객체지향 프로그래밍(2) 최종 기말 과제 구현 보고서

작성자: 김홍준

날짜: 2019.12.11

구현한 코드를 설명하기 앞서 공지사항의 [객체지향프로그래밍(2)] 컴포넌트 기반 설계를 활용한 MonsterWorld 제작본의 첨부파일을 기반으로 작성했으며 테트리스 코드는 서범주 교수님의 상속 모델로 구현한 테트리스 코드(이하 테트리스 코드)를 참고했음을 밝힙니다.

1. 수행해야할 과제
   - 컴포넌트 기반 설계를 이용해 테트리스 만들기
2. 기능 세분화

   ①. 테트리스 블록들과 맵을 만들고 출력하기

   ②. 테트리스 블록의 이동 및 회전

   ③. 테트리스 블록 판정, 다음 블록, 다음 패널 추가

   ④. 테트리스 블록 줄 지우기, 점수, 최고 점수, 게임 오버 추가

2 - ① 테트리스 블록들과 맵을 만들고 출력하기 구현

먼저 테트리스 블록과 맵의 width, height, position 을 컨트롤 하기 위해 Transform.h 파일에 여러 함수를 추가했다. 여기서 width 와 height 는 scale 로 관리하기 위해 setScale 함수를 만들었다.

```
// 블럭의 위치를 옮기기 위해 만든 함수
// 테트리스의 하드 드롭을 구현하기 위함
void setPosition(float x, float y)
{ position = Vector2(x, y); }

// 블럭의 width와 height를 조정하기 위해 만든 함수
void setScale(float x, float y)
{ scale = Vector2(x, y); }

// 블럭의 이동을 위해 만든 함수
void moveUp()
{ position = position + Vector2::down; }

void moveDown()
{ position = position + Vector2::up; }

void moveLeft()
{ position = position + Vector2::left; }

void moveRight()
{ position = position + Vector2::right; }
```

그리고 테트리스 코드에서는 테트리스 블록을 회전하는 함수가 있어서 rotation 도 컨트롤 하기 위해 함수를 추가했다.

```
// 블럭의 회전을 위해 만든 함수
void rotate() // x로 90도 회전
{ rotation.x += 90; }
```

위 기능을 추가시킨 후 일단 GameEngine.cpp 에서 맵을 만들고 블록을 출력하는 창을 만들기 위해 GameObject map 을 만들었다. 여기서 position 은 맵의 출력 위치를 정하고 scale 은 width 와 height 를 담당하므로 맵의 크기를 정한다.

```
GameObject* map = new GameObject(
    "map", nullptr, "map", "",
    // 왼쪽부터 position, scale(width, height)를 위해 추가
    Vector2{ 1, 1 }, Vector2{ 30, 40 }
);
objs.push_back(map);
```

위 코드가 잘 수행하게 만들기 위해 GameObject 생성자에서 Scale 을 추가로 받게 했고 parent 가 있을 경우 자기 객체를 부모의 children 에 들어가도록 만들었다.

```
// width와 height를 쓰기 위해 scale을 추가
GameObject(const string& name, GameObject* parent = nullptr,
    const string& tag = "", const string& shape = "",
    const Vector2& pos = Vector2::zero,
    const Vector2& scale = Vector2::ones
);
```

```
// width, height를 위한 scale 추가
GameObject::GameObject(const string& name,
    GameObject* parent,
    const string& tag, const string& shape,
    const Vector2& pos, const Vector2& scale
    )
    : name(name), tag(tag), enabled(true), parent(parent),
    transform(new Transform(this, shape, pos, scale) ) {
    components.clear();
    components.push_back(transform);
    // 부모 GameObject가 있으면 그 부모의 children에 추가
    if (parent != nullptr) {
        parent->children.push_back(this);
    }
}
```

