제출일 : 2022. 05. 29.

KWANGWOON UNIVERSITY

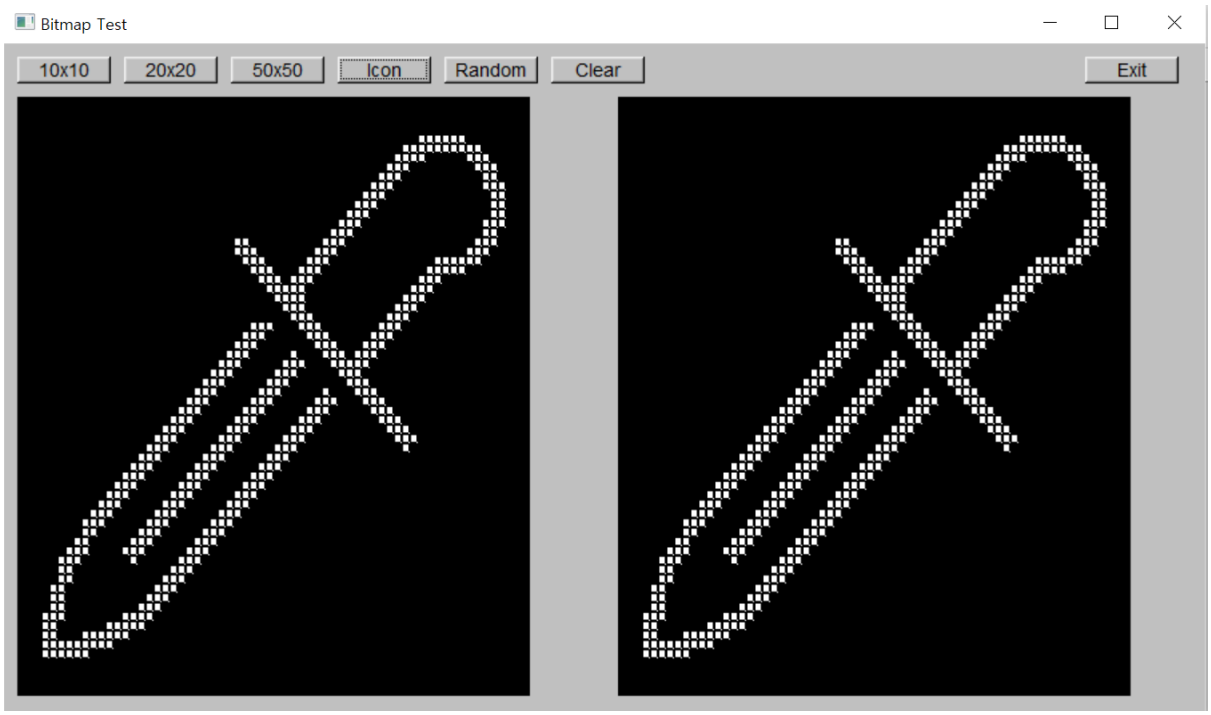
담당교수	이강훈 교수님	학 번	2021203034
학 과	소프트웨어학부	이 름	허찬영

KWANGWOON UNIVERSITY

■ Begin with a summary of your results

Which requirements did you fulfill? And which didn't you?

```
class BinaryBitmap {
public:
    BinaryBitmap(int w, int h);

    void clear();

    bool get(int x, int y) const;
    void set(int x, int y, bool v);

    int getWidth() const;
    int getHeight() const;
private:
    vector<bool> pixel_values;
    int width, height;
};
```

