객체지향프로그래밍
QT-Project 보고서

2021202085 전한아솔

먼저 3개의 게임에서 공통적으로 적용되는 부분을 설명하자면, Qt Creator에서 프로그램을 실행하면 테트리스, 뿌요뿌요, 뿌요테트리스 중 하나를 선택하여 게임을 실행할 수 있어야 하며 게임이 종료되었을 때 완전히 정지되고, 사용자에게 게임이 종료되었음을 알려야 한다.
모든 게임에서 유닛(뿌요 or 블록) 빈 공간과 서로 다르게 시각화되어야 한다.
모든 게임에서 방향키로 현재 유닛을 움직일 수 있으며 1초가 지나면 한 칸씩 밑으로 떨어진다. 또한 스페이스바를 누르게 되면 Hard drop하여 더 이상 내려갈 수 없을 때까지 밑으로 내려간다.
모든 유닛은 x키를 눌러서 시계방향으로 회전하고 z키를 눌러서 반시계 방향으로 회전한다. 이때 다른 유닛에 충돌하거나 게임 보드를 벗어나게 되는 상황이라면 회전하지 않는다.
유닛의 조작이 완전히 종료되고 더 이상 제거할 수 있는 유닛이 없고 뿌요뿌요나 뿌요테트리스의 경우에 중력도 적용이 완료되었을 때 다음 유닛이 생성되어야 한다.
각 게임마다 달라지는 게임 판의 열 크기, 행 크기, 유닛의 크기는 각 file에서 전역변수로 선언했다.

<테트리스>

1. Introduntion

테트리스란 [20][10] 크기의 게임판에 7개의 무작위 블록 모양 중 하나가 생성되어 낙하한다. 게임판에 한 줄이 전부 테트로미노로 채워지게 되며 게임판 전체에 중력이 작용한다. 예를 들어 맨 밑 한 줄이 테트로미노로 가득 차게 되어서 지워진다면 그 위에 있던 모든 테트로미노가 한 칸 밑으로 내려오게 된다.

2. Algorithm

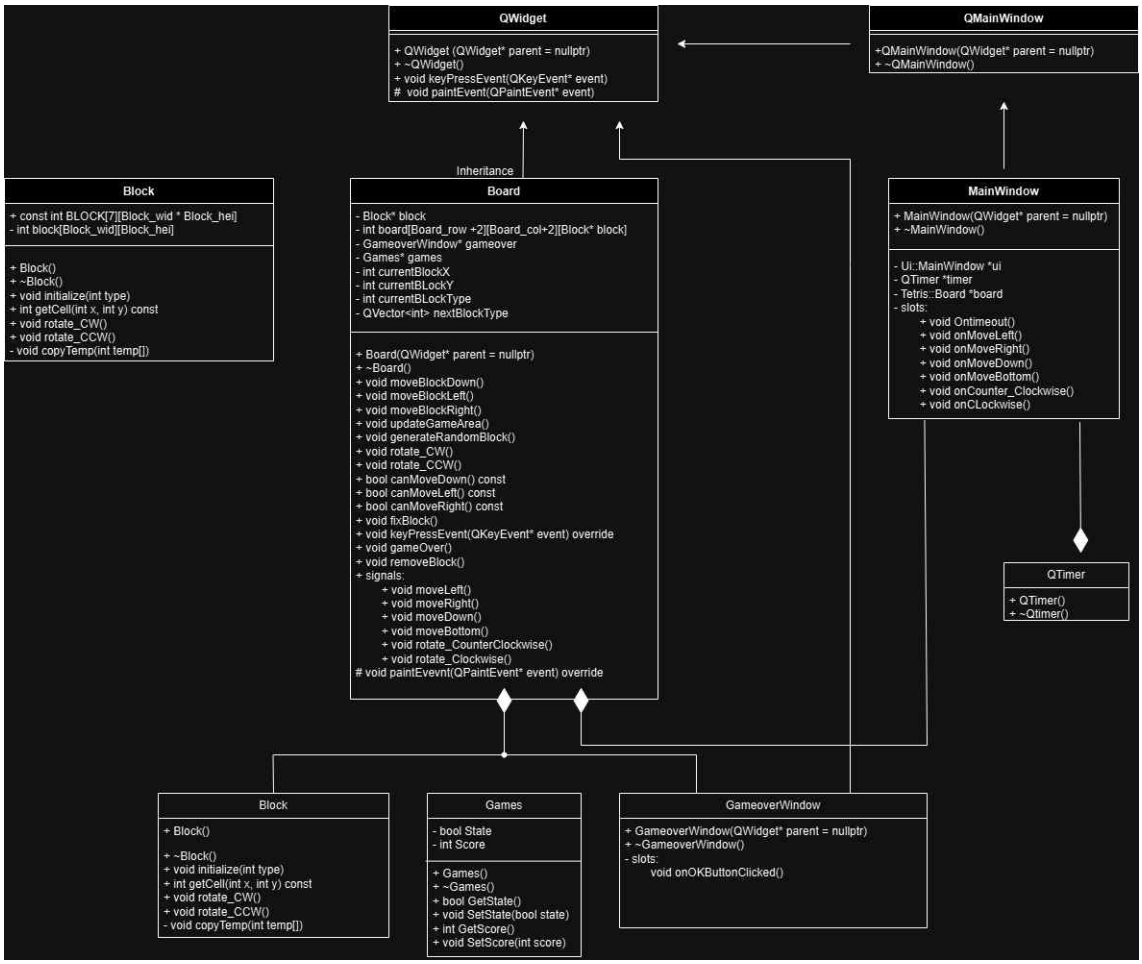
테트리스 게임 판의 열(Board_col)은 10, 행(Board_row)은 20이고 블록이 저장된 형태는 4x4이므로 블록의 너비(Block_wid)와 높이(Block_hei)는 4로 저장하여 전역변수로 선언했다.
게임을 시작하게 되면 MainWindow 클래스의 생성자가 호출되는데, 이 MainWindow의 생성자에서는 Board 클래스의 생성자를 호출하고 Board의 생성자는 Block의 생성자, Games의 생성자, GameoverWindow의 생성자를 호출한다. 따라서 Block, Games, GameoverWindow, Board, MainWindow의 순서대로 생성자가 호출되며, Board의 생성자에서는 랜덤으로 블록을 생성하는 함수를 호출하여 7개의 블록 중 하나를 랜덤으로 생성한다. MainWindow의 생성자에서는 타이머를 시작하는데 타이머가 끝난다면(1초) signal을 발생시켜 signal과 연결된 slot이 생성되고, 이 slot에서는 테트로미노를 한 칸 내리는 함수가 실행된다. 따라서 1초마다 테트로미노는 한 칸씩 내려간다.

