2017년 2학기

객체지향프로그래밍

CSED232

Assignment #3

담당교수 : 윤은영

학번 : 20160074

학과 : 화학공학과

이름 : 고진민

POVIS ID : eric9709

--------------------------------------------------------------------------------------------------------------

<명예서약>

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

--------------------------------------------------------------------------------------------------------------

1. **프로그램 개요**

이 프로그램은 간단한 게임 프로그램이다. 플레이어는 세포를 조작하여 다른 생물체들을 잡아먹으며 세포의 크기를 키워나간다. 플레이어 외 다른 생물체에는 적 세포 (Enemy), 바이러스 (Virus), 먹이 (Feed) 가 있으며, 각각 모양, 색, 움직이는 경향성 등이 다르다.

플레이어 세포와 적 세포는 자신보다 크기가 작은 다른 생물체를 잡아먹을 수 있고, 다른 생물체를 잡아먹기 위해선 잡아먹으려는 생물체와 충돌해야 한다. 하지만 자신보다 크기가 큰 생물체와 충돌한 경우, 세포가 사망하게 된다.

생물체를 잡아 먹을 경우, 일반적으로 잡아먹은 생물체의 크기에 비례해 세포의 크기가 증가하지만 바이러스를 잡아 먹은 경우 바이러스의 크기에 비례해 세포의 크기가 감소한다. 바이러스는 특정한 패턴을 가지고 움직이며, 4초마다 크기가 감소하고 4회 크기가 감소하면 소멸하게 된다.

게임 종료 조건은 플레이어 세포가 크기가 0보다 작게 되어 사멸하는 경우와 세포의 크기가 커져서 화면 크기를 넘어버리는 경우가 있다. 앞의 것을 패배, 뒤의 것을 승리로 임의로 지정하기로 한다.

1. **Class 구성**

|  |  |
| --- | --- |
| **Bio : public QObject,**  **public QGraphicsRectItem** | **Cell : public Bio** |
| - int direction;  - int size; | - enum Direction{Up, Down, Left, Right};  - bool boost; |
| +Bio();  +virtual void Draw() = 0;  +virtual void Move() = 0;  +int getDirection();  +void setDirection(int direction);  +float getSize();  +void setSize(float size);  +void operator +(Bio\* bio);  +bool operator >(Bio\* bio); | +Cell();  +void keyPressEvent(QKeyEvent \*event);  **public slots:**  +void Draw();  +void Move();  +void Drain(); |
| **Enemy : public Bio** | **Feed : public Bio** |
| - enum Direction{Up, Down, Left, Right}; |  |
| +Enemy();  **public slots:**  +void Move();  +void ChangeDir();  +void Draw(); | +Feed();  +void Draw();  +void Move(); |

|  |  |
| --- | --- |
| **Virus : public Bio** | **MainWindow : public QMainWindow,**  **public QGraphicsRectItem** |
| - enum Direction{Right, Up, Left, Down};  - int shrinkNum; | - Ui::MainWindow \*ui;  - QGraphicsScene\* scene;  - QGraphicsView\* view;  - int totalNum;  - Bio\* bio[10000];  - Cell\* cell; |
| +Virus();  **public slots:**  +void Move();  +void Shrink();  +void Draw(); | +explicit MainWindow(QWidget \*parent = 0);  +~MainWindow();  +void Game();  +void GameOver();  +void GameWin();  **public slots:**  +void New();  **private slots:**  - void on\_pushButton\_2\_clicked();  - void on\_pushButton\_clicked(); |

모든 클래스에 대해서 생성자를 이용해 필요한 작업을 수행했다. Bio 클래스는 플레이어 클래스인 Cell을 포함해서 Enemy, Feed, Virus 클래스의 Base class 이고, UI 구현을 가능하게 하기 위해 QObject와 QGraphicsRectItem을 상속받는다. private 멤버 변수로는 한 변의 크기를 저장하는 size 변수와 방향을 저장하는 direction 변수가 있다. 이 둘에 대한 get, set 메소드가 존재한다. Bio 클래스에는 특히 Move() 와 Draw() 메소드가 있는데, 이들을 pure virtual method로 지정해 하위 클래스에서 구현하도록 했으며 동시에 > 와 + 연산자 오버로딩을 수행해 객체의 size를 비교하고 더하거나 뺄 수 있게 했다.

