**2017년 2학기**

**[CSED 232] 객체지향프로그래밍**

**ASSN 3 보고서**

제목: 세포 증식 게임

POVIS ID: dj1122

학과: 산업경영공학과

학번: 20130220

이름: 이다진

“나는 이 프로그래밍 과제를 다른  사람의 부적절한 도움 없이 완수하였습니다.”

목차

Ⅰ. 개요1

Ⅱ. 전체 조와 알고리즘1

1. 전체 구조1

2. 알고리즘1

3. 클래스 구성4

Ⅲ. 프로그램 실행 방법7

Ⅳ. 예제 및 실행 결과7

Ⅴ. 실행 결과에 대한 토의9

1. 토론9

2. 결론10

3. 개선 방향10

Ⅰ. 개요

본 프로그램은 C++을 사용하여 “세포 증식 게임”을 구현하고 QT를 사용하여 실행 및 화면에 출력한다. 프로그램 구현을 통해 C++의 상속 기능, 다형성, 연산자 오버로딩을 활용 방법을 습득한다. QT 사용을 통해 GUI(Graphic User Interface) Programming을 접해본다. 이러한 과정을 통해 게임을 현실감있게 시각화하고, 개발도구의 개성과 활용 범위을 인식한다.

Ⅱ. 전체 구조와 알고리즘

1. 전체 구조

(1) 메인 창에서 기본 메뉴 버튼을 출력한다.

(2-1) START 버튼 클릭: 메인 창을 숨기고 세포 증식 게임을 창을 띄운다.

(2-2) EXIT 버튼 클릭: 창을 닫고 메인 프로그램을 종료한다.

(3) Game 객체를 생성하고 게임을 시작한다.

(4) 사용자(셀)와 30개의 생명체(적, 바이러스, 먹이)를 생성하여 화면에 출력한다.

(5) 15초마다 5개의 생명체를 생성하여 화면에 출력한다.

(6) 사용자와 적들은 충돌 시 크기 비교를 통해 크기가 증감하거나 소멸된다.

(7) 사용자가 사라지면 메인 프로그램을 종료한다.

2. 알고리즘

(1) Game 객체를 생성하고 게임을 시작한다.

1. QGraphicsScene 객체 scene을 생성한다.

2. Game 객체를 game을 생성하며, 매개변수로 scene을 전달한다.

3. game의 메소드 playGame()을 호출한다.

(2) 사용자(셀)를 생성하여 화면에 출력한다.

void Game::playGame() {

1) Cell 객체를 생성하며, 매개변수로 scene을 전달한다.

2) Cell 생성자 내에서 scene에 객체를 추가하여 화면에 출력한다.

3) 생성된 Cell 객체를 Bio\* 타입으로 다이나믹 캐스팅하여 player 변수에 저장한다.

}

(3) 30개의 생명체(적, 바이러스, 먹이)를 생성하여 화면에 출력한다.

void Game::playGame() {

1) makeBios() 메소드를 6번 호출한다.

void makeBios() {

2) 1~100 의 난수를 생성한다.

3-1-1) 난수가 1~70 인 경우 Feed 객체를 생성하며, 매개변수로 scene을 전달한다.

3-1-2) Feed 생성자 내에서 scene에 객체를 추가하여 화면에 출력한다.

3-1-3) 생성된 Feed 객체를 Bio\* 타입으로 다이나믹 캐스팅하여 bios 리스트에 추가한다.

3-2) 난수가 71~85인 경우 Virus 객체를 대상으로 3-1의 과정을 진행한다.

3-3) 난수가 86~100인 경우 Enemy 객체를 대상으로 3-1의 과정을 진행한다.

}

}

(4) 15초마다 5개의 생명체를 생성하여 화면에 출력한다.

1) QTimer 객체를 생성하여 makeTimer에 할당한다.

2) makeTimer 기준으로 15초마다 makeBios()를 호출한다.

(5-1) 사용자는 충돌 시 크기 비교를 통해 크기가 증가하거나 소멸된다.

1) collingItems() 로 사용자와 충돌한 객체들을 collided 리스트에 저장한다.

2) 사용자와 collided 리스트의 객체들의 크기를 비교한다. (오버로딩된 > 연산자 이용)

사용자의 크기가 큰 경우

3-1) 사용자의 크기를 증가시킨다. (오버로딩된 + 연산자 이용)

3-2) 단, Virus 객체와 충돌한 경우 크기를 감소시킨다. (오버로딩된 + 연산자 이용)

3-3) 사용자와 충돌한 객체를 소멸시킨다.