<Block>

테트로미노는 4x4의 2차원 형태이므로 1차원으로 저장 시, 16칸이 된다. 따라서 행은 7, 열은 16인 2차원 int형 배열에 랜덤 블록 7개가 1차원에 형태로 block 클래스에 정의되어 있다.
4x4의 칸에서 빈칸은 0, 블록이 존재하는 칸은 2로 설정했다.

block 클래스에 멤버 함수로는 현재 블록을 시계·반시계 방향으로 회전하여 저장하는 함수, 4x4의 2차원 칸의 인덱스를 반환하는 함수, 인자로 type을 전달받아 해당 type으로 현재 블록을 초기화하는 함수가 존재한다. 여기서 type이란 랜덤 블록들 중 하나를 의미한다.
<Board>
테트리스의 게임 판은 10x20의 형태이지만 빈칸은 0, 테두리는 1로 설정했다. 따라서 테두리의 크기까지 고려하여 게임판을 의미하는 2차원 배열은 [Board_col+2][Board_row+2]의 크기로 선언했다.
멤버 변수로는 게임판을 의미하는 board, Block 클래스의 객체인 block, 현재 상태를 의미하는 클래스(Games)의 객체인 games, 게임이 종료되면 종료 화면을 띄우는 클래스(GameoverWindow)의 객체인 gameover, 현재 블록의 좌표를 가리키는 int형 변수 currentBlockX, currentBlockY, 랜덤 블록 중 현재 블록의 타입을 저장하는 int형 변수 currentBlockType, 다음 생성될 블록들의 타입을 가지는 int형 벡터 nextBlockType이 있다. 여기서 블록의 좌표가 의미하는 것은 4x4 크기의 블록이 저장 되어있는 2차원 칸의 좌측 상단 모서리를 의미한다.
멤버 함수는 블록을 아래, 왼쪽, 오른쪽으로 이동시키는 함수, 아래, 왼쪽, 오른쪽으로 이동 가능한지 판단하는 함수, 현재 게임판과 블록을 그리는 함수, 랜덤으로 블록을 생성하는 함수, 시계·반시계 방향으로 회전하는 함수, 블록을 고정하는 함수, 블록을 제거하는 함수, 게임 종료를 판단하는 함수 등등이 있다. Board 클래스의 회전 함수는 게임판 내에서 회전 시에 다른 테트로미노와 충돌이 없는가, 게임판을 벗어나지 않는가 등을 판단하고 가능 여부에 따라서 Block 클래스의 회전 함수를 호출한다.
그리고 QT에는 특이한 요소인 signal과 slot이 존재하는데, signal은 이벤트가 발생하면 신호를 보내는 것이고 slot은 이 signal을 전달받아 함수를 호출하는 것이다.
Board 클래스는 QWidget 클래스를 상속받는데, 게임판과 블록을 그리는 함수와 키가 눌러지면 그 키에 맞는 signal을 보내는 함수가 QWidget 클래스에 존재한다. 따라서 오버라이딩을 통해서 테트리스 게임의 조건에 맞게 수정했다. 키를 누르면 signal이 전달되어야 하므로 Board에는 signal이 존재하는데, 이는 방향키에 따라 블록을 움직이고 Z를 누르면 반시계 방향 회전, X를 누르면 시계 방향 회전을 실행할 때 사용된다.
<MainWindow>
MainWindow 클래스의 멤버로는 타이머를 동작하기 위한 QTimer 객체, Board 클래스의 함수를 호출하여 블록을 동작해야 하므로 Board 객체가 존재하고 Board 클래스에서 전달되는 signal을 MainWindow 클래스에서 slot으로 전달하기 위해서 slot을 정의하고 connect를 이용하여 생성자에서 signal과 slot을 연결한다. 또한 타이머가 종료되면 발생하는 signal과 slot을 연결한다. slot에서는 signal에 맞는 Board 클래스의 함수를 호출한다. 예를 들어 X키가 눌리면 발생되는 signal과 slot을 연결하고 이 slot에서는 Board 클래스의 시계방향 회전 함수를 호출한다. 테트리스 파일의 signal과 MainWindow 파일의 slot을 MainWindow 생성자에서 연결하므로 MainWindow.h 파일은 테트리스의 헤더파일을 포함한다.

3. UML Diagram



전체적으로 크게 보자면 테트로미노에 관한 클래스인 Block, 게임판과 게임판에서 작용하는 모든 행동에 관한 클래스인 Board, 현재 게임의 상태, 점수를 관리하는 클래스인 Games, 게임이 종료되었을 때 게임 종료 화면을 출력하는 클래스인 GameoverWindow, Board에서 발생한 시그널을 받아서 내가 원하는 함수를 호출, 즉 내가 원하는 행동을 하도록 지시하거나 1초가 지날 때마다 시그널을 발생시켜 테트로미노가 한 칸 내려가도록 하는 클래스인 MainWindow로 구성된다.

+는 public, -는 private, #은 protected를 의미한다.

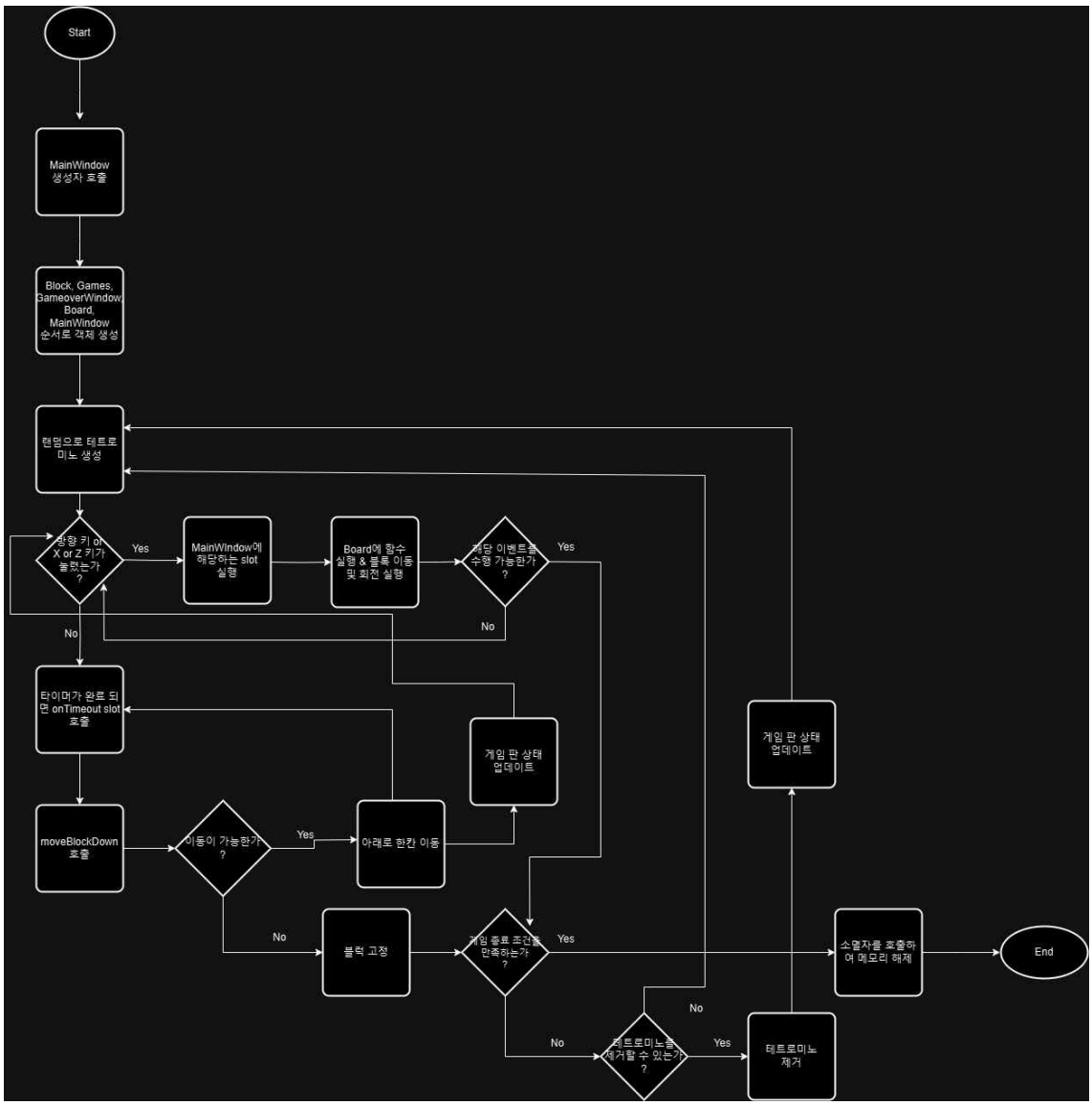
메인 클래스인 Block, Board, MainWindow 클래스를 보면 private에는 멤버 변수를, public에는 멤버함수를 포함하여, 클래스의 외부에서 각 클래스 변수에 직접 접근하여 값을 변경하지 못하도록 구현했다.

