# QtProject

# 객체지향프로그래밍설계 보고서


과 목 객체지향프로그램밍설계

담당교수 심동규교수님

학 과 컴퓨터정보공학부

학 번 2022202104

이 름 김유찬

# 1. Introduction

Qt는 다양한 운영체제에서 application을 개발할 수 있게 해준다. 즉 동일한 코드이면 다양한 플랫폼에서 작동할 수 있게 해주는데 이를 크로스 플랫폼이라고 한다. 또한 이 Qt에서는 버튼, 리스트 등 다양한 사용자 인터페이스 구성 요소를 제공해주기 때문에 코드로만 구현하는 것뿐만 아니라 시각적인 요소까지 더해 좀 더 쉽게 빠르게 개발자가 프로그램을 만들 수 있게 도와준다. 또한 Qt는 다양한 커뮤니티와 문서, 튜토리얼, 예제가 있기 때문에 처음에 배울 때도 쉽게 할 수 있는 장점도 있다. 한국에서도 여러 기업에서 쓰고 있는 플랫폼이다.

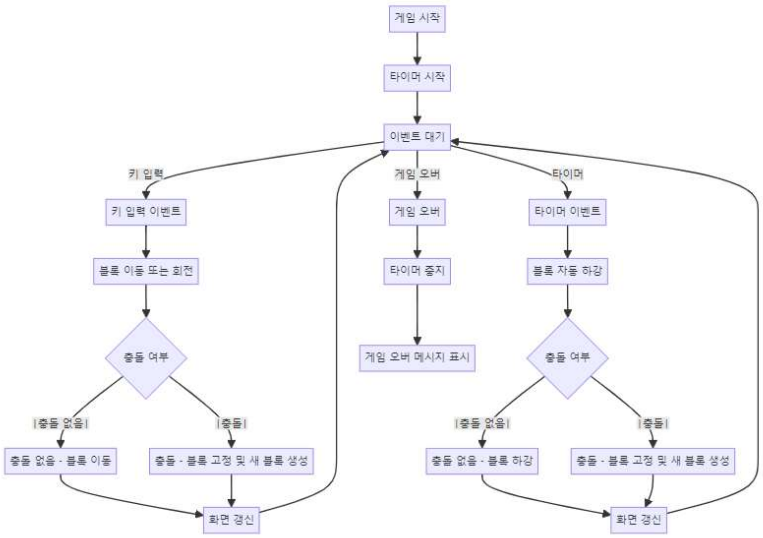
우리는 이 Qt를 사용하여 어릴 적에 많이 게임해봤던 테트리스, 뿌요뿌요, 뿌요테트리스를 만들어 볼 것이다.
1) 테트리스는 다들 알고 있겠지만 주어진 블록을 바닥에 떨어뜨리고 그 한줄에 모두 채워지면 점수를 얻고 한 줄이 사라지는 게임이다. 물론 시간이 지날수록 블록은 계속 아래로 떨어진다.

2) 뿌요뿌요는 테트리스처럼 시간이 지나면 블록이 떨어지고 바닥에 떨어지면 이웃한 색 같은 블록이 4개 이상이면 그 블록들이 사라지고 점수를 얻는 게임이다.