사용자의 크기가 작은 경우

4) 사용자 객체를 소멸시키고 메인 프로그램을 종료한다.

(5-2) 적들은 충돌 시 크기 비교를 통해 크기가 증가하거나 소멸된다.

(5-1)번과 동일하지만 적의 크기가 작은 경우 메인 프로그램을 종료하지 않는다.

(6) 사용자가 사라지면 메인 프로그램을 종료한다.

1) (5-1)번에서 사용자보다 큰 객체와 충돌한 경우 사용자 객체를 소멸시키고 메인 프로그램을 종료한다.

2) 사용자의 크기가 0이하가 된 경우 사용자 객체를 소멸시키고 메인 프로그램을 종료한다.

3) 사용자가 화면 밖으로 나간 경우 사용자 객체를 소멸시키고 메인 프로그램을 종료한다.

3. 클래스 구성

(1) Game

멤버 변수

1) Bio \* player: 사용자 객체를 담는 포인터 변수

2) QList<Bio \*> bios: 생명체의 객체들을 담는 QList

3) QGraphicsScene \* s: 게임 화면의 주소를 담는 포인터 변수

4) QTimer \* makeTimer: 15초마다 생명체 생생을 위한 시간을 세는 변수

메소드

1) Game(QGraphicsScene \* s): 생성자

2) void playGame(): 게임을 진행하는 메소드

3) void makeBios(): 생명체 5개를 생성하는 메소드

(2) Bio

멤버 변수

1) float size: 생명체의 크기

2) float axisX: 생명체의 x좌표 위치

3) float axisY: 생명체의 y좌표 위치

4) QGraphicsScene \* s: 게임 화면의 주소를 담는 포인터 변수

메소드

1) Bio(), Bio(QGraphicsScene \* s): 생성자

2) void operator+(Bio\* bio): 바이러스와 충돌할 경우 자신의 크기를 감소시키고, 다른 생명체와 충동한 경우 크기 자신의 크기를 증가시키는 연산자

3) bool operator>(Bio\* bio): 생명체끼리의 크기를 비교하는 연산자

4) virtual void Draw(): 생명체를 화면에 그리는 가상 메소드

5) virtual void Move(): 생명체를 움직이는 가상 메소드

6) float getSize(), void setSize(float size), float getAxisX(), void setAxisX(float axisX), float getAxisY(), void setAxisY(float axiyY), QGraphicsScene \* getS(): get & set 메소드

(3) Cell

Cell은 Bio, QObject, QGraphicsEllipseItem를 상속받는다.

멤버 변수

1) enum Direction {LEFT, RIGHT, UP, DOWN}: 상, 하, 좌, 우 중 현재 객체가 움직이는 방향을 나타내는 변수

2) enum Direction direction: 움직이는 방향을 저장하는 변수

3) bool boost: Booster On/Off 여부를 나타내는 변수

4) float velocity: 생물체의 속도를 나타내는 변수

5) QTimer \* moveTimer: 50ms마다 생명체 움직임을 위한 시간을 세는 변수

6) QTimer \* boostTimer: 50ms마다 생명체의 크기를 줄이기 위해 시간을 세는 변수

메소드

1) void Move(): Direction, boost 변수에 따라 Cell을 움직이는 함수

2) void Boost(): 생명체의 크기를 0.95배로 줄이는 함수

3) Cell(), Cell(QGraphicsScene \* s): 생성자

4) ~Cell(): 소멸자

5) void Draw(): Cell을 게임 화면 내에 그리는 함수. size에 비례하며 파란색 원의 형태이다.

6) void keyPressEvent(QKeyEvent\* event): 유저의 키 입력을 관리. 방향키 입력에 대해 direction을 바꾸고, 스페이스바 입력에 대해 boost 값을 바꾼다.

(4) Enemy

Enemy는 Bio, QObject, QGraphicsEllipseItem를 상속받는다.