맵을 만든 후 테트리스의 블록과 모양을 구현하기 위해 상속 모델로 구현한 테트리스 코드 내에서 BlockShape 구조체를 꺼내왔다. 이 코드는 Utils.h 파일에 저장하였다.

```
// 서범주 교수님의 상속 모델로 구현한 테트리스 코드 중
struct BlockShape {
    string shape;
    int width;
    int height;

    static vector<BlockShape> candidates;
};
```

Candidates 내용은 테트리스 코드 내 candidates 를 Utils.cpp 파일에 정의했다.

```
// 서범주 교수님의 상속 모델로 구현한 테트리스 코드 중
vector<BlockShape> BlockShape::candidates = {
    { "\xB2\xB2 \xB2 \xB2", 2, 3},
    { "\xB2\xB2\xB2\xB2",  2, 2 },
    { "\xB2\xB2\xB2\xB2",  4, 1 },
    { "\xB2\xB2\xB2 \xB2 ", 2, 3 },
    { " \xB2\xB2\xB2\xB2 ", 2, 3 },
    { "  \xB2\xB2\xB2 \xB2", 2, 3 },
    { "\xB2 \xB2\xB2 \xB2", 2, 3 }
};
```

이어서 GameEngine.cpp 에 블록들을 만들고 관리하기 위해 BlockSet 을 만들었다. 그 이유는 BlockSet 을 Block 들의 부모로 지정하면 관리하기가 편하기 때문이며 그 이외의 기능은 없기 때문에 Active 는 false 로 두었다.

```
// 테트리스 블럭들을 관리하기 위한 임시 블럭들의 부모
GameObject* blockSet = new GameObject(
    "blockSet", map, "blockSet", "",
    Vector2{ -1, -1 }, Vector2{ 0, 0 }
);
objs.push_back(blockSet);
blockSet->setActive(false);
```

테트리스 블록들은 테트리스 코드와 scale 을 이용해 BlockSet 의 자식으로 들어가며 테트리스 게임이 돌아갈 때는 새로운 GameObject 를 만들어 아래 블록들 중 한 개를 복사해 랜덤으로 나오게끔 만들 예정이다. 따라서 Active 는 false 로 두었다.

```
// 블럭 정보들을 불러옴
vector<BlockShape>& blockInfo = BlockShape::candidates;
```

```
// 테트리스 블럭 중 L미노
GameObject* lmino = new GameObject(
    "lmino", blockSet, "prepareBlock", blockInfo.at(0).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(0).width, blockInfo.at(0).height }
);
objs.push_back(lmino);
lmino->setActive(false);
```

```
// 테트리스 블럭 중 O미노
GameObject* omino = new GameObject(
    "omino", blockSet, "prepareBlock", blockInfo.at(1).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(1).width, blockInfo.at(1).height }
);
objs.push_back(omino);
omino->setActive(false);
```

```
// 테트리스 블럭 중 I미노
GameObject* imino = new GameObject(
    "imino", blockSet, "prepareBlock", blockInfo.at(2).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(2).width, blockInfo.at(2).height }
);
objs.push_back(imino);
imino->setActive(false);
```

```
// 테트리스 블럭 중 J미노
GameObject* jmino = new GameObject(
    "jmino", blockSet, "prepareBlock", blockInfo.at(3).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(3).width, blockInfo.at(3).height }
);
objs.push_back(jmino);
jmino->setActive(false);
```

```
// 테트리스 블럭 중 z미노
GameObject* zmino = new GameObject(
    "zmino", blockSet, "prepareBlock", blockInfo.at(4).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(4).width, blockInfo.at(4).height }
);
objs.push_back(zmino);
zmino->setActive(false);
```

```
// 테트리스 블럭 중 t미노
GameObject* tmino = new GameObject(
    "tmino", blockSet, "prepareBlock", blockInfo.at(5).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(5).width, blockInfo.at(5).height }
);
objs.push_back(tmino);
tmino->setActive(false);
```

```
// 테트리스 블럭 중 s미노
GameObject* smino = new GameObject(
    "smino", blockSet, "prepareBlock", blockInfo.at(6).shape, Vector2{ 1,1 },
    Vector2{ blockInfo.at(6).width, blockInfo.at(6).height }
);
objs.push_back(smino);
smino->setActive(false);
```

