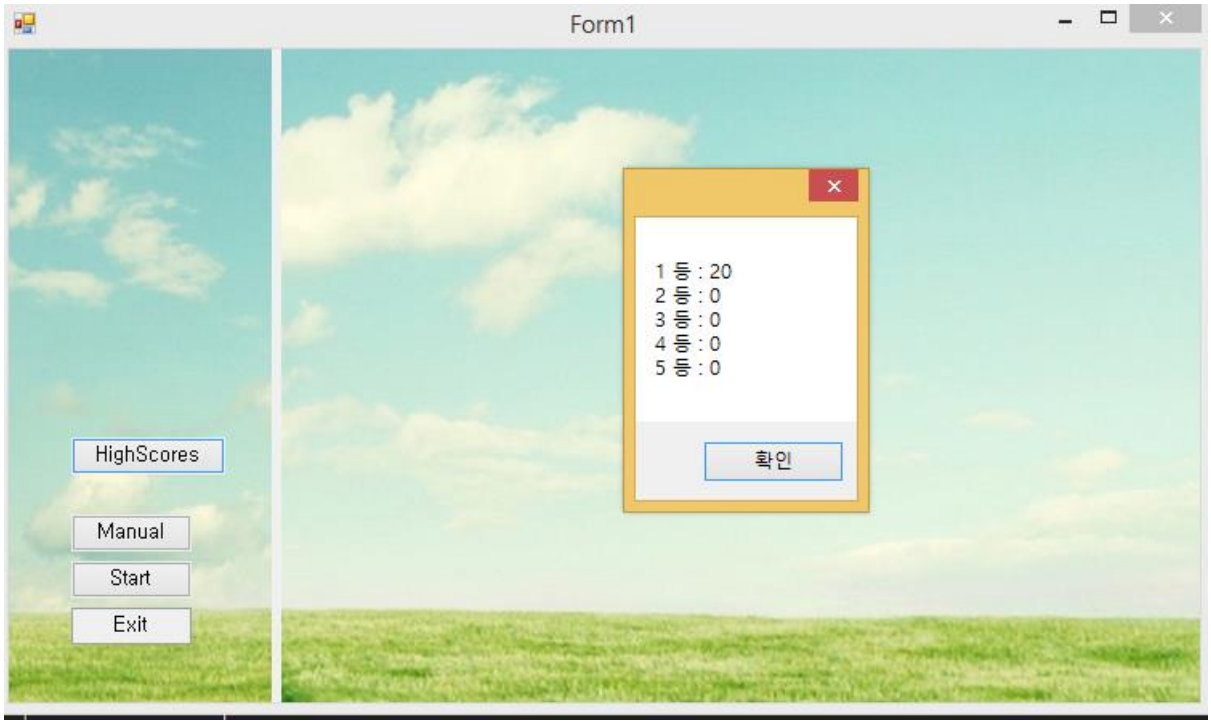
```
if (myrabbit.health < 1)
{
    Application.Restart();
}
```

체력이 0이 되면 application.Restart() 메소드 실행

8. High Score를 구현하는 기능



처음 실행시



20점을 먹고 종료 한 뒤 다시 켰을 때

```
        string[] scores = new string[5];
        if (File.Exists(@"./rank.txt") == false)
        {
            FileStream a = File.Create(@"./rank.txt");
            a.Close();

            StreamWriter sw = new StreamWriter(@"./rank.txt");
            for (int i = 0; i < 5; i++)
            {
                sw.WriteLine(0.ToString());
            }
            sw.Close();
        }
        StreamReader sr = new StreamReader(@"./rank.txt");
        for (int i = 0; i < 5; i++)
        {
            scores[i] = sr.ReadLine();
        }
        sr.Close();
        for (int i = 0; i < 5; i++)
            i_scores[i] = Int32.Parse(scores[i]);
        Array.Sort(i_scores);
        Array.Reverse(i_scores);
```

위는 Form이 로드 될 때 실행되는 코드이다. Rank.txt 파일이 있는지 검사하는데 없다면 새로 만든 다음에 0값을 5개 추가한다. 그 후 StreamReader로 값을 모두 읽어온 후에

Array.Sort를 실행하는데 이렇게 하면 배열은 내림차순으로 정렬된다. 그리하여 내림차순으로 정렬한 뒤 한번 뒤집어 주면 오름차순이 된다.

```
    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        i_scores[5] = my_rabbit.score;
        Array.Sort(i_scores);
        Array.Reverse(i_scores);
        StreamWriter sw = new StreamWriter(@"./rank.txt");
        for (int i = 0; i < 5; i++)
        {
            sw.WriteLine(i_scores[i].ToString());
        }
        sw.Close();
    }
```

위는 Form이 클로징 될 때 실행되는 코드이다. 현재 내가 가지고 있는 점수를 배열의 6번째 멤버로 넣고 배열을 오름차순으로 정렬한 뒤 앞에서부터 5개만 파일에 쓴다.