공통적으로 Bio를 상속받는 클래스는 Move()와 Draw()메소드를 구체적으로 정의해야 하는데, Feed 클래스는 아무것도 하지 않는다. 대부분 Move() 메소드의 경우 direction에 맞게 이동을 수행하고, Cell과 Enemy 클래스는 충돌을 추가로 확인해 작업을 수행한다. Draw() 메소드는 공통적으로 setRect 메소드를 이용해서 화면 내의 위치를 갱신하는 함수이다.

Cell 클래스는 boost 여부에 따라 size와 속도를 바꾸는 Drain() 메소드가 존재하고, Enemy 클래스는 무작위로 방향을 바꾸는 ChangeDir() 메소드가 존재하고, Virus 클래스는 일정 주기마다 size가 감소하고 일정 시간 수 소멸시키는 Shrink() 메소드가 존재한다.

MainWindow 클래스는 UI 구현의 중추가 되는 클래스이다. 초기 화면에서 버튼 클릭에 따른 행동을 지정하는 슬롯이 있고, 게임 종료 시 창을 띄워 메시지를 출력하는 메소드와 게임을 수행하는 Game() 메소드가 있다.

1. **소스 코드 및 프로그램 설명**

main 함수를 제외한 .cpp 파일에 대한 헤더 파일은 class의 정의뿐이고 기타 메소드의 구현은 .cpp 파일에 진행되어 있으므로 헤더 파일에 대한 설명은 생략하도록 한다. 또한 class 구현 파일에서의 get, set 메소드에 대한 설명은 자명하므로 생략한다. 기본적으로 클래스 별 size 범위, 이동 속력 등은 매뉴얼의 표에 있는 것을 기준으로 같게 구현했다. 단, 모양은 모두 직사각형이었다. 이유는 후술하겠다.

**<main.cpp>**

이전과는 다르게 main 함수에서 하는 일은 거의 없다. 그냥 srand 함수로 랜덤성을 부여하고 이후 MainWindow 클래스의 인스턴스를 하나 생성한 후 show() 메소드로 화면에 출력하면 끝이다. ui 파일은 우리가 수정할 수 없고 pro 파일도 설명하지 않겠다.

**<bio.cpp>**

bio.cpp에서는 > 와 + 연산자 오버로딩을 수행해 객체의 size를 비교하고 더하거나 뺄 수 있게 했다. + 연산자의 경우 void Bio::operator +(Bio\* bio) 로 오버로딩 했으며 bio의 타입이 Feed 혹은 Enemy인 경우에는 그 size를 this의 size에 더하도록 했고, Virus인 경우에는 그 size를 this의 size에서 빼기로 했다. 별도의 반환값은 존재하지 않는다. > 연산자의 경우 bool Bio::operator >(Bio\* bio) 으로 오버로딩 했다. bio 객체와의 size를 비교해 this의 size가 더 큰 경우 true를, 같거나 작은 경우 false를 반환하도록 오버로딩했다. bio 클래스에서 오버로딩을 한 이유는 구현에 있어서의 편의성 때문이다.

**<cell.cpp>**

생성자에서는 먼저 고정된 size와 위치를 지정한다. Draw 메소드를 한 번 호출하고 방향을 random으로 지정한 뒤 boost가 처음에는 없도록 한다. 이후 QTimer와 connect를 이용해 일정 시간 간격으로 할 행동을 지정했다. 50ms마다 Move()를 호출해 이동하고, 500ms 마다 Drain()을 호출해 boost 여부에 따라 상황을 바꾸고, 50ms마다 Draw를 호출해 화면에 인스턴스의 상황을 호출한다.

Draw() 메소드는 객체의 위치와 size를 지정하도록 하는 멤버 메소드이다.

Move() 메소드는 boost 상태일때와 아닐 때를 나누어, 현재 객체의 direction에 따라 이동 방향에 맞게 이동한다. 단, 화면의 끝에 도달한 경우는 화면을 벗어나는 방향으로 움직일 수 없게 하였다. 또한 추가로 충돌 여부를 판단한다. Enemy 인스턴스와 충돌한 경우 크기를 비교해 cell이 더 크면 + 연산자로 합치고 충돌한 인스턴스를 제거한다. 그렇지 않은 경우에는 cell을 제거하고 게임오버 메시지 창을 띄운 뒤 프로그램을 바로 종료한다. Feed 인스턴스와 충돌한 경우는 비교 없이 바로 + 연산자를 수행해 합치고 충돌한 Feed 인스턴스를 제거한다. Virus 인스턴스와 충돌한 경우는 역시 비교 없이 + 연산자를 수행해 합친다. 이후 cell의 size를 확인해 0 이하이면 같은 방법으로 게임을 종료한다.