위 블록들 중 한 개의 블록만 복사하여 저장하도록 하기 위해 GameObject 객체를 만들었으며 맵에 띄울 것이기 때문에 부모를 map 으로 설정했다.

```
GameObject* movingBlock = new GameObject(
    blockSet->children.at(rand() % 7), "mino", "movingBlock"
);
movingBlock->setParent(map);
objs.push_back(movingBlock);
```

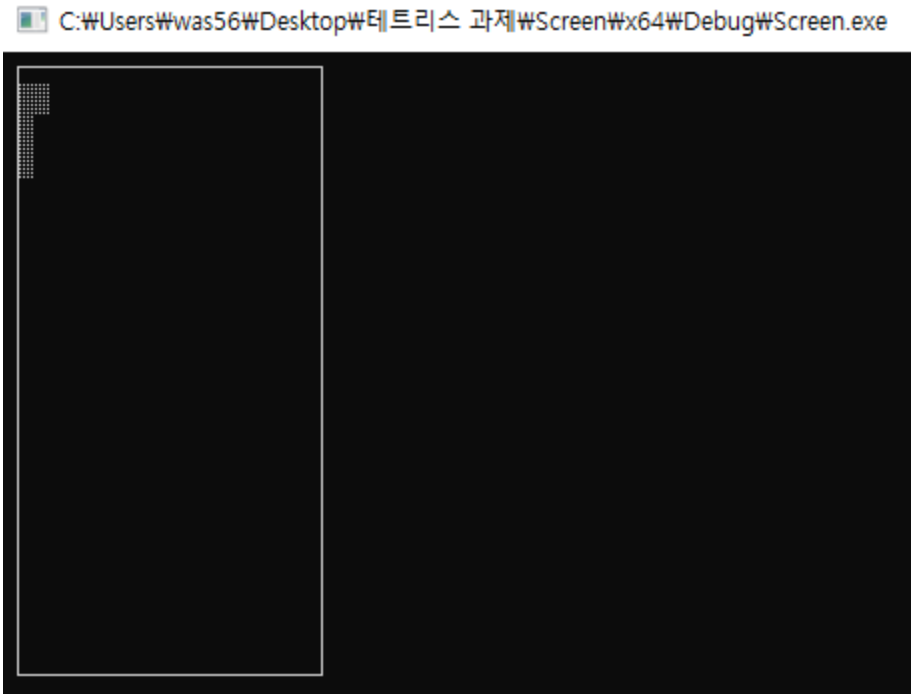
위 코드가 잘 작동시키기 위해 GameObject 에 생성자를 추가하였다.

```
// 생성자 추가
GameObject::GameObject(const GameObject* gameObject,
    const string& name, const string& tag)
    : name(name), tag(tag),
    enabled(true), parent(nullptr),
    transform(
        new Transform(
            this,
            gameObject->getConstTransform()->getShape(),
            gameObject->getConstTransform()->getPosition(),
            gameObject->getConstTransform()->getScale()
        )
    )
{
    if (name == "") {
        // name을 정하지 않았다면 부모의 이름을 씀
        this->name = gameObject->getName();
    }
    if (tag == "") {
        // tag를 정하지 않았다면 부모의 태그를 씀
        this->tag = gameObject->getTag();
    }
    components.clear();
    components.push_back(transform);
}
```

이로써 2 - ①을 구현하기 위한 객체들을 모두 생성했고 Transform.cpp 의 update 를 수정시켜 맵과 테트리스 블록을 출력하도록 만들었다.

```
void Transform::update()
{
    // 한바퀴 이상을 돌아왔을 때 그 만큼의 회전을 뺌
    if (rotation.x >= 360) {
        rotation.x -= 360;
    }

    // 맵일 때 맵을 그림
    // 맵을 그리는건 MapScript파일을 만들어 적용하는게 최적화는 잘 될거 같다.
    // 하지만 목적에 맞게 하는건 Transform이 맞는거 같다.
    if (gameObject->getTag() == "map") {
        Vector2 mapPosition = position;
        mapPosition.x -= 1, mapPosition.y -= 1;
        screen.drawRect(mapPosition, scale.X() + 2, scale.Y() + 2);
    }

    if (shape == "") return;
    // width, height가 0 이하일 때
    if (scale.x <= 0 || scale.y <= 0) return;
    // scale로 width, height를 표시
    screen.draw(shape.c_str(), scale.X(), scale.Y(), position);
}
```