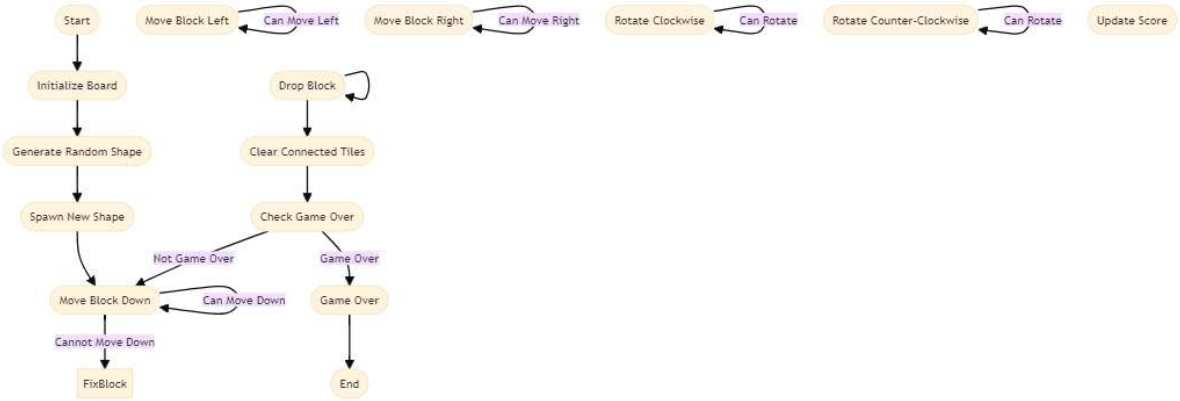
3) 뿌요 테트리스는 뿌요와 테트리스를 합쳐 놓은 게임이다. 사라지는 것은 뿌요 블록끼리 뿌요 규칙으로 사라지고 테트리스 블록끼리 테트리스 규칙으로 사라진다. 하지만 뿌요가 밑에 있고 테트리스가 그 위에 떨어지면 뿌요는 사라지고 사라진만큼 위에 같은 뿌요가 그 위치로 떨어진다.

# 2. Flowchart

1)Tetris




2) PuyoPuyo




# 2. Pseudocode

## 1) TerisGame pseudocode

```
class TetrisGame:
    function TetrisGame() {
        ROW = 22
        COL = 10
        axis_row = 0
        axis_col = COL / 2
        score = 0
        gameover = false
        초기화 board (크기: ROW x COL, 초기값: "Empty")
        초기화 boardColors (크기: ROW x COL, 초기값: QColor(255, 255, 255))
        초기화 랜덤 시드

        shapes = [
            I 모양,
            O 모양,
            T 모양,
            S 모양,
            Z 모양,
            J 모양,
            L 모양
        ]

        shapeColors = [
            I 모양 - Cyan,
            O 모양 - Yellow,
            T 모양 - Purple,
            S 모양 - Green,
            Z 모양 - Red,
            J 모양 - Blue,
            L 모양 - Orange
        ]

    }

    ~TetrisGame()

    function initializeBoard() {
        모든 셀을 "Empty"로 초기화
        첫 두 줄을 회색(QColor(192, 192, 192))으로 설정
        나머지 줄은 흰색(QColor(255, 255, 255))으로 설정
    }

    function getRandomShape()  {
        return 랜덤 모양 인덱스
    }
```

```
function getRandomShape()  {
    return 랜덤 모양 인덱스
}

function fixBlock() {
    현재 블록의 색상을 어둡게 변환
    현재 블록을 보드에 고정
    게임 종료 여부 검사:
        만약 종료 조건 만족 시:
            gameover = true
            이벤트 gameOver() 발생
}

function darkenColor(color)  {
    return 어두운 색상
}

function brightenColor(color)  {
    return 밝은 색상
}

function clearFullLines() {
    linesToClear 초기화
    보드의 각 행을 순회하면서 가득 찬 행을 찾음
    만약 삭제할 행이 없으면 종료
    삭제할 행을 밝게 표시
    이벤트 linesCleared 발생
    지연 후 삭제:
        삭제할 행들을 위의 행으로 채움
        최상단 행은 비움
    이벤트 linesCleared 다시 발생