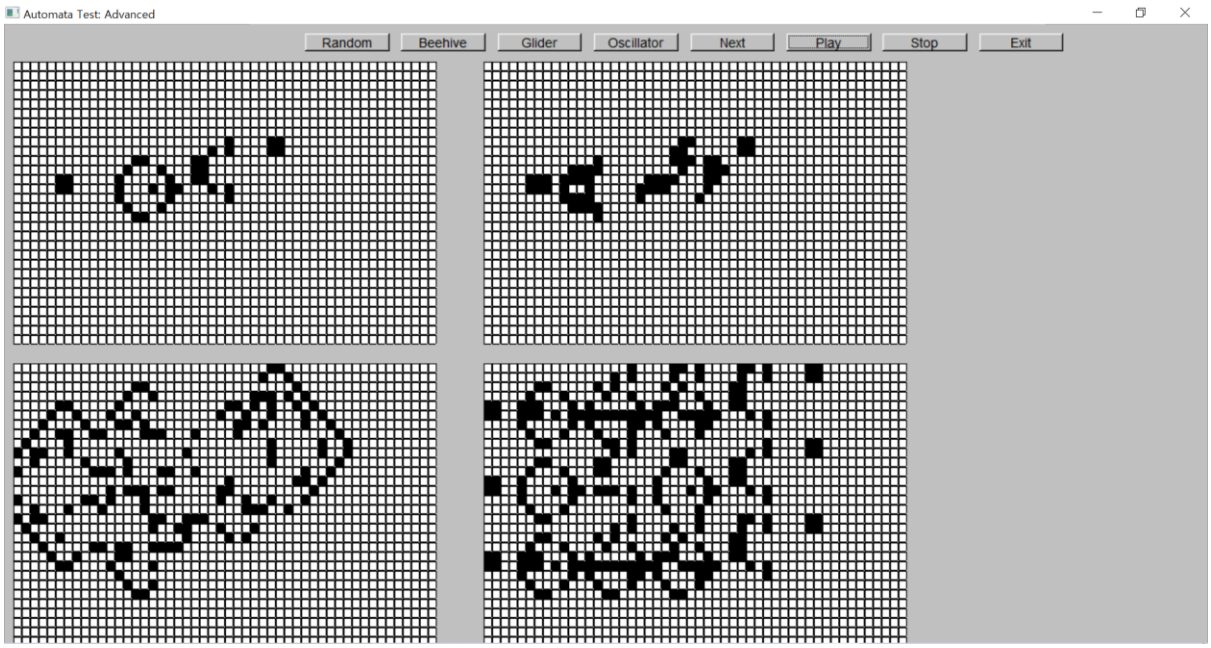


BinaryBitmap Class를 구현한 HW2.h, HW2.cpp 파일을 통해 main_bitmap_driver.

cpp를 실행 시켰다. 창 크기를 조절하는 버튼을 클릭하면 창의 크기가 조절되고 Icon 버튼을 누르면 위와 같은 그림이 출력된다. Random 버튼을 누르면 셀이 랜덤적으로 배치된다.

```cpp
class CellularAutomata {
public:
    enum class InitType { CLEAN, RANDOM, BEEHIVE, GLIDER, OSCILLATOR };

    CellularAutomata(int w, int h);
    virtual ~CellularAutomata();

    void initialize(InitType t);
    void clear();
    bool copy(const CellularAutomata& automata);

    virtual void update();

    void set(int x, int y, bool v);
    bool get(int x, int y) const;

    int getWidth() const;
    int getHeight() const;
protected:
    BinaryBitmap* binary_bitmap;
    int c_width, c_height;
};
```

```cpp
class LifeAutomata :public CellularAutomata {
public:
    LifeAutomata(int w, int h);
    void update() override;
};

class SeedAutomata : public CellularAutomata {
public:
    SeedAutomata(int w, int h);
    void update() override;
};


class ReplicatorAutomata : public CellularAutomata {
public:
    ReplicatorAutomata(int w, int h);
    void update() override;
};
```



BinaryBitmap의 기능을 상속받은 Cellularautomata class와 Cellularautomata를 상속 받은 Lifeautomata를 구현하여 main_automata_driver_advanced.cpp를 실행시켰다.

Did you implement some additional features? What are those?

```
class SeedAutomata : public CellularAutomata {
public:
    SeedAutomata(int w, int h);
    void update() override;
};


class ReplicatorAutomata : public CellularAutomata {
public:
    ReplicatorAutomata(int w, int h);
    void update() override;
};
```

Advanced를 구현하기 위해 SeedAutomata class와 ReplicatorAutomata class를 구현했다.

```
virtual ~CellularAutomata();
```

메모리 누수를 막기 위해 소멸자를 public 영역에 추가했다.