위 코드들을 적용하여 컴파일한 결과 아래 사진과 같이 잘 작동하는 것을 볼 수 있다..



2 - ① 테트리스 블록들과 맵을 만들고 출력하기 구현 완료

2 - ② 테트리스 블록의 이동 및 회전 구현

먼저 블록에 적용할 스크립트를 만들며 start 함수에는 블록의 처음 위치를 설정하고 update 함수에는 키를 받고 움직이거나 회전하는 함수를 실행시키도록 만든다.

```
class BlockScript :
    public Component
{
    float speed;

public:
    BlockScript(GameObject* gameObject);
    ~BlockScript();

protected:

    void start();

    void update();

};
```

```
BlockScript::BlockScript(GameObject* gameObject)
    : Component(gameObject), speed(0.05f)
{
}
```

```
void BlockScript::start()
{
    if (!GameObject::Find("map")) return;
    // 맵 x축으로 가운데에 나오게끔 위치 저장
    int x = GameObject::Find("map")->getConstTransform()->getScale().X() / 2;
    // 맵에 나올 블럭 위치 설정
    this->gameObject->getTransform()->setPosition(x, 1);
}

void BlockScript::update()
{
    this->gameObject->getTransform()->plusPosition(Vector2::up * speed);

    if (Input::GetKeyDown(KeyCode::Down)) {
        this->gameObject->getTransform()->moveDown();
        // 2번을 동시에 내려가지 않도록 소수점 날림
        this->gameObject->getTransform()->setPosition(
            this->gameObject->getTransform()->getPosition().X(),
            this->gameObject->getTransform()->getPosition().Y()
        );
    }
    if (Input::GetKeyDown(KeyCode::Left)) {
        this->gameObject->getTransform()->moveLeft();
    }
    if (Input::GetKeyDown(KeyCode::Right)) {
        this->gameObject->getTransform()->moveRight();
    }
    if (Input::GetKeyDown(KeyCode::Up)) {
        // rotation의 x 증가 및 블럭의 회전
        this->gameObject->getTransform()->rotateShape();
    }
}
```

위 코드가 잘 작동할 수 있도록 GmaeObject 파일에 Find 함수를 구현한다.

```
GameObject* GameObject::Find(const string& path) {
    for (auto gameObject : GameObject::gameObjects) {
        if (gameObject->getName() == path) {
            return gameObject;
        }
    }
    return nullptr;
}
```

그리고 Utils.h 에 Vector2 operator*을 구현한다.

```
static friend Vector2 operator*(const Vector2& a, const float b)
{ return Vector2(a.x * b, a.y * b); }
```

마지막으로 Transform 파일에 rotateShape 함수를 구현한다.

```
// 블럭이 회전하고 난 후 블럭의 모양을 설정하는 함수
void rotateShape();
```

```
void Transform::rotateShape()
{
    rotate();
    string shape;
    // 서범주 교수님의 상속 모델로 구현한 테트리스 코드 중
    char* tempShape = new char[scale.X()*scale.Y()];
    for (int y = 0; y < scale.Y(); y++)
        for (int x = 0; x < scale.X(); x++)
            tempShape[(scale.X() - 1 - x)*scale.Y() + y]
            = this->shape.at(y*scale.X() + x);
    shape.assign(tempShape);
    setShape(shape);
    swap(scale.x, scale.y);
    delete[] tempShape;
}
```