    score 증가
}

function checkGameOver()  {
    첫 두 줄 검사:
        만약 "Empty"가 아닌 셀이 있으면 return true
    return false
}

function getCurrentShape()  {
    return 현재 블록 모양
}
```

```
function brightenColor(color)  {
    return 밝은 색상
}

function clearFullLines() {
    linesToClear 초기화
    보드의 각 행을 순회하면서 가득 찬 행을 찾음
    만약 삭제할 행이 없으면 종료
    삭제할 행을 밝게 표시
    이벤트 linesCleared 발생
    지연 후 삭제:
        삭제할 행들을 위의 행으로 채움
        최상단 행은 비움
    이벤트 linesCleared 다시 발생
    score 증가
}

function checkGameOver()  {
    첫 두 줄 검사:
        만약 "Empty"가 아닌 셀이 있으면 return true
    return false
}

function getCurrentShape()  {
    return 현재 블록 모양
}

function rotateCurrentShapeClockwise() {
    rotated = 회전된 블록 (시계방향)
    만약 회전 가능하면:
        현재 블록을 회전된 블록으로 설정
}

function rotateCurrentShapeCounterClockwise() {
    rotated = 회전된 블록 (반시계방향)
    만약 회전 가능하면:
        현재 블록을 회전된 블록으로 설정
}

function rotateShape(shape, clockwise)  {
    n = shape 크기
    rotated 초기화 (크기: n x n)
    블록 회전 로직
    return 회전된 블록
}
```

```
function canRotate(rotatedShape)  {
    n = rotatedShape 크기
    회전 후의 위치 계산:
        만약 새 위치가 경계를 넘거나 충돌하면 return false
    return true
}

function canMoveDown()  {
    next_row = 현재 블록의 다음 행 위치
    현재 블록 모양 순회:
        만약 다음 행 위치가 경계를 넘거나 충돌하면 return false
    return true
}

function dropBlock() {
    while (canMoveDown()) {
        현재 블록을 한 칸 아래로 이동
    }
    function fixBlock() 호출
    function clearFullLines() 호출
    function spawnNewShape() 호출
}

function generateNextShapes() {
    while (nextQueue 크기 < 5) {
        nextQueue에 랜덤 블록 추가
    }
}

function spawnNewShape() {
    axis_row = 0
    axis_col = COL / 2
    현재 블록 인덱스 = nextQueue의 첫 번째 블록
    nextQueue에서 블록 하나 제거
    현재 블록 모양 = shapes[currentShapeIndex]
    function generateNextShapes() 호출
}
```