멤버 변수

1) enum Direction {LEFT, RIGHT, UP, DOWN}: 상, 하, 좌, 우 중 현재 객체가 움직이는 방향을 나타내는 변수

2) enum Direction direction: 움직이는 방향을 저장하는 변수

3) float velocity: 생물체의 속도를 나타내는 변수

5) QTimer \* moveTimer: 50ms마다 생명체 움직임을 위한 시간을 세는 변수

6) QTimer \* dirTimer: 1초마다 생명체 움직임 방향 변경을 위한 시간을 세는 변수

메소드

1) void Move(): Direction 변수에 따라 Enemy를 움직이는 함수

2) void changeDir(): 3초 간격으로 direction을 랜덤하게 바꿔주는 함수

3) Enemy(), Enemy(QGraphicsScene \* s): 생성자

4) ~Enemy(): 소멸자

5) void Draw(): Enemy를 게임 화면 내에 그리는 함수. size에 비례하며 빨간색 원의 형태이다.

(5) Virus

Virus는 Bio, QObject, QGraphicsRectItem를 상속받는다.

멤버 변수

1) enum Direction {LEFT, RIGHT, UP, DOWN}: 상, 하, 좌, 우 중 현재 객체가 움직이는 방향을 나타내는 변수

2) enum Direction direction: 움직이는 방향을 저장하는 변수

3) float velocity: 생물체의 속도를 나타내는 변수

4) int moveCnt: 생물체 움직임 횟수를 세는 변수

5) int sizeCnt: 생물체 크기 감소 횟수를 세는 변수

6) QTimer \* moveTimer: 1초마다 생명체 움직임을 위한 시간을 세는 변수

7) QTimer \* sizeTimer: 4초마다 생명체 크기를 줄이기 위한 시간을 세는 변수

메소드

1) void Move(): Virus를 움직이는 함수. 1초 간격으로 오른쪽, 위, 왼쪽, 아래 순서대로 반복하여 움직인다.

2) void Shrink(): 4초 간격으로 size를 감소시키는 함수

3) Virus(), Virus(QGraphicsScene \* s): 생성자

4) ~Virus(): 소멸자

5) void Draw(): Virus를 게임 화면 내에 그리는 함수. size에 비례하며 검은색 정사각형의 형태이다.

2) void changeDir(): 1초 간격으로 direction을 랜덤하게 바꿔주는 함수

(6) Feed

Feed는 Bio, QObject, QGraphicsRectItem를 상속받는다.

메소드

1) Feed(), Virus(QGraphicsScene \* s): 생성자

2) ~Feed(): 소멸자

3) void Draw(): Feed를 게임 화면 내에 그리는 함수. size에 비례하며 초록색 정사각형의 형태이다.

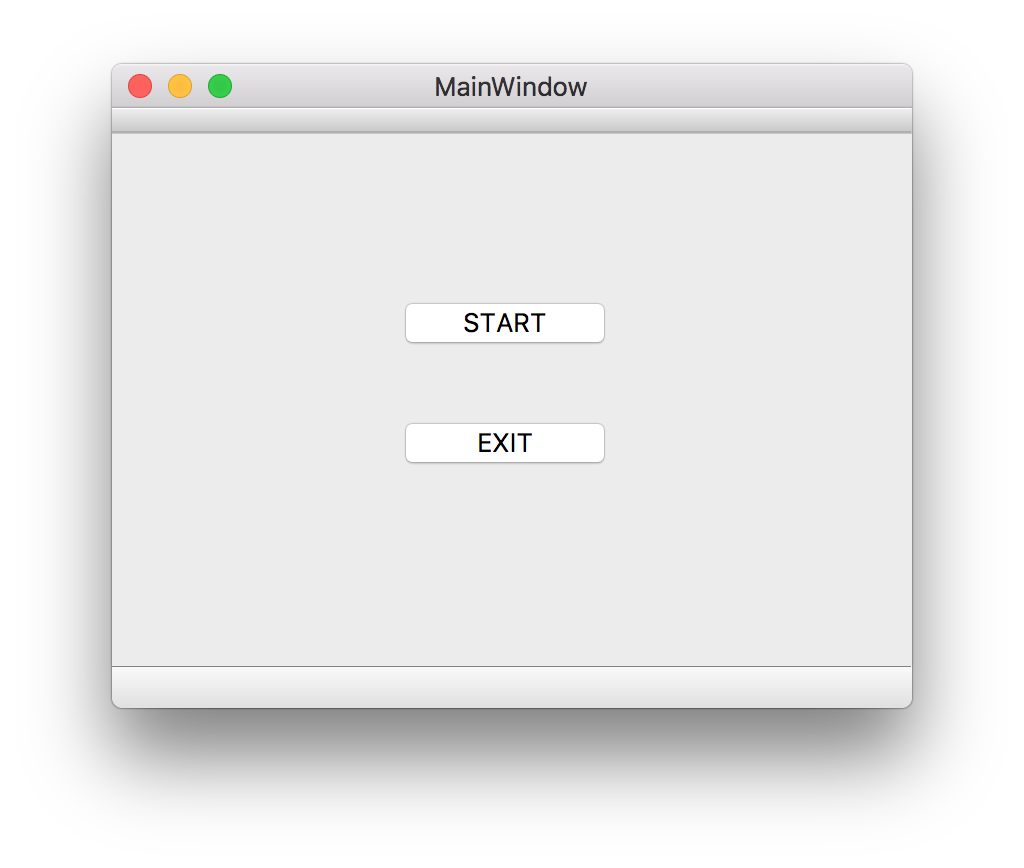
4) void Move(): 아무 동작도 하지 않고 return 한다.