이후 GameEngine.cpp 파일에 추가했던 movingBlock 객체에 BlockScript 컴포넌트를 추가시킨다.

```
GameObject* movingBlock = new GameObject(
    blockSet->children.at(rand() % 7), "mino", "movingBlock"
);
movingBlock->setParent(map);
movingBlock->addComponent<BlockScript>();
objs.push_back(movingBlock);
```

참고로 setParent 함수에 부모의 자식 배열에 자식이 들어가도록 수정하였다.

```
void setParent(GameObject* parent)
{
    this->parent = parent;
    // 부모의 자식으로 설정될 경우 자식 배열에 넣는다.
    parent->children.push_back(this);
}
```

위 코드들을 적용한 컴파일한 결과 아래 사진들을 통해 잘 작동하는 것을 볼 수 있다.

2 - ② 테트리스 블록의 이동 및 회전 구현 완료

2 - ③ 테트리스 블록 판정, 다음 블록, 다음 패널 추가 구현

먼저 다음 패널을 만들고 맵처럼 출력하기 위해 태그를 map 으로 설정했다.

```
GameObject* nextPanel = new GameObject(
    "nextPanel", nullptr, "map", "",
    Vector2{ 35, 1 }, Vector2{ 10, 10 }
);
objs.push_back(nextPanel);
```

다음 블록 표시 및 정보를 알기 위해 생성하였다.

```
GameObject* nextBlock = new GameObject(
    blockSet->children.at(rand() % 7), "nextMino", "nextBlock"
);
nextBlock->setParent(nextPanel);
nextBlock->addComponent<NextBlockScript>();
objs.push_back(nextBlock);
```

NextBlockScript 스크립트를 작성하기 전에 테트리스 블록이 바닥이나 블록에 의해 드랍 판정이 되게끔 하기 위해 map 에 MapScript 컴포넌트를 넣었다.

```
GameObject* map = new GameObject(
    "map", nullptr, "map", "",
    // 왼쪽부터 position, scale(width, height)를 위해 추가
    Vector2{ 1, 1 }, Vector2{ 30, 40 }
);
map->addComponent<MapScript>();
objs.push_back(map);
```

그 후 MapScript 스크립트는 블록이 바닥에 닿고 그 블록이 어디에 놓고 어떤 모양이 되었는지 기억해야하기 때문에 map 이라는 변수를 만들고 동적할당하여 map 의 역할을 하도록 만들었다. place 라는 함수를 이용해 블록이 놓였을 때 map 에 블록이 저장된다. changeChar 는 변수 map 을 기반으로 스크린에 출력해야하기 때문에 map 을 출력하도록 string 으로 바꾸고 shape 에 적용하는 함수이다.

```
class MapScript :
    public Component
{
    // 블럭 판정을 위한 맵
    bool* map;
    char* shape;

public:
    MapScript(GameObject* gameObject);
    ~MapScript();

protected:

    void start();

    void update();

private:

    // 서범주 교수님의 상속 모델로 구현한 테트리스 코드 중
    void place(const string& shape, const Vector2& pos, int w, int h);

    // map을 string으로 바꾸기 위한 함수
    void changeChar();

};
```

```
MapScript::MapScript(GameObject* gameObject)
    : Component(gameObject),
    map(
        new bool[
            (gameObject->getTransform()->getScale().X()) *
            (gameObject->getTransform()->getScale().Y())
    ]),
    shape(new char[
        this->gameObject->getTransform()->getScale().X() *
            this->gameObject->getTransform()->getScale().Y()
    ])
{
}


MapScript::~MapScript()
{
    if (map) { delete[] map; }
    if (shape) { delete[] shape; }
}
```

```cpp
void MapScript::start()
{
    for (int i = 0;
        i < gameObject->getTransform()->getScale().Y();
        i++)
    {
        for (int j = 0;
            j < gameObject->getTransform()->getScale().X();
            j++)
        {
            map[i * gameObject->getTransform()->getScale().X() + j]
                = false;
        }
    }
}

void MapScript::update()
{
    if (!GameObject::Find("mino")) return;

    auto movingBlock = GameObject::Find("mino");

    if (!movingBlock->isActive()) {
        Transform* movingBlockTransform
            = movingBlock->getTransform();
        // 놓기
        place(
            movingBlockTransform->getShape(),
            movingBlockTransform->getPosition(),
            movingBlockTransform->getScale().X(),
            movingBlockTransform->getScale().Y()
        );
    }
    // map을 string으로 바꾼 후 shape에 적용
    changeChar();
}
```

```cpp
void MapScript::place(const string& shape, const Vector2 & pos, int w, int h)
{
    Vector2 correction = Vector2(pos.x - 1, pos.y - 1);
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            if (shape[j + i * w] != ' ')
                map[
                    correction.X() + j + (correction.Y() + i) *
                        this->gameObject->getTransform()->getScale().X()
                ] = true;
        }
    }
}

void MapScript::changeChar()
{
    for (int i = 0;
        i < gameObject->getTransform()->getScale().Y();
        i++)
    {
        for (int j = 0;
            j < gameObject->getTransform()->getScale().X();
            j++)
        {
            if (map[i * gameObject->getTransform()->getScale().X() + j]) {
                shape[i * gameObject->getTransform()->getScale().X() + j] = '\xB2';
            }
            else {
                shape[i * gameObject->getTransform()->getScale().X() + j] = ' ';
            }
        }
    }
    this->gameObject->getTransform()->setShape(shape);
}
```