## 2) TetrisWindow

```
TetrisWindow class

function TetrisWindow(parent) {
    p_game를 새로운 TetrisGame 객체로 초기화
    p_game의 linesCleared 신호와 updateBoard 슬롯을 연결
    p_game의 gameOver 신호와 handleGameOver 슬롯을 연결
    타이머 시작(500ms 간격)
    윈도우 크기 조정(700x1100)
    윈도우 제목 설정("Tetris")
}

function ~TetrisWindow() {
    p_game 객체 삭제
}

function paintEvent(event) {
    ROW, COL, board, boardColors, axis_row, axis_col, score를 p_game에서 가져옴
    QPainter 객체 생성
    하얀색 배경의 게임 보드 그리기
    "NEXT" 텍스트 그리기
    보드의 각 셀을 순회하며 블록 색상으로 그리기
    현재 블록 그리기
    다음 블록 그리기
    현재 점수 텍스트 그리기
}

function keyPressEvent(event) {
    if(p_game이 게임오버 상태라면)
        아무것도 하지 않음
    키 이벤트 처리{
        아래 키: moveBlockD()
        왼쪽 키: moveBlockL()
        오른쪽 키: moveBlockR()
        Z 키: p_game의 블록 반시계 방향 회전, drawNext()
        X 키: p_game의 블록 시계 방향 회전, drawNext()
        스페이스 키: p_game의 블록 떨어뜨리기, drawNext()
    }
}

function timerEvent(event) {
    if(p_game이 게임오버 상태라면)
        아무것도 하지 않음
    타이머 이벤트 처리:
        만약 event의 타이머 ID가 timer와 일치하면 moveBlockD()
        drawNext()
}
```

```

function moveBlockD() {
    if(canMoveD()){
        p_game의 axis_row 증가
    }
    else{
        p_game의 블록 고정, 라인 제거, 새로운 블록 생성
        drawNext()
    }
}

function moveBlockL() {
    if(canMoveL()){
        p_game의 axis_col 감소
        drawNext()
    }
}

function moveBlockR() {
   if(canMoveR()){
        p_game의 axis_col 증가
        drawNext()
    }
}

function canMoveD() {
    int next_row = p_game의 axis_row + 1
    현재 블록 모양 가져옴
    블록의 각 셀을 순회하며 이동 가능 여부 검사
    if(다음 행 위치가 경계를 넘거나 충돌하면)
        return false
    else
        return true
}

function canMoveL() {
    int next_col = p_game의 axis_col - 1
    현재 블록 모양 가져옴
    블록의 각 셀을 순회하며 이동 가능 여부 검사
    if(다음 열 위치가 경계를 넘거나 충돌하면)
       return false
    else
        return true
    }

function canMoveR() {
    int next_col = p_game의 axis_col + 1
    현재 블록 모양 가져옴
```

```
    블록의 각 셀을 순회하며 이동 가능 여부 검사
    if(다음 열 위치가 경계를 넘거나 충돌하면)
        return false
    else
        return true
    }

function rotateBlockClockwise() {
    p_game의 블록 시계 방향 회전
    drawNext()
}

function rotateBlockCounterClockwise() {
    p_game의 블록 반시계 방향 회전
    drawNext()
}

function fixBlock() {
    p_game의 블록 고정
}

function createNewBlock() {
    p_game의 새로운 블록 생성
}

function drawNext() {
    if(p_game이 게임오버 상태){
        모든 빈 셀을 고정 블록으로 설정
        return
    }
    update()
}

function updateBoard() {
    update()
}

function handleGameOver() {
    타이머 종료
    게임 오버 메시지 박스 표시
}
```