Block 클래스에는 랜덤 블록의 모양을 1차원 형태로 저장한 BLOCKS 변수는 public으로 선언되어 있는데, Board 클래스에서 BLOCKS에 접근하도록 구현했기 때문이다. 하지만 BLOCKS에 접근은 할 수 있어도 블록의 모양을 바꾸면 안되기 때문에 const 키워드를 붙였다.

그리고 멤버 함수인 CopyTemp는 인자로 전달받은 포인터에 현재 블록을 복사하는 함수인데, public으로 선언하게 되면 외부에서 Block 클래스의 블록을 가져올 수 있으므로 private

함수로 선언하였다. 이를 통해서 객체지향의 특성 중 캡슐화를 적용했음을 알 수 있다.
Board 클래스는 QWidget 클래스를 public으로 상속하여 QWidget 클래스의 keyPressEvent 함수나 paintEvent 함수를 함수 오버라이딩을 통해 재정의하여 사용할 수 있도록 했다. 이를 통해서 객체지향의 특성 중 상속과 다형성을 적용했음을 알 수 있다. 또한 Board 클래스에서도 마찬가지로 멤버 변수는 클래스 외부에서 접근하지 못하도록 private으로 선언하였고 멤버 함수는 public으로 선언했다.
Block 클래스와 Board 클래스, 전역변수는 모든 Tetris라는 이름 공간에 선언되도록 구현하여 나중에 뿌요뿌요나 뿌요테트리스에서 같은 이름을 사용하는 함수라도 혼동되지 않도록 하였다.

4. Flowchart



5. pseudocode

<Block>
인자로 전달받은 타입으로 블록을 초기화하는 함수:
	for(i는 0부터 블록 너비 -1까지)
		for(j는 0부터 블록 높이 -1까지)
			1차원으로 저장된 랜덤 블록 중 인자로 전달된 타입에 따라 하나를
			2차원인 현재 블록에 복사
		end for
	end for

시계방향 회전:
	회전될 블록이 저장될 임시 2차원 배열 선언&현재 블록으로 초기화
	for(블록이 저장된 4x4의 칸에 행을 0부터 반복)
		for(4x4칸에 열을 0부터 반복)
			임시 2차원 배열은 왼쪽 상단부터 열을 증가시키며(오른쪽으로)
			반복
			현재 블록에 오른쪽 상단부터 행을 증가시키며(아래쪽으로) 반복
			현재 블록에 임시 2차원 배열에 저장된 값을 저장
		end for
	end for

반시계방향 회전:
	회전될 블록이 저장될 임시 2차원 배열 선언&현재 블록으로 초기화
	for(블록이 저장된 4x4의 칸에 행을 0부터 반복)
		for(4x4칸에 열을 0부터 반복)
			임시 2차원 배열은 왼쪽 상단부터 열을 증가시키며(오른쪽으로)
			반복
			현재 블록에 왼쪽 하단부터 행을 감소시키며(위쪽으로) 반복
			현재 블록에 임시 2차원 배열에 저장된 값을 저장
		end for
	end for
<Board>
랜덤으로 블록을 생성하는 함수:
	if(게임 종료 상태가 아니라면)
		다음 블록 타입을 저장하는 벡터에서 하나 가져옴
		다음 블록 타입을 저장하는 벡터에 0~6의 랜덤 값을 삽입
		블록의 처음 x좌표는 4, y좌표는 0으로 초기화
		block 객체의 해당 타입으로 블록을 초기화하는 함수 호출
	end if

아래로 한 칸 움직이는 함수:
	if(아래로 움직일 수 있는가)

```
            y좌표 1 증가
        else
            블록 고정
            게임 종료를 판단하는 함수 호출
            블록을 제거하는 함수 호출
            새로운 블록을 랜덤 생성하는 함수 호출
        게임판 업데이트 함수 호출

오른쪽으로 한 칸 움직이는 함수:
        if(오른쪽으로 움직일 수 있는가)
            x좌표 1 증가
        end if
        게임판 업데이트 함수 호출
 왼쪽으로 한 칸 움직이는 함수:
        if(왼쪽으로 움직일 수 있는가)
            x좌표 1 감소
        end if
        게임판 업데이트 함수

아래로 움직일 수 있는지 판단하는 함수:
        for(i는 0부터 블록의 높이까지)
            for(j는 0부터 블록의 너비까지)
                if(4x4칸에 블록이 존재하는 곳이라면)
                    새로운 x좌표=현재 x좌표+j
                    새로운 y좌표는=현재 y좌표+i+1
                    if(새로운 y좌표가 보드의 행의 크기+1 이상이거나 새로운
                    x, y좌표에 해당하는 게임판에 다른 블록이 있다면)
                        움직일 수 없음
                    end if
                end if
            end for
        end for
        움직일 수 있음

오른쪽으로 움직일 수 있는지 판단하는 함수:
        for(i는 0부터 블록의 높이까지)
            for(j는 0부터 블록의 너비까지)
                if(4x4칸에 블록이 존재하는 곳이라면)
                    새로운 x좌표=현재 x좌표+j+1
                    새로운 y좌표=현재 y좌표+i
                    if(새로운 x좌표가 보드의 열의 크기+1 이상이거나 새로운
```

```
                        x, y좌표에 해당하는 게임판에 다른 블록이 있다면)
                                움직일 수 없음
                        end if
                end if
        end for
end for
움직일 수 있음
```

왼쪽으로 움직일 수 있는지 판단하는 함수:

```
for(i는 0부터 블록의 높이까지)
        for(j는 0부터 블록의 너비까지)
                if(4x4칸에 블록이 존재하는 곳이라면)
                        새로운 x좌표=현재 x좌표+j-1
                        새로운 y좌표=현재 y좌표+i
                        if(새로운 x좌표가 0보다 작거나 새로운
                        x, y좌표에 해당하는 게임판에 다른 블록이 있다면)
                                움직일 수 없음
                        end if
                end if
        end for
end for
움직일 수 있음
```