이후 블록이 바닥이나 블록에 의해 드랍 판정하는 부분을 BlockScript 에 구현하였으며 테트리스 코드에 있는 함수를 참고했다.

```
class BlockScript :
    public Component
{
    float speed;

public:
    BlockScript(GameObject* gameObject);
    ~BlockScript();

protected:

    void start();

    void update();

private:
    // 서범주 교수님의 상속 모델로 구현한 테트리스 중
    bool isGrounded();

};
```

```
void BlockScript::update()
{
    Transform* myTransform = this->gameObject->getTransform();

    myTransform->plusPosition(Vector2::up * speed);

    if (Input::GetKeyDown(KeyCode::Down)) {
        myTransform->moveDown();
        // 2번을 동시에 내려가지 않도록 소수점 날림
        myTransform->setPosition(
            myTransform->getPosition().X(),
            myTransform->getPosition().Y()
        );
    }
    if (Input::GetKeyDown(KeyCode::Left)) {
        myTransform->moveLeft();
    }
    if (Input::GetKeyDown(KeyCode::Right)) {
        myTransform->moveRight();
    }
    if (Input::GetKeyDown(KeyCode::Up)) {
        // rotation의 x 증가 및 블럭의 회전
        myTransform->rotateShape();
    }
    if (Input::GetKeyDown(KeyCode::Space)) {
        // 땅이나 블럭에 닿을 때 까지 블럭을 내림
        while (!isGrounded()) {
            this->gameObject->getTransform()->moveDown();
        }
    }

    if (isGrounded()) {
        this->gameObject->setActive(false);
    }
}
```

```
bool BlockScript::isGrounded()
{
    // 서범주 교수님의 상속 모델로 구현한 테트리스 코드 중
    Vector2 pos = this->gameObject->getTransform()->getPosition();
    pos.x -= 1, pos.y -= 1;
    int h = this->gameObject->getTransform()->getScale().Y();
    int w = this->gameObject->getTransform()->getScale().X();
    string shape = this->gameObject->getTransform()->getShape();

    if (!GameObject::Find("map")) { return false; }
    Transform* mapTransform = GameObject::Find("map")->getTransform();

    if (pos.Y() + h >= mapTransform->getScale().Y()) return true;

    for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; j++)
            if (mapTransform->getShape()[
                pos.X() + j + (pos.Y() + i + 1) *
                    mapTransform->getScale().X()
            ] != ' ' && shape[j + i * w] != ' ')
                return true;
    }
    return false;
}
```