3) PuyoPuyoWindow

```
BEGIN PuyoPuyoWindow CLASS

    // Constructor
    FUNCTION PuyoPuyoWindow(parent: QWidget)
    {
        INHERIT QWidget(parent)
        p_game = NEW PuyoPuyoGame()

        CONNECT p_game.tilesCleared TO updateBoard
        CONNECT p_game.gameOver TO handleGameOver

        timer = START_TIMER(500) // Start timer with 500ms interval
        RESIZE_WINDOW(700, 1100)
        SET_WINDOW_TITLE("Puyo Puyo")
    }

    // Destructor
    FUNCTION ~PuyoPuyoWindow()
    {
        DELETE p_game
    }

    FUNCTION paintEvent(event: QPaintEvent)
    {
        DECLARE ROW, COL, board, boardColors, axis_row, axis_col, score FROM p_game
begin PuyoPuyoWindow class

    // Constructor
    function PuyoPuyoWindow(parent: QWidget)
    {
        inherit QWidget(parent)
        p_game = new PuyoPuyoGame()

        connect p_game.tilesCleared to updateBoard
        connect p_game.gameOver to handleGameOver

        timer = start_timer(500) // Start timer with 500ms interval
        resize_window(700, 1100)
```

```
        set_window_title("Puyo Puyo")
    }

    // Destructor
    function ~PuyoPuyoWindow()
    {
        delete p_game
    }

    function paintEvent(event: QPaintEvent)
    {
        declare ROW, COL, board, boardColors, axis_row, axis_col, score from p_game

        create QPainter painter(this)
        painter.setBrush(WHITE_BRUSH)
        painter.drawRect(50, 50, COL * BlockSize, ROW * BlockSize)

        painter.setPen(BLACK_PEN)
        painter.setFont("Arial", 14)
        painter.drawText(QRect(50 + COL * BlockSize + 20, 50, 100, 20), "NEXT")

        for(i in 0 to ROW)
        {
            for(j in 0 to COL)
            {
                if (i < 2)
                {
                    painter.setBrush(GRAY_BRUSH)
                }
                else
                {
                    painter.setBrush(WHITE_BRUSH)
                }
                painter.drawRect(j * BlockSize + 50, i * BlockSize + 50, BlockSize, BlockSize)
                if (board[i][j] is "FixedBlock")
                {
                    painter.setBrush(boardColors[i][j])
                    painter.drawEllipse(j * BlockSize + 50, i * BlockSize + 50, BlockSize, BlockSize)
                }
            }
        }

        declare shape, shapeColors from p_game.getCurrentShape()

        for(i in 0 to shape.size)
        {
            for(j in 0 to shape[i].size)
            {
                if (shape[i][j] is 1)
                {
                    declare row = axis_row + i
                    declare col = axis_col + j
                    painter.setBrush(shapeColors[i][j])
                    painter.drawEllipse(col * BlockSize + 50, row * BlockSize + 50, BlockSize, BlockSize)
                }
            }
        }

        declare next_shape_y = 50 + 100
        declare tempQueue = p_game.nextQueue
        declare count = 0

        while (not tempQueue.empty and count < 2)
        {
            declare nextShape = tempQueue.front()
            tempQueue.pop()
            declare next_shape, nextColors from nextShape

            for(i in 0 to next_shape.size)
            {
                for(j in 0 to next_shape[i].size)
                {
                    if (next_shape[i][j] is 1)
                    {
                        painter.setBrush(nextColors[i][j])
                        painter.drawEllipse(COL * BlockSize + 70 + j * BlockSize, next_shape_y + i * BlockSize, BlockSize, BlockSize)
                    }
                }
            }

            next_shape_y += (next_shape.size() + 1) * BlockSize
            count++
        }

        painter.setPen(BLACK_PEN)
        painter.setFont("Arial", 14)
        painter.drawText(QRect(50 + COL * BlockSize + 20, ROW * BlockSize + 80, 150, 20), "current score : " + score)
    }

    function keyPressEvent(event: QKeyEvent)
    {
        if (p_game.gameover)
        {
            return
        }

        switch (event.key)
        {
            case Qt::Key_Down:
                moveBlockD()
            case Qt::Key_Left:
                moveBlockL()
            case Qt::Key_Right:
```

```
                moveBlockR()
            case Qt::Key_Z:
                p_game.rotateCurrentShapeCounterClockwise()
                drawNext()
            case Qt::Key_X:
                p_game.rotateCurrentShapeClockwise()
                drawNext()
            case Qt::Key_Space:
                p_game.dropBlock()
                drawNext()
            default:
                // No action
        }
    }

    function timerEvent(event: QTimerEvent)
    {
        if (p_game.gameover)
        {
            return
        }

        if (event.timerId is timer)
        {
            moveBlockD()
            drawNext()
        }
    }

    function moveBlockD()
    {
        if (canMoveD())
        {
            p_game.axis_row++
```

```
        }
        else
        {
            p_game.fixBlock()
            p_game.clearConnectedTiles()
            createNewBlock()
        }
        drawNext()
    }

    function moveBlockL()
    {
        if (canMoveL())
        {
            p_game.axis_col--
            drawNext()
        }
    }

    function moveBlockR()
    {
        if (canMoveR())
        {
            p_game.axis_col++
            drawNext()
        }
    }