9. 추가 구현 점프 및 사운드

```
        else if (e.KeyCode == Keys.NumPad8)
        {
            if (location.Y == 278)
            {
                (new Thread(() =>
                {
                    Sound_jump.Play();
                    double p = location.Y;
                    int i = 0;
                    int dir = -1;
                    while (i < 10)
                    {
                        form.Invoke(new MethodInvoker(delegate ()
                        {
                            if (location.Y == p - 75)
                                dir = 1;

                            location.Y = location.Y + (dir * 15);
                            myrabbit.Location = new Point(myrabbit.Location.X, location.Y);

                        }));
                        i++;
                        Thread.Sleep(30);
                    }
                })).Start();

            }
        }
```

Rabbit.move 메소드에 있는 코드이다. 만약 입력된 방향키가 NumPad8 일 때 실행된다. 이때 현재 rabbit의 y값이 278(땅)이 아니면 그냥 입력을 무시하도록 했다.

그 후 새로운 스레드가 람다식으로 생성되고 점프에 알맞은 사운드가 플레이 되고 75 거리만큼 위로 뛰었다가 다시 내려온다.

```
        SoundPlayer Sound_jump = new SoundPlayer(sosil3_homework.Properties.Resources.Jump);
        SoundPlayer Sound_anger = new SoundPlayer(sosil3_homework.Properties.Resources.anger);
```

위는 rabbit 에서 사용되는 sound 소스 리소스 들이다. 점프 할 때와 무적모드에 돌입했을때 2가지 사운드를 사용한다.

그후 버튼 스타트를 누르면

```
        }
        private void btn_Start_Click(object sender, EventArgs e)
        {
            label1.Font = new Font("Tahoma", 30, FontStyle.Italic);
            timer.Interval = 1000;
            timer.Tick += new EventHandler(timer_elapsed);
            timer.Start();
            Sound_Start.Play();
        }
```

타이머가 시작되고 1초마다 timer_elapsed 메소드가 시작된다.

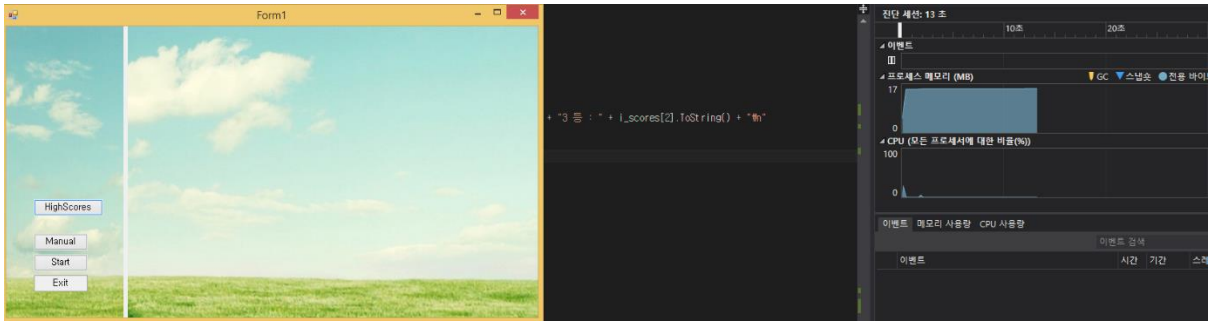
```
void timer_elapsed(object e, EventArgs a)
{
    s--;
    label1.Text = s.ToString();
    label1.Visible = true;
    label1.Invalidate(true);
    if (s == 0)
    {
        label1.Dispose();
        timer.Dispose();
        groupBox1.Visible = true;
        btn_Start_fake.Visible = true;
        btn_Start.Visible = false;
        pictureBox_cloud.Visible = true;
        pictureBox_nomal_rabbit.Visible = true;
        pictureBox_nomal_rabbit.Enabled = false;
        groupBox1.Parent = pictureBox_left;

        pictureBox_nomal_rabbit.BackColor = Color.Transparent;
        pictureBox_nomal_rabbit.Parent = pictureBox_background;

        pictureBox_cloud.BackColor = Color.Transparent;
        pictureBox_cloud.Parent = pictureBox_background;

        my_cloud = new cloud(pictureBox_cloud.Location);
        my_itemManager = new itemManager(this, pictureBox_background, pictureBox_cloud, pictureBox_nomal

        cloud = new Task(new Action(Cloud_move));
        cloud.Start();

        label_health.Text = "Health : " + my_rabbit.health.ToString();
        label_Score.Text = "Score : " + my_rabbit.score.ToString();
        label_Level.Text = "Level : " + (my_rabbit.score + 1).ToString();
        this.DoubleBuffered = true;

    }
}
```