■ For each requirement (basic/advanced), explain how you fulfilled it

(중요한 기능을 가진 부분만 설명하겠습니다.)

```
BinaryBitmap::BinaryBitmap(int w, int h)
    :width{ w }, height{ h }
{
    pixel_values.resize(width*height, false);
}

void BinaryBitmap::clear() {
    for (int i = 0; i < pixel_values.size(); i++)
    {
        pixel_values[i] = false;
    }
}
```

BinaryBitmap class는 생성자를 이용해 width와 height를 초기화 했고 resize 함수를 이용하여 셀들의 상태를 가진 1차원 배열 pixel_values를 false로 초기화 했다. Clear()함수는 pixel_values를 false로 설정해 모든 셀들을 죽게 하는 기능을 한다.

```
bool BinaryBitmap::get(int x, int y) const
{
    if ((x >= 0 && x < width) && (y >= 0 && y < height))
    {
        if (pixel_values[(y * width + x)] == 1)
            return true;
        else
            return false;
    }
    else
        return false;
}

void BinaryBitmap::set(int x, int y, bool v)
{
    if ((x >= 0 && x < width) && (y >= 0 && y < height))
        pixel_values[y*width+x] = v;
}
```

set함수는 pixel_values에 주어진 영역 안에 있는 셀의 상태를 설정하는 기능을 한다.

get함수는 주어진 영역 안에 있는 셀의 상태에 따라 해당 위치에 셀이 살아 있으면 True를 반환, 죽어있으면 false를 반환하여 화면에 셀의 상태를 정상적으로 출력할 수 있도록 돕는 기능을 한다. 이 때 pixel_values는 1차원이지만 화면엔 2차원으로 출력을 해야 하므로 값을 잘 조정(y* width + x)하여 구현했다.

```
CellularAutomata::CellularAutomata(int w, int h)
    : c_width{w}, c_height{h}
{
    binary_bitmap = new BinaryBitmap(w, h);
}

CellularAutomata::~CellularAutomata() {
    delete binary_bitmap;
}
```

```
protected:
    BinaryBitmap* binary_bitmap;
    int c_width, c_height;
```

CellularAutomata class는 BinaryBitmap의 기능을 이용하기 위해 BinaryBitmap과 결합되도록(has-a 관계) 설정했다. 생성자에서 창의 크기를 초기화하고 new를 이용하여 메모리를 할당하였다. 그리고 메모리 누수를 막기 위해 소멸자를 이용하여 메모리를 해제 시켰다.

```
case InitType::RANDOM:
{
    srand((int)time(NULL));

    for (int i = 0; i < c_width; i++)
    {
        for (int j = 0; j < c_height; j++)
        {
            if (rand() % 2 == 0)
                set(i, j, false);
            else
                set(i, j, true);
        }
    }

    break;
```

enum class InitType와 void initialize(InitType t) 함수를 통해 각 패턴들을 초기화 시켰다.

Random의 경우 rand()함수를 통해 셀의 상태를 set함수를 이용하여 무작위로 설정했다.

```
case InitType::OSCILLATOR:
    binary_bitmap->clear();
    set(5, 12, true);
    set(5, 13, true);
    set(6, 12, true);
    set(6, 13, true);

    set(15, 10, true);
    set(14, 10, true);
    set(13, 11, true);
    set(12, 12, true);
    set(12, 13, true);
    set(12, 14, true);
    set(13, 15, true);
    set(14, 16, true);
    set(15, 16, true);
    set(16, 13, true);
    set(17, 11, true);
```

OSCILLATOR 패턴은 set함수를 이용하여 초기 패턴을 설정했다. CLEAN, BEEHIVE, GLIDER 역시 동일한 방식으로 초기화를 했다.

```
bool CellularAutomata::copy(const CellularAutomata& automata)
{
    if (automata.getHeight() == binary_bitmap->getHeight() && automata.getWidth() == binary_bitmap->getWidth()) {
        for (int j = 0; j < getHeight(); j++) {
            for (int i = 0; i < getWidth(); i++) {
                set(i, j, automata.get(i, j));
            }
        }
        return true;
    }
    else
        return false;
}
```

다음은 copy 함수이다. 이 함수는 셀의 상태를 가지고 있는 automata를 인자로 받는다. Automata의 크기와 binary_bitmap의 크기가 같을 경우 true를 반환하고 아닐 경우 false를 반환한다. 그리고 bool형으로 설정된 automata의 정보와 set함수를 이용하여 모든 셀을 복사한다.