```

```
    FUNCTION drawNext()
    {
        IF (p_game.gameover)
        {
            FOR EACH i IN 0 TO p_game.ROW
            {
                FOR EACH j IN 0 TO p_game.COL
                {
                    IF (p_game.board[i][j] IS "Empty")
                    {
                        p_game.board[i][j] = "FixedBlock"
                    }
                }
            }
            RETURN
        }
        update()
    }

    FUNCTION updateBoard()
    {
        update()
    }

    FUNCTION handleGameOver()
    {
        KILL_TIMER(timer)
        DISPLAY_MESSAGE("Game Over", "Game Over! Your score: " + p_game.score)
    }

END CLASS
```

4) PuyoPuyoGame

```
begin PuyoPuyoGame class

    // Constructor
    function PuyoPuyoGame()
    {
        ROW = 14
        COL = 6
        axis_row = 0
        axis_col = COL / 2
        score = 0
        gameover = false
        board.resize(ROW, vector<string>(COL, "Empty"))
        boardColors.resize(ROW, vector<QColor>(COL, QColor(255, 255, 255)))
        srand(time(0))

        shapes = {
            // Puyo shape
            {{0, 0, 0},
             {0, 1, 0},
             {0, 1, 0}}
        }

        shapeColors = {
            QColor(255, 0, 0),   // Red
            QColor(255, 255, 0), // Yellow
            QColor(0, 255, 0),   // Green
            QColor(0, 0, 255),   // Blue
            QColor(128, 0, 128)  // Purple
        }

        initializeBoard()
        generateNextShapes()
        spawnNewShape()
    }
```

```
    // Destructor
    function ~PuyoPuyoGame()
    {
    }

    function initializeBoard()
    {
        for(i in 0 to ROW)
        {
            for(j in 0 to COL)
            {
                board[i][j] = "Empty"
                if (i < 2)
                {
                    boardColors[i][j] = QColor(192, 192, 192) // First two rows gray
                }
                else
                {
                    boardColors[i][j] = QColor(255, 255, 255) // Others white
                }
            }
        }
    }

    function getRandomShape(): vector<vector<int>>
    {
        return shapes[0]
    }

    function getRandomColors(): vector<vector<QColor>>
    {
        colors.resize(3, vector<QColor>(3, QColor(255, 255, 255)))
        colors[1][1] = shapeColors[rand() % shapeColors.size()]
        colors[2][1] = shapeColors[rand() % shapeColors.size()]
        return colors
    }
```

```
function fixBlock()
{
    for(i in 0 to currentShape.size())
    {
        for(j in 0 to currentShape[i].size())
        {
            if (currentShape[i][j] == 1)
            {
                row = axis_row + i
                col = axis_col + j
                board[row][col] = "FixedBlock"
                boardColors[row][col] = darkenColor(currentShapeColors[i][j])
            }
        }
    }

    if (checkGameOver())
    {
        gameover = true
        emit gameOver()
    }
}

function darkenColor(color: QColor): QColor
{
    factor = 50
    return QColor(max(color.red() - factor, 0),
                  max(color.green() - factor, 0),
                  max(color.blue() - factor, 0))
}
```

```
function clearConnectedTiles()
{
    tilesCleared = false
    visited.resize(ROW, vector<bool>(COL, false))
    directions = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}}
    connected.clear()

    for(i in 0 to ROW)
    {
        for(j in 0 to COL)
        {
            if (board[i][j] == "FixedBlock" and not visited[i][j])
            {
                connected.clear()
                stack.push({i, j})
                color = boardColors[i][j]

                while (not stack.empty())
                {
                    x, y = stack.top()
                    stack.pop()
                    if (x < 0 or x >= ROW or y < 0 or y >= COL or visited[x][y] or board[x][y] != "FixedBlock" or boardColors[x][y] != color)
                    {
                        continue
                    }

                    visited[x][y] = true
                    connected.push_back({x, y})

                    for(dir in directions)
                    {
                        stack.push({x + dir[0], y + dir[1]})
                    }
                }
```

```
                if (connected.size() >= 4)
                {
                    for(pos in connected)
                    {
                        boardColors[pos.first][pos.second] = brightenColor(boardColors[pos.first][pos.second])
                    }

                    QTimer.singleShot(200, [this, connected, color]()
                    {
                        for(pos in connected)
                        {
                            board[pos.first][pos.second] = "Empty"
                            boardColors[pos.first][pos.second] = QColor(255, 255, 255)
                        }

                        for(col in 0 to COL)
                        {
                            empty_row = ROW - 1
                            for(row in ROW - 1 down to 0)
                            {
                                if (board[row][col] != "Empty")
                                {
                                    swap(board[row][col], board[empty_row][col])
                                    swap(boardColors[row][col], boardColors[empty_row][col])
                                    empty_row--
                                }
                            }
                        }
```

```
                        }
                    }

                    score += connected.size()
                })
            }
        }
    }
}

function checkGameOver(): bool
{
    for(j in 0 to COL)
    {
        if (board[1][j] != "Empty")
        {
            return true
        }
    }
    return false
}
```