Drain() 인스턴스는 boost 여부를 확인해 boost 상태인 경우 size를 0.95배 감소시킨다.

keyPressEvent() 인스턴스는 사용자의 키 입력을 인식해 방향키의 방향에 따라서 cell의 direction을 바꿔주고, space 키를 누른 경우 boost의 상태를 전환한다.

**<enemy.cpp>**

생성자의 경우, cell의 그것과 거의 같지만 몇 가지 부분에서 차이를 띤다. 먼저 size를 정해진 범위 내에서 랜덤으로 지정했고, 위치 역시 임의의 랜덤한 위치에서 생성되도록 했다. 또한 Cell의 Drain() 메소드 대신 ChangeDir 메소드를 QTimer와 connect를 이용해 3초마다 호출되게 해 3초마다 방향이 바뀌게 했다.

Draw() 메소드는 Cell의 그것과 같다.

Move() 메소드는 Cell의 그것과 이동 속도만 다르게 하여 설정했고, Cell 클래스와 마찬가지로 충돌 시 작업을 수행한다. 단, 이 경우 this 의 클래스가 Enemy이기 때문에 this의 size가 0 이하가 된다 해도 화면에서는 사라지지만 게임이 종료되지는 않는다.

ChangeDir() 메소드는 random하게 direction을 바꿔주는 메소드이다.

**<feed.cpp>**

생성자에서는 굉장히 간단한 작업만 하고 끝난다. 이동하지 않으므로 direction을 임의로 설정한다. size 와 위치는 random으로 지정한다. Size는 정해진 범위 내로 지정되도록 구현한다. 이후 화면에 setRect 메소드를 이용해 배치하면 끝난다.

Draw() 와 Move() 메소드는 별다른 작업 없이 return 한다.

**<virus.cpp>**

생성자의 경우, enemy의 그것과 거의 같다. size를 정해진 범위 내에서 랜덤으로 지정했고, 위치 역시 임의의 랜덤한 위치에서 생성되도록 했다. shrinkNum을 처음에는 0으로 설정했다. 또한 Enemy의 ChangeDir() 메소드 대신 Shrink() 메소드를 QTimer와 connect를 이용해 4초마다 호출되게 해 크기를 줄이거나 4번 크기가 줄은 경우 객체를 소멸시키도록 구현했으며, Move()의 호출 주기를 1초로 바꾸었다.

Draw() 메소드는 Cell의 그것과 같다.

Move() 메소드는 Cell의 그것과 한 번에 이동하는 거리만 다르게 하여 설정했고 Cell과 Enemy 클래스와는 다르게 충돌을 검사하지 않는다.

Shrink() 메소드는 shrinkNum이 4인 경우 객체를 소멸시키고, 그렇지 않은 경우는 size를 0.75배 한 것으로 바꾼다.

**<mainwindow.cpp>**

이 파일에서 실질적인 게임 구현이 일어난다.

먼저 생성자에서는 ui를 설정하고 QGraphicsScene과 QGraphicsView의 포인터 변수인 scene과 view를 동적할당 받는다. 이후 화면에 출력될 옵션을 지정한다. 소멸자에서는 ui를 제거한다.

Game() 메소드가 실제 게임이 구현되는 영역이다. 먼저 cell 인스턴스를 동적할당 받고, flag와 focus 지정을 통해 키를 입력받을 수 있게 하고 색을 지정한다. 또한 게임 시작 때 존재해야 하는 30개의 bio 인스턴스를 매뉴얼에 정해진 확률대로 Enemy, Virus, Feed가 생성되게 하고, 동적할당으로 만든다. 원래는 이 과정에서 cell의 영역과 겹치지 않도록 설정해야 하지만, 다양한 방법을 시도한 끝에 실패했다. 하지만 확률적으로 cell과 겹치게 생성될 확률은 극히 낮아서 실제 게임을 진행하는 데에 있어서 심각한 수준의 문제는 되지 않았다. 이후 현재까지 생성된 bio의 객체의 수를 표현하는 totalNum의 값을 30으로 바꾸고, scene에 bio 인스턴스들과 cell 인스턴스를 추가한다. 이수 QTimer를 이용해 5초마다 New() 인스턴스를 이용해 5개씩 새로운 bio 인스턴스가 화면에 생성되도록 한다.