게임판에 그림을 그리는 함수:

```
//현재 보드의 상태를 그림
for(1부터 게임판의 행 크기까지)
        for(1부터 게임판의 열 크기까지)
                if(게임판의 인덱스가 0이라면)
                        테두리가 검은색인 하얀색 사각형을 그림
                else if(게임판의 인덱스가 2라면)
                        테두리가 검은색인 회색 사각형을 그림
        end for
end for

//현재 움직이는 블록을 그림
for(0부터 블록의 높이-1 까지)
        for(0부터 블록의 너비-1 까지)
                if(4x4칸의 인덱스가 2라면)
                        테두리가 검은색인 회색 사각형을 그림

                end if
```

```
        end for
    end for

    //다음에 나올 블록 5개를 그림
    보드의 오른쪽 칸에 출력하도록 위치 조정
    for(다음 블록의 타입이 저장된 벡터의 크기만큼 반복)
        벡터의 블록 타입 중 하나를 저장
        저장한 타입의 블록으로 임시 1차원 배열을 초기화
        for(블록의 높이만큼 반복)
            for(블록의 너비만큼 반복)
                if(임시 1차원 배열에 블록이 존재하는 칸이라면)
                    테두리가 검은색이 회색 사각형을 그림
                end if
            end for
        end for
    end for

시계방향 회전:
    block 객체의 시계방향 회전 함수 호출
    for(i는 0부터 블록의 높이까지)
        for(j는 0부터 블록의 너비까지)
            임시 2차원 배열에 회전시킨 블록을 저장
        end for
    end for
    for(i는 0부터 블록의 높이까지)
        for(j는 0부터 블록의 너비까지)
            if(4x4칸에 블록이 존재하는 곳만)
                if(게임판[현재 y좌표 + i][현재 x좌표 + j]에 다른 블록이
                존재하거나 판을 넘어가면)
                    회전할 수 없음
                    block 객체의 반시계방향 회전 함수 호출
                end if
            end if
        end for
    end for

반시계방향 회전:
    block 객체의 반시계방향 회전 함수 호출
        for(i는 0부터 블록의 높이까지)
            for(j는 0부터 블록의 너비까지)
                임시 2차원 배열에 회전시킨 블록을 저장
```

```
                end for
            end for
        for(i는 0부터 블록의 높이까지)
            for(j는 0부터 블록의 너비까지)
                if(4x4칸에 블록이 존재하는 곳만)
                    if(게임판[현재 y좌표 + i][현재 x좌표 + j]에 다른 블록이
                    존재하거나 판을 넘어가면)
                        회전할 수 없음
                        block 객체의 시계방향 회전 함수 호출
                    end if
                end if
            end for
        end for

키보드 입력에 따라서 signal을 발생시키는 함수:
        swtich(키 입력)
        case 방향키 왼쪽:
            왼쪽 이동 signal 발생
        case 방향키 오른쪽:
            오른쪽 이동 signal 발생
        case 방향키 아래쪽:
            아래쪽 이동 signal 발생
        case 스페이스바:
            아래쪽으로 최대한 이동 signal 발생
        case Z키:
            반시계방향 회전 signal 발생
        case X키:
            시계방향 회전 signal 발생
        나머지:
            아무것도 반환 X

블록을 지우는 함수:
        for(i는 게임판 행의 크기부터 1까지)
            for(j는 1부터 게임판 열의 크기까지)
                if(게임판[i][j]에 빈칸이 하나라도 있다면)
                    블록을 지울 수 없음
                    break
                end if
            end for
            if(i행이 모두 채워져 있다면)
            for(k는 i부터 2까지)
```

```
                    for(l은 1부터 보드의 열 크기까지)
                    k-1번째 행을 k번째 행에 저장
                    end for
            end for
            for(m은 1부터 보드의 열 크기까지)
                    1행의 m열을 모두 빈칸으로 초기화
            end for
            한 줄이 삭제되면 윗줄이 내려오므로 한번 더 검사 진행 -> i는 1증가
    end for
```

게임 종료를 판단하는 함수:

```
    for(i는 1부터 보드의 열 크기까지)
            if(1행 i열에 블록이 하나라도 있다면)
                    게임 종료
                    break;
            end if
    end for
    if(게임 종료 조건을 만족한다면)
            게임 종료 화면 실행
            현재 게임 상태를 종료로 바꿈
    end if
```

6. 고찰

테트리스에는 7개의 랜덤 블록이 있는데, 블록을 1차원의 형태로 저장해 두었다. 하지만 직접 게임판에서 블록은 컨트롤할 때 , 4x4크기의 2차원 배열로 생각해서 움직인다. 1차원의 블록을 2차원으로 옮겨 복사하고, 2차원의 블록을 1차원의 형태로 옮기는 과정에 있어서 굉장히 헷갈리는 부분이 많았다. 대부분의 2중 for문의 형태는 I는 행, j는 열을 가리키도록 구현했는데, I+블록의 높이*j의 형태로 작성하면, 0~15의 수를 순차적으로 나타낼 수 있다. 이 방법을 통해서 1차원과 2차원의 블록을 다루었다.

7. 동영상 & 검증 시나리오

https://youtu.be/cpGrB99OrLk

동영상에서 볼 수 있듯, 오른쪽 부분에는 앞으로 나올 5개의 랜덤 블록의 형태가 보여지고 있다. 각 블록은 생성 후 1초마다 타이머가 작동하여 무조건 아래로 하강한다. 오른쪽, 왼쪽 아래쪽 방향키를 누르면 해당 방향으로 이동이 가능할 시에, 이동한다. 또한 스페이스바를 누르면 하드 드롭이 동작하도록 구현했고 Z키를 누르면 반시계 방향으로 회전, X키를 누르면 시계 방향으로 회전함을 볼 수 있다. 블록이 맨 밑의 한 줄을 모두 채웠을 때, 한 줄이 없어지고 그 위의 블록이 모두 한 줄 내려오는 모습도 담아냈다. 마지막으로 테트리스의 종료 조건인 맨 윗줄에 하나라도 블록이 있음을 만족하면 'Gameover'라는 메시지가 나타나며 사용자에게 게임이 종료되었음을 알리고 OK버튼을 누르면 게임이 종료되도록 구현했다.
또한 점수는 한 번에 몇 줄을 지우던 1점이 추가되도록 구현했다.