Celularautomata class의 멤버인 set, get, getWidth, getHeight 함수는 BinaryBitmap class의 멤버를 이용하여 정의했다. Update함수는 virtual 함수로 설정하여 자식 클래스에서 함수를 재정의 하도록 구현했다.

```
void LifeAutomata::update()
{
    BinaryBitmap* copy_map = new BinaryBitmap(getWidth(), getHeight());

    for (int j = 0; j < getHeight(); j++)
    {
        for (int i = 0; i < getWidth(); i++)
            copy_map->set(i, j, this->get(i, j));
    }
    for (int y = 0; y < getHeight(); y++) {
        for (int x = 0; x < getWidth(); x++) {
            int cnt = 0;

            if (copy_map->get(x - 1, y - 1) == 1) { cnt++; }
            if (copy_map->get(x - 1, y - 0) == 1) { cnt++; }
            if (copy_map->get(x - 1, y + 1) == 1) { cnt++; }
            if (copy_map->get(x, y - 1) == 1) { cnt++; }
            if (copy_map->get(x + 1, y - 1) == 1) { cnt++; }
            if (copy_map->get(x, y + 1) == 1) { cnt++; }
            if (copy_map->get(x + 1, y + 0) == 1) { cnt++; }
            if (copy_map->get(x + 1, y + 1) == 1) { cnt++; }
```

```
            if (copy_map->get(x, y) == 1) {
                if (!(cnt == 2 || cnt == 3))
                    set(x, y, false);
            }
            else {
                if (cnt == 3)
                    set(x, y, true);
            }
        }
    }

    delete copy_map;
```

LifeAutomata class에 있는 update() 함수는 Cellularautomata class에 있는 update() 함수를 오버라이딩 하여 재정의한다. New 연산자를 이용하여 copy_map을 메모리에 할당하여 copy_map에 모든 셀들의 상태를 저장한다. 그리고 반복문과 제어문을 통해 이웃한 셀들의 상태를 확인하고 주어진 조건에 따라 셀들의 상태를 재정의 한다. LIfeAutomata::update() 함수는 B3/S23 규칙을 따랐으며 살아 있는 셀에 이웃한 셀이 2개나 3개가 아니라면 죽이고 죽어 있는 셀 주변에 살아있는 셀 3개가 있으면 살리는 규칙으로 구현하였다. 그리고 delete 연산자를 통해 메모리를 해제하였다.

```
if (copy_map->get(x, y) == 1) {
    set(x, y, false);
}
else {
    if (cnt == 2)
        set(x, y, true);
}
```

다음은 SeedAutomata class이며 B2/S 규칙을 따른다.

```
if (get(x, y) == 1) {
    if (cnt == 1)
        copy_map->set(x, y, true);
    else if (cnt == 3)
        copy_map->set(x, y, true);
    else if (cnt == 5)
        copy_map->set(x, y, true);
    else if (cnt == 7)
        copy_map->set(x, y, true);
    else
        copy_map->set(x, y, false);
}
else {
    if (cnt == 1)
        copy_map->set(x, y, true);
    else if (cnt == 3)
        copy_map->set(x, y, true);
    else if (cnt == 5)
        copy_map->set(x, y, true);
    else if (cnt == 7)
        copy_map->set(x, y, true);
    else
        copy_map->set(x, y, false);
}
```

다음은 Replicator로 B1357/S1357 규칙을 따른다. 이 class의 update() 함수는 LifeAutomata class의 멤버인 update함수와 동일한 방식으로 구현했다.

■ Conclude with some comments on your work

Key challenges you have successfully tackled

상속, 포인터와 참조를 사용하고 그래픽 라이브러리를 이용하여 그래픽 프로그램이 정상적으로 작동할 수 있었다.

Limitations you hope to address in the future

상속과 virtual 함수, 포인터와 참조에 대한 이해도를 키워 더 수준 높은 프로그램을 제작해보고 싶다.