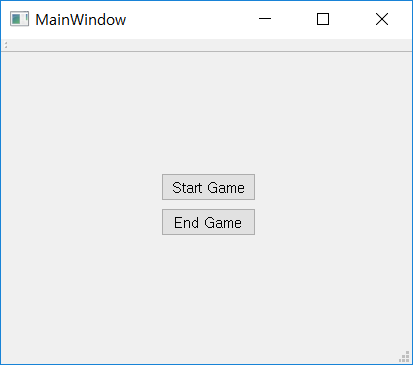
New() 메소드는 5개씩 bio 인스턴스를 랜덤하게 생성하는 과정이다. Game() 메소드에서 처음 30개씩 확률에 따라 동적할당을 진행하던 것을 5개씩 이어서 동적할당한다고 생각하면 편하다. 새로 생성된 인스턴스를 scene에 추가하고, totalNum을 5를 더한다. 역시 cell의 위치와 겹치지 않게 구현하는 것은 실패했다.

on\_pushButton\_clicked() 메소드는 맨 처음 초기 화면에서 end game 버튼을 눌렀을 때 할 일을 나타낸다. 메시지 박스와 함께 프로그램을 종료하게 된다.

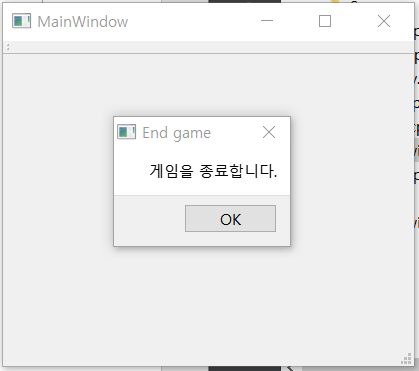
on\_pushButton\_2\_clicked() 메소드는 초기 화면에서 start game 버튼을 눌렀을 때 할 일을 나타낸다.

GameOver()와 GameWin() 메소드는 cell이 사멸하거나 화면을 덮은 경우 게임을 종료하도록 하는 메소드이다. 각각 메시지박스를 출력한다.