<뿌요뿌요>

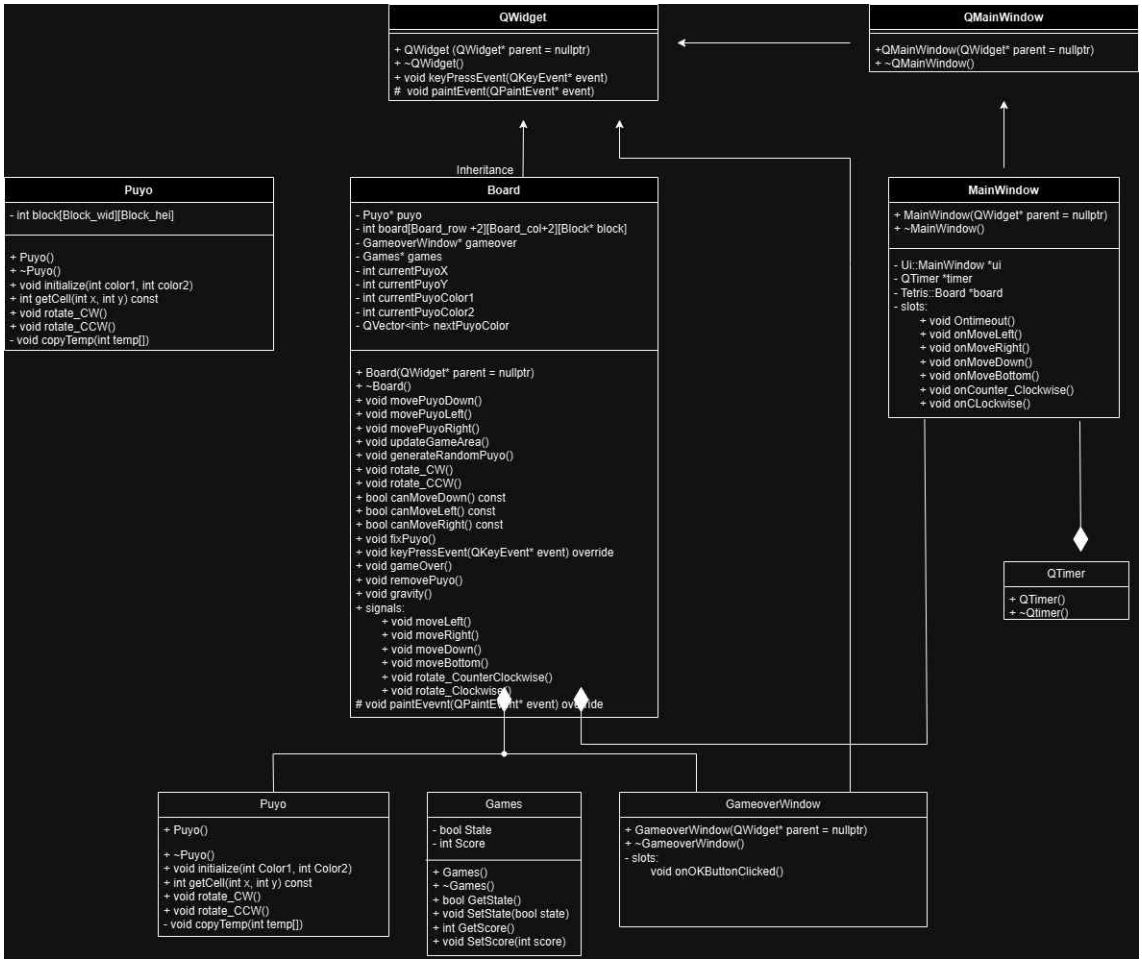
1. Introduction

뿌요뿌요는 [12][6] 크기의 게임 판에 뿌요의 색상인 빨강, 노랑, 초록, 파랑, 보라색이 랜덤으로 하나씩 들어있는 2개의 블록이 붙어서 생성된다. 즉, 25개의 랜덤한 뿌요가 생성되는 게임이다. 이 게임도 테트리스와 마찬가지로 회전, 하강, 오른쪽·왼쪽으로 이동할 수 있다. 뿌요뿌요와 테트리스의 다른 점으로는 중력의 작용 방식이 있다. 뿌요뿌요는 뿌요가 생성되고 최대한으로 내려가게 되면 중력이 작용한다. 따라서 2개로 구성된 뿌요 중 하나의 밑에 공간이 존재한다면 따로 분리되어 내려갈 수 있다. 테트리스와 다른 또 하나의 요소는 뿌요의 제거인데, 색상이 같은 뿌요가 4개 이상 붙어서 존재하면 뿌요는 제거될 수 있다. 이 때 대각선은 붙어있다고 판단하지 않는다. 이렇게 제거가 이루어진 후에도 중력이 작용한다.

2. Algorithm

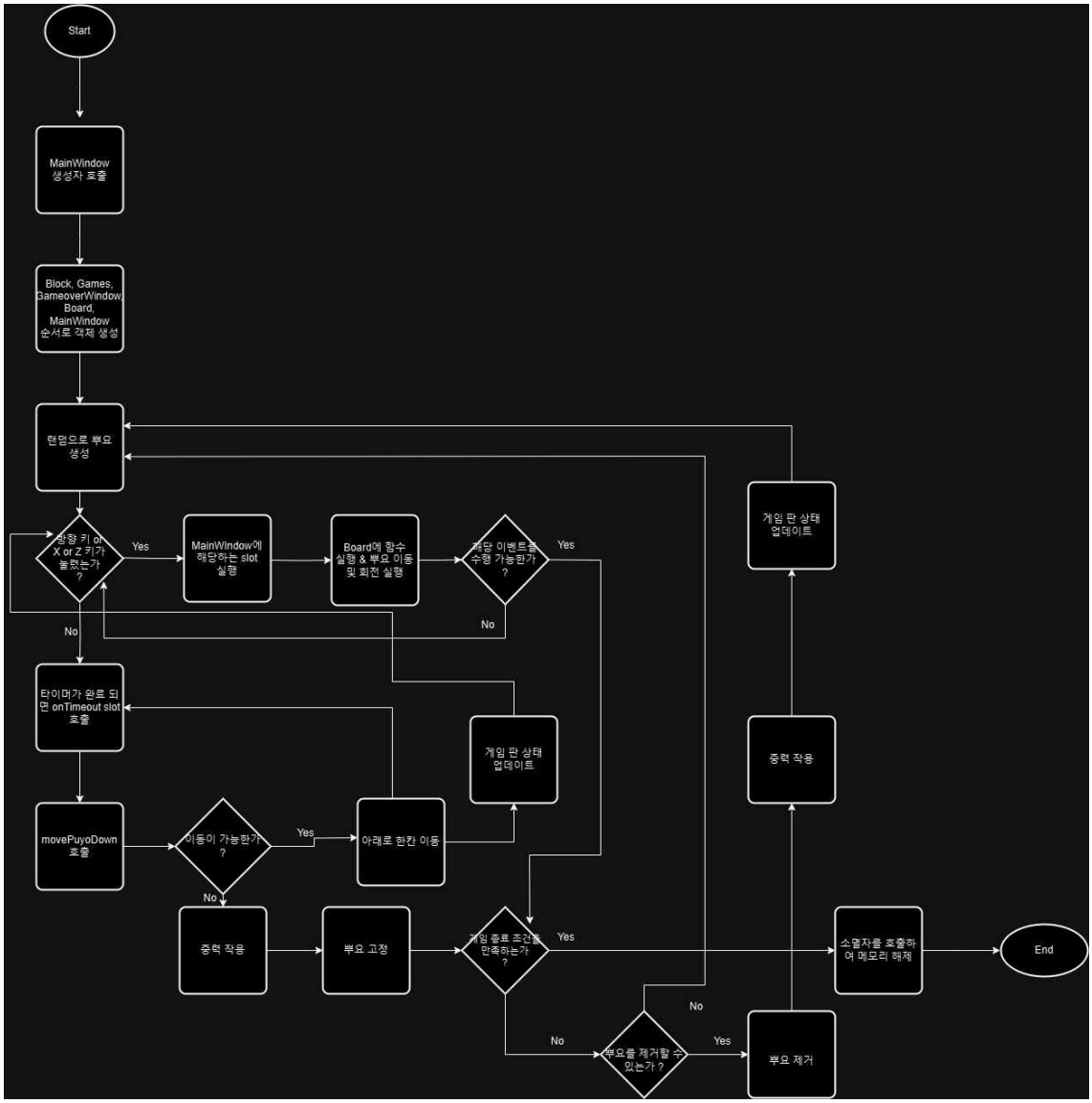
뿌요뿌요도 테트리스와 마찬가지로 크게 Puyo, Board 클래스로 구현했다. Puyo 클래스에는 테트리스의 Block 클래스처럼 뿌요를 초기화하는 함수, 회전 함수, 좌표값을 반환하는 함수가 존재하고 현재 뿌요를 의미하는 2차원 배열이 멤버변수로 존재한다. 뿌요뿌요는 [12][6]의 크기이므로 Board_col은 6, Board_row는 12이고 각 뿌요는 3x3의 칸에 담겨있다. 따라서 Puyo_wid=Puyo-hei는 3으로 초기화했다.
Board 클래스는 테트리스의 Board 클래스와 거의 동일 하나, 중력을 적용해야 하므로 중력을 적용하는 함수가 존재하고 뿌요를 제거하는 함수는 테트리스의 블록을 제거하는 함수와 알고리즘과 구현 방식이 다르다. 뿌요는 인접한 상하좌우 4칸을 탐색해야 하므로 탐색 알고리즘을 사용해야 하는데 그중에서 넓이 우선 탐색을 사용했다. 때문에, 게임판의 좌표를 설정해야 하므로 좌표를 의미하는 Coordinate라는 구조체를 정의하였다. 이 구조체의 멤버는 x좌표 y좌표가 있으며 인자로 전달 받은 x, y로 좌표를 저장하도록 했다. 넓이 우선 탐색으로 뿌요를 지우는 알고리즘은 pseudo 코드 부분에서 설명하겠다.
또한 뿌요뿌요도 마찬가지로 signal은 puyopuyo.h, slot은 mainwindow.h에 선언되어 있고 MainWindow 클래스에서 Board 클래스의 함수를 호출하여 동작하는 식으로 구현했다.