그후 timerelapsed 가 4번 실행 = 4초가 지나가면

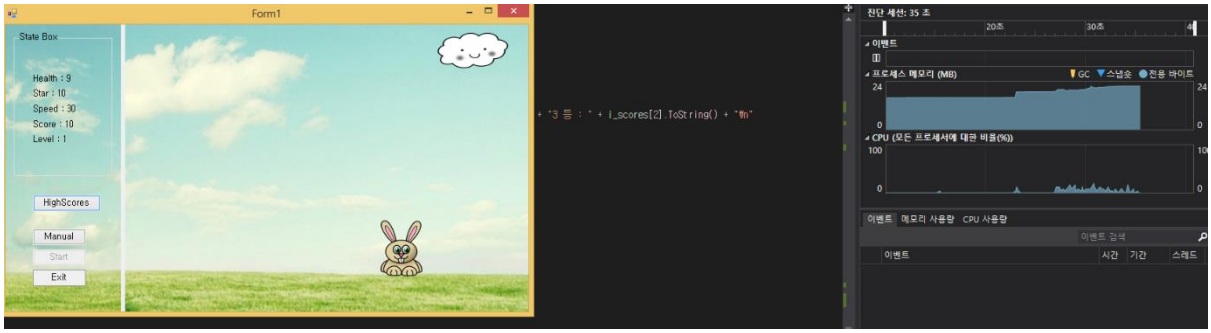
게임이 본격적으로 시작하게 된다.

10. 고찰

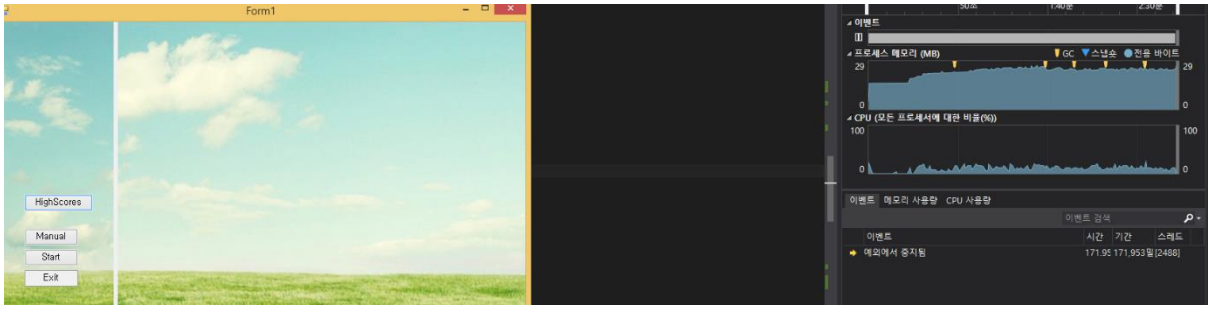
이번 과제는 꽤나 어려웠다. 자원 관리를 잘 하지 않으면 렉이 걸려서 이 부분에서 상당히 고민했던 것 같다.



게임 시작 직전의 메모리 상황 약 17mb 사용중



게임 시작 한 직후 메모리 상황 약 24mb 사용



약 레벨 5까지 도달 했을 때의 메모리 상황

약 29mb

테스트 해본 결과 약 30레벨로 하여 구름도 빨리 움직이고 아이템도 자주 생성되고 해도 30mb를 넘지 않는 것 같았다.

그리고 사운드플레이어도 c#에서 기본으로 제공되는 SoundPlayer 클래스를 사용했는데 이를 사용할 시 동시에 여러 개의 재생이 되지 않아 매우 아쉬웠다.

현재 코드를 보면 invoke를 해주고 여러 자원에 접근하기 위해서 생성자가 굉장히 지저분 하다.

이 부분에 대한 해결책을 singleton 패턴을 사용하는 방법이 있다고 듣기는 했는데 이미 코드를 다 짜놓은 상태에서 알게 되어서 너무 늦었다.

이번 과제에서 토끼와 아이템, 구름의 움직임을 부드럽게 하려고 굉장히 많은 애를 썼다. 각각 쓰레드를 두고 돌리게 되면 쓰레드 끼리 서로 돌아갈려고 경쟁을 하게 되고, 경쟁하지 않게 sleep 메소드를 사용해도 sleep 값이 충분히 크지 않으면 렉이 걸리게 되어 이 부분을 조정하는게 매우 힘들었다. 이에 대한 해결책으로 아이템은 task로 구현하였더니 렉이 훨씬 적게 걸리게 되었다.

사실 이번 과제는 12점을 목표로 하려고 했었으나 과제가 너무 많아서 자체 하향 조정 했다. 과제를 시작할 때 객체지향적으로 클래스도 많이 만들고 여러 메소드들을 만들고 하였는데 결과는 코드가 너무 더러워 진 것 같아서 조금 아쉽다.