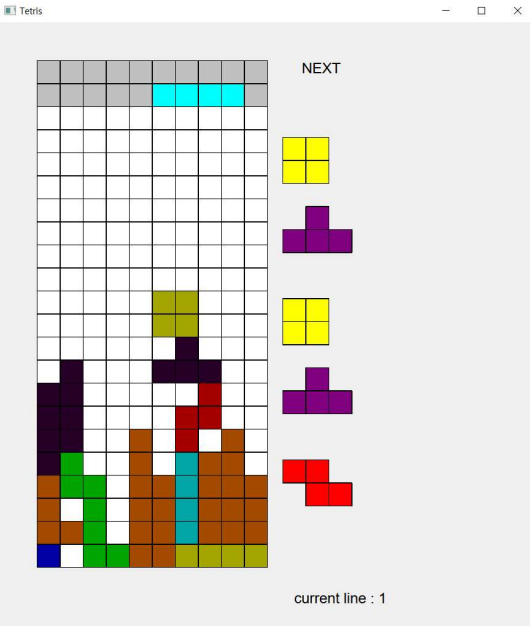
```
function generateNextShapes()
{
    while (nextQueue.size() < 2)
    {
        nextQueue.push({getRandomShape(), getRandomColors()})
    }
}

function spawnNewShape()
{
    axis_row = 0
    axis_col = COL / 2
    nextShape = nextQueue.front()
    currentShape = nextShape.first
    currentShapeColors = nextShape.second
    nextQueue.pop()
    generateNextShapes()
}

end class
```

# 3. 결과화면

1) Tetris




- 블록에 중력이 작용하여 시간에 따라 아래로 떨어진다. 그리고 스페이스를 누르면 바로 밑으로 떨어진다. 또 방향키를 왼쪽, 오른쪽 키를 누르면 블록이 왼쪽, 오른쪽으로 움직이고 x,z 키를 누르면 시계방향, 반시계방향으로 움직인다. 또한 밑으로 떨어져서 블록이 더 이상 움직이지 않는다면 원래 블록보다 살짝 어둡게 만들어준다. 그리고 한 줄이 완성되었을 때 사라지기 전에 원래 블록보다 살짝 밝게 만들어주고 사라지게 만든다.

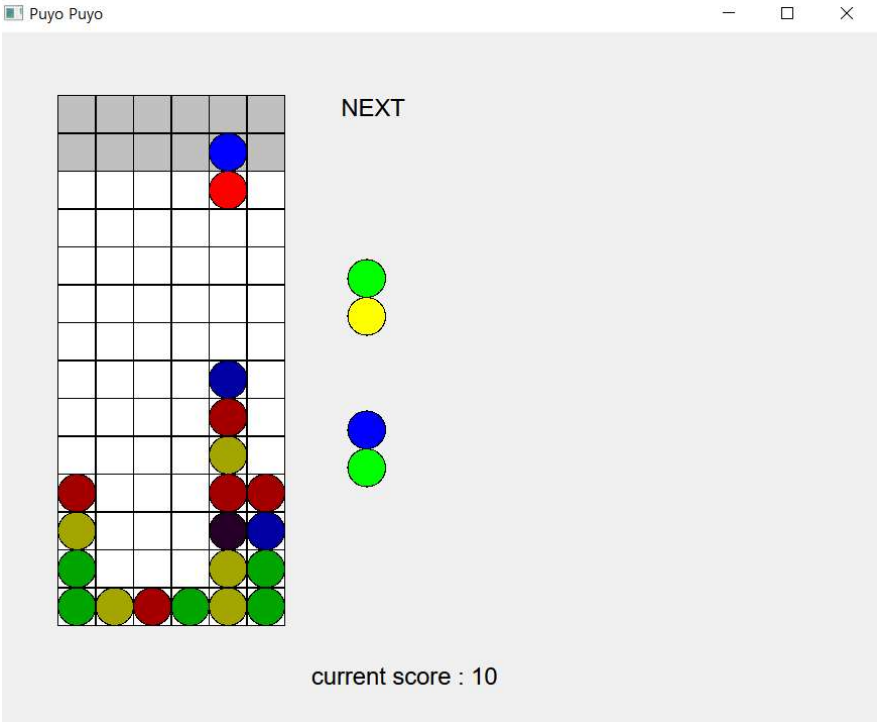
- 보드 오른쪽에는 다음 블록 5개가 이미 있고 밑에는 점수화면이 있다. 보드 위에 2줄에는 회색인데 여기서 블록이 나오는 것이다. 만약 회색 칸에 블록이 고정이 되면 Gameover 화면이 나오며 점수도 동시에 같이 나온다.