3. UML diagram



테트리스와 마찬가지로 각 클래스마다 멤버 변수, 멤버 함수가 존재하며 Board 클래스는 QWidegt 클래스는 상속하여 keyPressEvent 함수나 paintEvent함수를 오버라이딩하여 정의한다. 이 또한 Puyopuyo라는 이름공간 안에 정의하여 나중에 혼동되지 않도록 하였다. 테트리스와 달라진 점은 Board 클래스의 멤버로 뿌요의 색을 의미하는 PuyoColor 변수가 두 개 선언되었다는 점이다. 테트리스에서는 7개의 랜덤한 테트로미노가 생성되고 이를 1차원의 형태로 저장해 두었으므로 랜덤으로 int형 값을 발생시켜 해당 번호에 맞는 랜덤 테트로미노를 생성했다. 하지만 뿌요뿌요에서는 5개의 색상 중 하나를 골라서 2개를 붙여 뿌요를 만들어야 하므로 해당 색상을 의미하는 PuyoColor 변수를 두 개 선언했다. 따라서 Puyo 클래스에 initialize 함수는 인자로 색을 두 가지 전달받아 해당 값에 맞는 색으로 뿌요를 지정하도록 구현했다.

4. Flowchart



5. Pseudo code

테트리스와 같은 알고리즘을 공유하는 함수는 제외했다.

게임판에 그림을 그리는 함수:

```
//현재 게임판의 상태를 그림
for(1부터 게임판의 행 크기만큼)
        for(1부터 게임판의 열 크기만큼)
                switch(보드의 인덱스)
                case 0:
                하얀 사각형을 그림
                case 4:
                빨간 원을 그림
                case 5:
                노란 원을 그림
                case 6:
                초록 원을 그림
                case 7:
                파란 원을 그림
                case 8:
                보라색 원을 그림
                나머지:
                생략
        end for
end for
//
for(뿌요가 담긴 칸의 높이만큼)
        for(뿌요가 담긴 칸의 너비만큼)
                if(칸의 인덱스가 0이라면)
                        continue
                end if
                switch(칸의 인덱스)
                case 4:
                빨간 원을 그림
                case 5:
                노란 원을 그림
                case 6:
                초록 원을 그림
                case 7:
                파란 원을 그림
                case 8:
```

```
                보라 원을 그림
                나머지:
                생략
        end for
    end for
    //다음에 나올 5개의 뿌요 그리기
    for(0부터 다음 뿌요의 색이 저장된 벡터의 크기만큼 2씩 증가)
        벡터에서 값을 하나 꺼내어 저장
        다음 값을 꺼내어 저장
        해당 값을 임시 2차원 배열의 [0][1], [1][1] 에 저장
        for(뿌요가 저장된 칸의 높이만큼)
            for(뿌요가 저장된 칸의 너비만큼)
                if(임시 2차원 배열에 뿌요가 존재한다면(0이 아니라면))
                switch(해당 인덱스)
                case 4:
                빨간 원을 그림
                case 5:
                노란 원을 그림
                case 6:
                초록 원을 그림
                case 7:
                파란 원을 그림
                case 8:
                보라색 원을 그림
                나머지 :
                생략
            end for
        end for
    end for

중력을 적용하는 함수:
    for(j는 1부터 게임판의 열 크기만큼)
        for(i는 게임판의 행 크기-1부터 1까지)
            int underrow=i+1: 뿌요의 아래 행을 의미
            int emptycnt=0: 빈칸의 개수
            while(아래의 행이 비어있고 판의 경계를 넘지 않는다면)
                아래 행 증가
                빈칸 개수 증가
            end while
        I   if(빈칸이 존재하면)
                뿌요를 최대한 아래로 내림
```

```
                        원래의 자리를 공백으로 저장
                end if
            end for
        end for

뿌요를 지우는 함수:
        무한 반복:
        x, y 좌표에 상하좌우를 의미하는 1차원 배열 선언
        해당 좌표에 방문했는지 저장할 2차원 bool형 배열 선언
        뿌요가 터졌음을 판단하는 조건을 의미하는 bool형 변수 선언
        for(1부터 게임판 행의 크기만큼)
            for(1부터 게임판 열의 크기만큼)
                if(판이 비어있지 않고 방문하지 않은 좌표라면)
                    뿌요의 좌표가 저장될 Coordinate형 1차원 벡터 선언
                    뿌요를 탐색하면서 좌표가 저장될 Coordinate형 큐 선언
                    현재 뿌요의 좌표를 큐에 삽입
                    2차원 배열에 해당 좌표 위치 체크
                    현재 뿌요의 좌표를 벡터에 삽입
                    while(큐가 비어있지 않다면)
                        현재 큐에 담긴 x, y좌표를 저장하고 큐에서 제거
                        for(0부터 4까지 반복하며 상하좌우 탐색)
                            상하좌우의 새로운 x, y좌표 선언
                            if(새로운 x좌표가 0 이하거나 보드의
                            행보다 크거나 새로운 y좌표가 0
                            이하거나 보드의 열보다 크다면)
                                continue
                            end if
                            if(인접한 좌표의 뿌요 색이 같고
                            방문하지 않은 좌표라면)
                                해당 좌표를 큐, 벡터에 삽입
                                2차원 배열에 해당 위치 방문
                                체크
                            end if
                        end for
                    end while
                    if(4개 이상 인접한 뿌요가 있다면)
                        뿌요가 터졌음을 의미하는 조건을 true로 설정
                        뿌요의 좌표가 담긴 벡터에 존재하는 좌표를
                        빈칸으로 만듦
                    end if
                end if
```

```
            end for
        end for
        if(터질 뿌요가 없다면)
            반복문 탈출
        else
            점수 증가
            중력 작용 함수 호출
            게임판 상태 업데이트

게임 종료를 판단하는 함수:
        if(게임판 1행 3열에 0이 아닌 인덱스가 존재한다면)
            현재 상태를 게임 종료 상태로 설정
            게임 종료 화면 출력
        end if
```