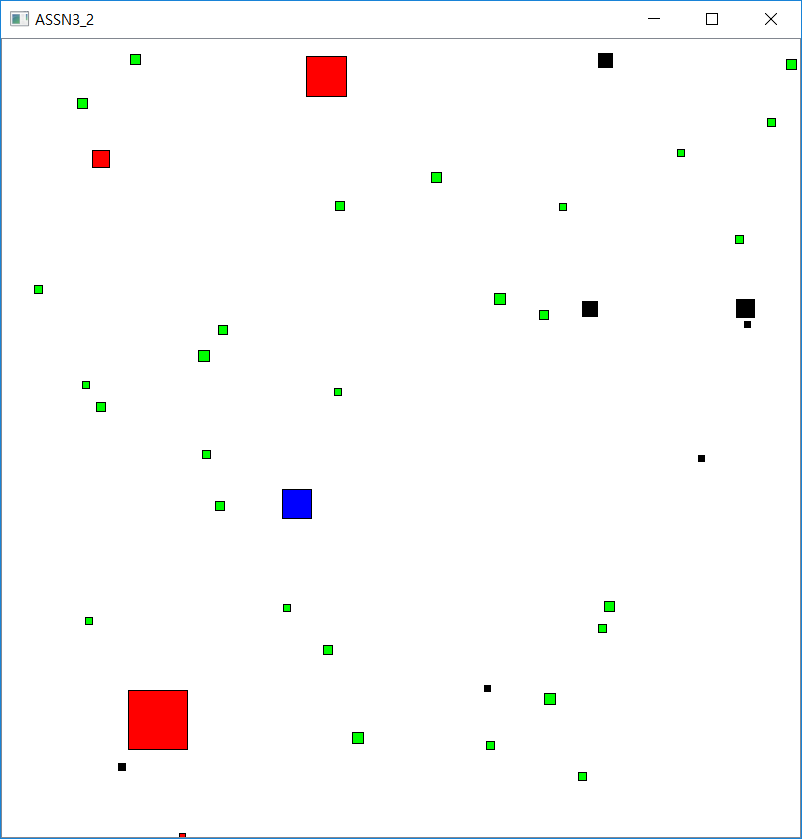
1. **프로그램 실행**



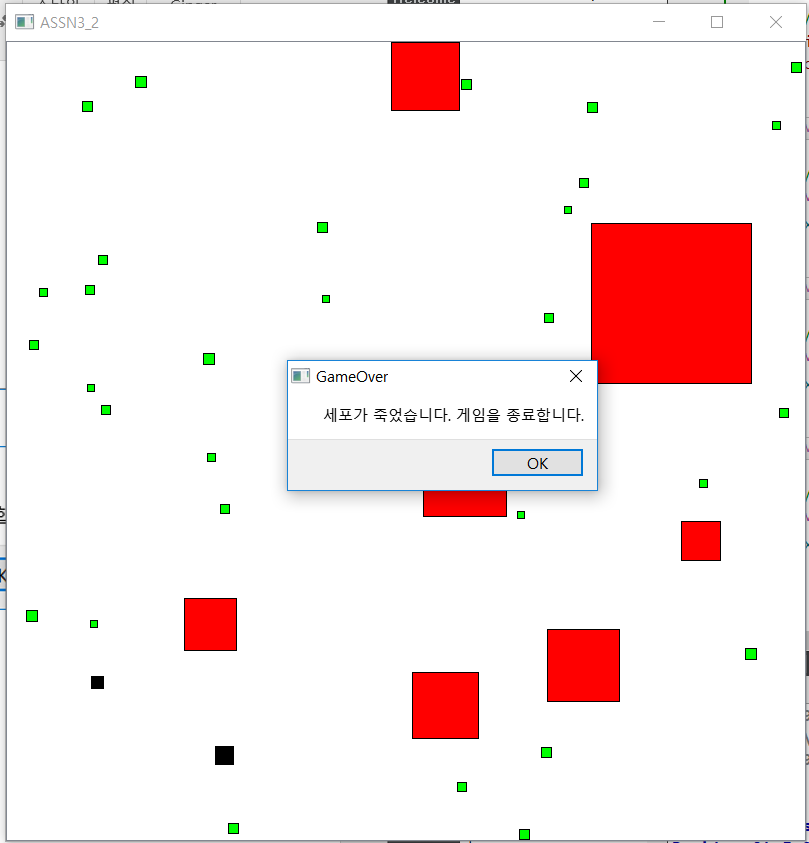
<프로그램을 처음 실행하면 나오는 화면>



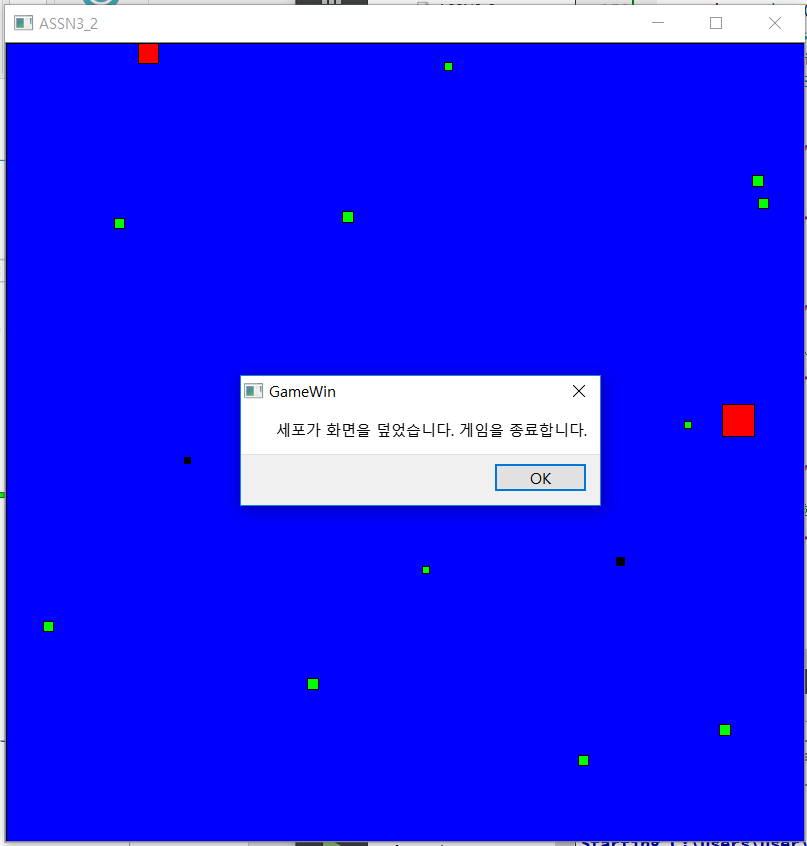
<End Game 버튼을 클릭하면 나오는 메시지박스. OK를 누르면 게임이 종료된다.>