- 블록이 한 줄 사라질 때 알고리즘 구현은 이중 for문으로 한 줄씩 한 줄씩 순회해서 한줄이 모두 블록이 차 있는지 확인하고 만약 다 차 있으면 없애고 위에 있는 블록을 아래로 내려오게 했다.

- 회전할 때는 int n = shape.size(); 로 하고 시계 방향일 때는 rotated[j][n - i - 1] = shape[i][j]; 반시계 방향일 때는 rotated[n - j - 1][i] = shape[i][j];로 해서 구현했다.

링크 : https://youtu.be/4gJAl097ToI

2) PuyoPuyo




- 테트리스 게임처럼 블록을 움직일 수 있고 회전할 수 있다. 단지 테트리스랑 다른 것은 타일이 사라질 때 조건이 한줄이 다 채워질 때가 아니라 뿌요뿌요 타일 4개 이상이 인접했을 때 타일이 사라지고 점수가 4점이 올라가는 구조이다.

- 타일 4개 이상이 인접했을 때 사라지는 알고리즘은 DFS를 써서 문제를 풀었다. 해당 타일 기준으로 상하좌우 움직여서 같은 색 타일인지 확인한다. 그리고 참고로 이미 방문한 타일은 확인 안 한다. 만약 해당 타일이 4개 이상이면 타일은 사라지고 중력이 작용하여 밑으로 떨어지게 된다.

- 테트리스와 같이 회전할 때는 int n = shape.size(); 로 하고 시계 방향일 때는 rotated[j][n - i - 1] = shape[i][j]; 반시계 방향일 때는 rotated[n - j - 1][i] = shape[i][j];로 해서 구현했다.

링크 : https://youtu.be/4gJAl097ToI

## 4. 고찰

Qt 는 프로젝트를 사용해서 테트리스, 뿌요뿌요, 테트리스 뿌요를 만들라고 했을 때는 많이 막막했지만 signal, slot 개념을 이해하고 UI 부분들을 공부해보니 블록을 사라지게 할 때 알고리즘을 어떻게 써야하는지 또 회전할 때 어떤 알고리즘으로 구현해야 할지가 어려웠지 나머진 그렇게 많이 어렵진 않았다. 이전에도 Python의 Tkinter 모듈을 이용해서 GUI 프로그램을 만들어봤는데 이 때도 처음에 어떻게 UI를 코드로 구현해야하는지 사용법에 처음에만 애 먹었지 구현할 때는 핵심 알고리즘 구현하는 부분이 마찬가지로 가장 어려웠다. BFS, DFS 구현은 객체지향프로그래밍 실습 때 문제를 풀어봐서 뿌요뿌요를 구현할 때 어렵지 않았다.

하지만 두 게임을 합친 뿌요 테트리스는 구현하지 못했다. 앞에서 만든 두 게임에 필요한 부분만 코드를 넣고 수정하면 쉽게 만들 수 있겠다고 생각하기 때문에 시간만 충분히 더 있었으면 구현을 할 수 있을 것 같다.