Ⅲ. 프로그램 실행 방법

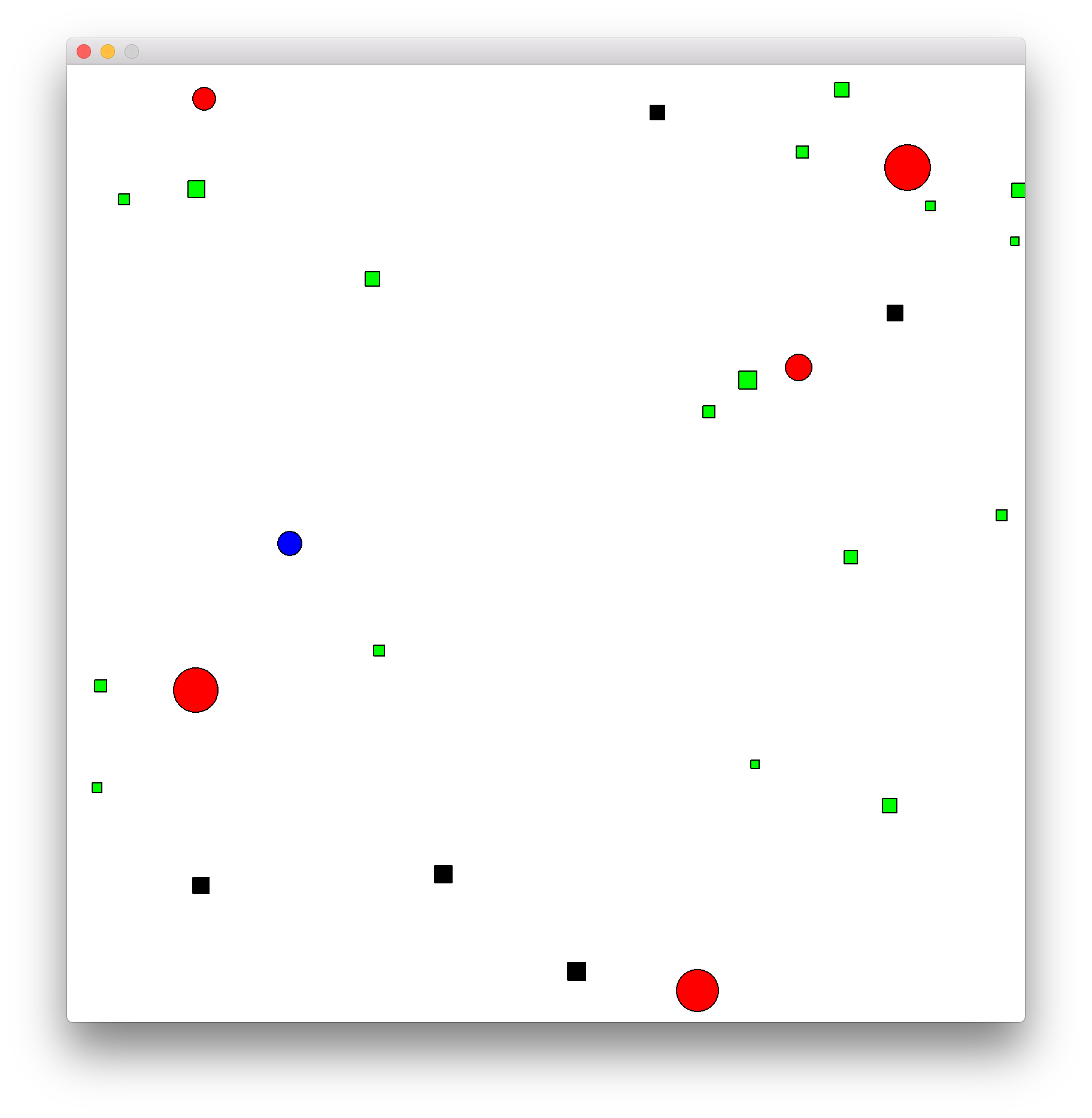
본 프로그램은 QT 프로젝트 내에서 실행된다. 필요한 모든 .cpp 파일, .h파일, .pro 파일이 포함되어야 하며 이때 .pro 파일은 일종의 makefile의 역할을 한다.