<Play Game을 누르면 나타나는 화면>



<세포가 size가 0이하가 되어 사멸한 경우. OK를 누르면 프로그램이 종료된다.>



<세포가 화면을 덮어(size가 화면보다 커서) 게임이 종료된 모습.>

1. **토론**

* 구현에 성공한 내용이 있고, 실패한 내용이 있다.
* 먼저 클래스 중 Cell와 Enemy는 원, 즉 QGraphicsEllipseItem 클래스를 상속받아서 모양이 원이 되도록 구현해야 했지만, 다양한 문제가 생겼다. 먼저 bio의 하위 인스턴스에 각각 도형에 맞는 클래스를 상속받으면 bio는 모호성을 방지하기 위해 그 클래스 혹은 QGraphicsItem 클래스를 상속받지 못한다. 그러나 이 경우 bio 인스턴스를 scene에 추가하지 못하는 사태가 발생한다. 즉 다이아몬드 상속을 받아서 모호성을 제거해야 했는데, 이 과정에서 실패가 너무 많이 일어나 그냥 게임 진행에 있어서 심각한 문제가 되지 않는다고 판단하여 구현을 포기했다. 그래서 bio 클래스에서 QGraphicsRectItem을 상속받아 모든 하위 객체가 정사각형이 되도록 구현했다. 하지만 색을 다르게 하는 부분은 성공적으로 구현할 수 있었고, 색을 통해서 각 객체의 종류를 구분할 수 있기 때문에 게임 진행에 있어서 도형이 지대한 영향을 미치지는 않는다.
* 다음으로 플레이어 cell 인스턴스와 겹치지 않게 다른 bio 인스턴스를 생성해야 하는데, 다양한 방법을 시도했으나 cell의 바깥쪽에 생기기는커녕 오히려 세포가 커진 이후에 문제가 발생하여 프로그램의 안정성을 위해서 구현을 포기했다. 다만 화면의 크기와 각 인스턴스의 size를 생각했을 때, 따로 처리를 하지 않아도 cell 인스턴스와 생성 시 겹치는 일은 거의 발생하지 않으며 때문에 게임 구현에 있어서는 치명적인 문제가 될 것이라고는 생각하지 않아 그대로 포기하고 구현했다. 만약 겹치는 위치에 생성된다 해도 상응하는 상호작용을 통해서 게임이 진행될 것이기 때문에 겹쳐도 큰 문제는 없다.
* 다만, 이러한 게임에 있어서 부가적인 구현을 제외한 연산자 오버로딩이나 pure virtual method와 같은 다형성 개념의 적용에 있어서는 성공적이었다.
* 이전 프로그래밍과는 다르게 UI를 추가로 도입하니 콘솔 창이 주된 출력장치가 아닌 검사 및 확인용으로 바뀌는 것을 알게 되었다.
* 잔에러가 많이 발생해서 고생했다. 하지만 구글링을 통해서 대부분 슬기롭게 해결할 수 있었다.

1. **결론**

* Qt Creator를 이용해 UI가 적용된 게임을 구현할 수 있었다.
* 연산자 오버로딩, virtual method와 같은 객체지향프로그래밍에 있어서 다형성 개념이 적용될 수 있는 부분에 대해서는 구현을 성공적으로 진행했다.
* 상속 개념을 이용해 게임에 필요한 클래스 구현을 할 수 있었다.
* 클래스에 따라서 출력되는 도형의 모양을 다르게 하거나, 새로운 인스턴스 생성 시 플레이어 cell과 겹치지 않게 구현하는 데에는 실패했지만, 게임 진행에 있어서 크게 영향을 미치는 내용은 아니었다.

1. **개선 방향**

* 클래스별 인스턴스의 도형을 다르게 하도록 개선할 수 있다.
* 새로운 인스턴스 생성 시 플레이어 cell과 겹치지 않게 개선할 수 있다.
* 게임이 끝나면 처음 화면으로 돌아가 새로운 게임을 시작하도록 구현할 수 있다.