6. 고찰

BFS 알고리즘을 이용하여 인접한 뿌요를 지우도록 했는데, BFS 알고리즘을 처음 접해봐서 잘 이해가 가지 않았다. 때문에 여러 블로그나 유튜브를 보고 큐의 특성인 FIFO를 이용하여 큐에 좌표를 저장하는 방식으로 구현했다. 또한 이 과정에서 x좌표와 y좌표를 이용하는데, 우리가 생각하는 x좌표는 가로축, y좌표를 세로축이므로 x좌표와 열을 매핑하고 y좌표와 행을 매핑하는 과정에서 굉장히 헷갈렸다. 이는 직접 펜으로 그려보면서 헷갈리지 않도록 조심하며 코드를 작성했다.

7. 동영상 링크 & 시나리오 검증

https://youtu.be/V9Bj3KZDp7E

테트리스와 마찬가지로 아래, 왼쪽, 오른쪽으로 이동, 시계·반시계 방향의 회전도 잘 작동되는 모습을 볼 수 있다. 또한 가만히 있어도 1초마다 뿌요가 하강하는 모습이 보인다.
테트리스와 다른 점인 뿌요 밑에 공간이 있다면 해당 뿌요가 밑으로 내려가고 4개나 4개 이상의 같은 색을 가진 뿌요가 인접하면 뿌요가 제거되고 중력이 작용하는 결과를 볼 수 있다. 마지막으로 종료 조건인 게임판의 0행 2열에 뿌요가 존재함을 만족해야 게임이 종료되는 모습을 볼 수 있다.

<뿌요뿌요테트리스>

1. Introduction

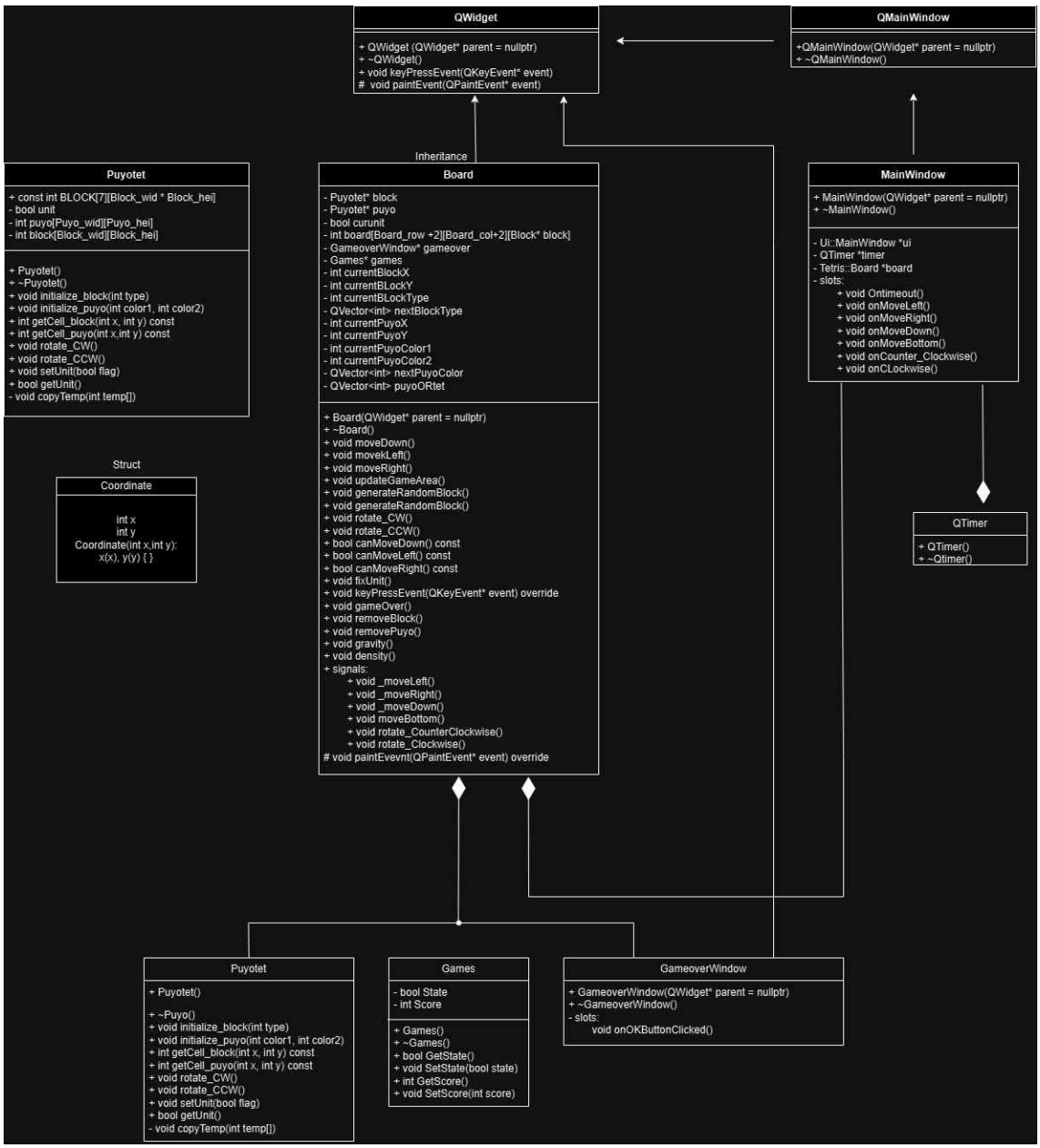
[16][8]크기의 게임판에 25개의 뿌요와 7개의 테트로미노가 랜덤으로 생성된다. 따라서 32개의 랜덤 유닛이 생성된다. 앞서 설명한 뿌요뿌요, 테트리스와 같이 아래, 왼쪽, 오른쪽으로 이동이 가능하고 시계·반시계 방향으로 회전 또한 모든 유닛이 가능하다.
뿌요뿌요테트리스에는 한가지 컨셉이 존재하는데, 테트로미노가 뿌요보다 밀도가 높아서 뿌요

위에 테트로미노가 올라가면 테트로미노가 밑으로 내려가고 아래에 있던 뿌요가 위로 올라온다.