Ⅳ. 예제 및 실행 결과

1. 메인 프로그램 실행 화면



2. 게임 실행 화면



Ⅴ. 실행 결과에 대한 토의

1. 토론

프로그램 제작 중 발생한 문제와 해결 방법을 작성하였다.

(1) 객체를 화면에 그리기

main 화면에서 게임 시작 버튼을 클릭하면 새 창에서 게임이 출력된다. 그런데 새 창의 주소를 각 객체가 알고 있어야 하므로 객체 생성 시 매개변수로 QGraphicsScene\* 타입의 포인터 변수를 전달하였다. 이때 Bio \* 타입에서 객체를 화면에 그리는 Draw()가 가상 함수이므로 생성자 안에 객체마다 Draw()메소드를 호출한다. 생명체들은 Bio\* 타입으로 관리되어야 하므로 생성자 호출 후 다이나믹 캐스팅하여 Bio\* 타입으로 형 변환한다.

(2) SIGNAL-SLOT 활용하기

moveTimer = new QTimer();

connect(moveTimer, SIGNAL(timeout()), this, SLOT(Move()));

moveTimer->start(50);

위의 코드는 QTimer를 이용하여 50ms가 경과할 때마다 Move()를 실행한다. Move()는 보통 함수와 달리 public slot 안에 선언되어야 하며 Move() 함수를 사용할 객체 클래스 내부에 Q\_OBJECT를 선언해야 한다.

(3) .pro 파일에서 선언 순서 변경하기

.cpp, .h 파일에 오류가 없는데도 Undefined symbols for architecture x86\_64 에러 메시지가 출력되며 빌드가 되지 않는 경우가 있다. 이 때 .pro 파일에서 자동으로 추가된 .cpp, .h 파일 명 간의 순서를 바꾸면 에러가 해결된다.

2. 결론

(1) 연산자 오버로딩의 유용함

프로그램 작성 중 가독성이 좋고 직관적인 함수의 이름을 짓는 것은 중요하다. 연산자 오버로딩을 사용하면 기호를 함수 이름처럼 사용할 수 있으며, 올바르게 이용할 경우 코드를 직관적이고 간결하게 작성할 수 있다.

(2) 개발 도구의 개성 인식

C++로 게임을 구현할 때 콘솔 화면에 게임을 출력하는 것은 불편하다. 화면 내에서 표현할 수 있는 형식도 제한되어 있고, 표현 방법도 어렵기 때문이다. 그런데 QT에서는 기존 라이브러리 클래스를 통해 쉽게 게임을 시각화할 수 있었다. collingItems() 등의 게임 실행 시 자주 발생하는 이벤트에 대한 API 또한 제공되었다. 본 과제를 통해 전통있고 기본적인 C++ 프로그래밍과 GUI 기반의 QT 프로그래밍의 차이를 인식할 수 있었다. 더 나아가 프로그램 목적에 맞는 도구를 선택하는 것의 프로그램 구현 시간을 크게 단축할 수 있다는 것을 알게 되었다.

3. 개선 방향

(1) 생명체 모양 개선

본 과제에서는 생명체 객체의 모양을 원, 정사각형 등으로 기본적인 도형과 색상을 이용해 표현하였다. 만약 객체의 특성에 걸맞게 모양을 구체화하거나 이미지로 객체를 표현한다면 게임 설명을 자세히 읽지 않아도 직관적으로 게임을 진행할 수 있으며 이는 사용자 경험을 향상으로 이어질 것이다.