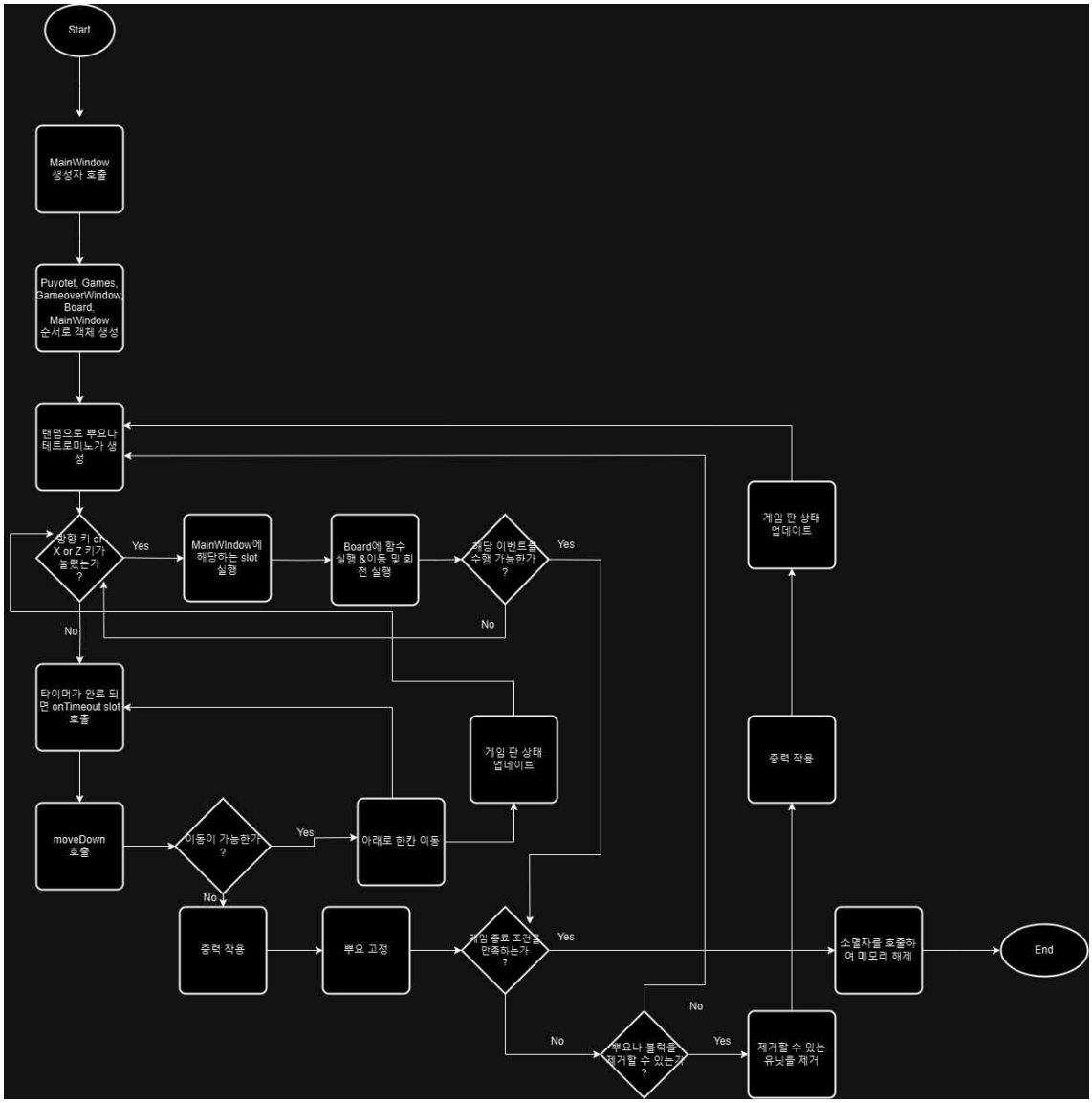
2. Algorithm

뿌요뿌요테트리스도 마찬가지로 유닛을 의미하는 Puyotet 클래스, 게임판을 의미하는 Board 클래스도 나뉘어 구성 되어있고 signal을 Puyopuyotetris파일에서 발생시켜 Mainwindow의 slot으로 받아 Mainwindow에서 board의 함수를 호출하는 방식을 똑같이 적용했다.
현재 유닛이 뿌요인지 테트로미노인지를 구분하는 변수를 멤버로 가져서 이 멤버 변수에 따라서 적용 시키는 함수가 다르다. 예를 들어서 하나의 회전 함수라도 if문을 사용하여 현재 유닛이 뿌요인가, 테트로미노인가에 따라서 다른 코드가 실행되는 방식으로 뿌요와 테트로미노를 구분하여 각 유닛에 맞는 동작을 시행하도록 구현했다.

3. UML diagram



4. Flowchart



5. pseudo Code

뿌요뿌요와 테트리스의 코드를 가져다 사용했으므로 뿌요뿌요와 테트리스에 없는 함수의 수도 코드를 작성하겠다.

뿌요와 테트로미노의 밀도 차이를 구현하는 코드:

```
int형 뿌요의 밑 행을 의미하는 변수(puyoline) 선언
for(1부터 게임판의 열 크기 만큼)
    for(게임판의 행 크기부터 1까지)
        if(블록이 존재하면)
            puyoline증가
        else
            반복 중단
end for
```

```
        end for
for(1부터 게임판의 열 크기만큼)
        for(puyoline부터 0까지)
                if(블록을 만나면)
                        뿌요위에 존재하는 블록의 개수 증가
                else if(빈칸을 만나면)
                        반복 중단
                else
                        블록 사이에 낀 뿌요의 정보를 저장
                        해당 뿌요를 빈칸으로 만듦
                end if
        end for
end for

for(1부터 게임판의 열 크기만큼)
        for(게임판의 행 크기부터 1까지)
                게임판 맨 아랫 줄부터 사이에 낀 뿌요의 정보로 저장
        end for
end for

        각 열에 블록 사이에 끼어있는 뿌요의 개수를 세어서 저장

for(1부터 게임판의 열 크기만큼)
        맨 밑줄+블록 사이에 끼어있는 뿌요의 개수의 행부터 블록으로 채움
end for
```

6. 고찰

수도코드에 작성한 코드는 뿌요와 테트로미노 사이에 밀도는 구현하는 함수를 작성한 코드이다. 이 코드는 블록 사이에 뿌요가 끼어있을 때, 블록위에 존재하는 뿌요의 정보, 개수를 저장하고 해당 뿌요를 빈칸으로 만든다. 맨 밑줄부터 해당 뿌요로 초기화 후 빈칸을 블록으로 채우는 방식으로 밀도 차이를 구현하려고 했다. 하지만 for문에서 구현이 잘못된 부분도 존재하고 블록 위에 블록이 올라가면 고정이 되는 상태인데 그 밑에 뿌요가 있다면 뿌요와 블록이 바뀌면 안되는데 이런 예외를 생각하지 않은 문제가 존재한다. 따라서 지금 작성한 코드를 추가하고 실행한 후, 뿌요가 바닥에 닿으면 게임이 종료되는 문제가 발생했고 이를 해결하지 못하고 제출하였다.
코드는 주석으로 처리하여 실행되지 않도록 했다.

7. 동영상 링크 & 시나리오 검증

https://youtu.be/Z-Wb1c0Aoe8

앞에 나올 랜덤 뿌요나 테트로미노 5개가 옆에 출력되고 뿌요나 테트로미노 모두 회전,
이동이 잘 되는 모습을 볼 수 있다. 또한 뿌요는 뿌요끼리, 블록은 블록끼리 지워지는 모습도
나타내었다. 고찰에서 서술한 것처럼 뿌요와 테트로미노의 밀도차이는 구현하지 못했기에
뿌요위에 테트로미노가 올라가도 뿌요가 테트로미노의 위로 올라오지는 않는다.