이 부분에서 끝내면 블록이 드랍 판정이 되고 새로운 블록이 나올 수 없으므로 NextBlock 의 NextBlockScript 컴포넌트를 이용하여 드랍 판정 된 블록을 처음 나왔던 것처럼 설정하도록 만든다.

```
class NextBlockScript :
    public Component
{
public:
    NextBlockScript(GameObject* gameObject);
    ~NextBlockScript();

protected:

    void start();

    void update();

private:

};
```

```cpp
void NextBlockScript::start()
{
    if (!GameObject::Find("nextPanel")) return;
    Transform* nextPanelTransform = GameObject::Find("nextPanel")->getTransform();
    // nextBlock패널의 x축으로 가운데에 나오게끔 위치 저장
    int x = nextPanelTransform->getScale().X() / 2
        + nextPanelTransform->getPosition().X();
    int y = nextPanelTransform->getPosition().Y() + 4;
    // nextBlock패널에 나올 블럭 위치 설정
    this->gameObject->getTransform()->setPosition(x, y);
}

void NextBlockScript::update()
{
    if (!GameObject::Find("mino")) { return; }
    GameObject* block = GameObject::Find("mino");
    Transform* blockTransform = block->getTransform();
    Transform* myTransform = this->gameObject->getTransform();

    // 블럭의 역할이 끝났으면 nextBlock 정보들을 Block정보에 옮김
    if (!block->isActive()) {

        if (!GameObject::Find("map")) { return; }
        Transform* mapTransform = GameObject::Find("map")->getTransform();
        int x = mapTransform->getScale().X() / 2
            + mapTransform->getPosition().X();

        // block에 정보를 넣고 다시 활성화
        blockTransform->setPosition(x, 1);
        blockTransform->setScale(myTransform->getScale());
        blockTransform->setShape(myTransform->getShape());
        block->setActive(true);

        // blockSet에서 새로운 정보를 nextBlock에 넣음
        if (!GameObject::Find("blockSet")) { return; }
        GameObject* nextBlock
            = GameObject::Find("blockSet")->getChildren().at(rand() % 7);
        myTransform->setShape(nextBlock->getTransform()->getShape());
        myTransform->setScale(nextBlock->getTransform()->getScale());
    }
}
```

추가적으로 GameObject 함수에서 버그가 일어나는 부분이 있어서 주석 처리하였고 위 코드들이 잘 작동하도록 Transform 파일에서 오버로딩을 이용해 함수를 구현하였다.

```
void GameObject::traverseStart() {
    if (enabled == false) return;

    for (auto comp : components)
    {
        comp->start();
    }
    /*
    for (auto child : children)
    {
        child->traverseStart();
    }
    */
}

void GameObject::traverseUpdate() {
    if (enabled == false) return;

    for (auto comp : components)
    {
        comp->update();
    }
    /*
    for (auto child : children)
    {
        child->traverseUpdate();
    }
    */
}
```

```
public:

    // 블럭의 위치를 옮기기 위해 만든 함수
    // 테트리스의 하드 드롭을 구현하기 위함
    void setPosition(float x, float y)
    { position = Vector2(x, y); }

    void setPosition(Vector2& pos)
    { position.x = pos.x, position.y = pos.y; }

    // 블럭의 위치를 더하기 위해 만든 함수
    // sleep이 거칠 때 마다 내려가지 않게 하기 위해 만들었다.
    void plusPosition(float x, float y)
    { position.x += x, position += y; }

    void plusPosition(Vector2& plusPos)
    {
        position.x = position.x + plusPos.x;
        position.y = position.y + plusPos.y;
    }

    // 블럭의 width와 height를 조정하기 위해 만든 함수
    void setScale(float x, float y)
    { scale = Vector2(x, y); }

    void setScale(const Vector2& scale)
    {
        this->scale = Vector2(scale.x, scale.y);
    }
```

위 코드들을 적용시키고 컴파일한 결과 아래 사진들처럼 잘 작동하였다.

2 - ③ 테트리스 블록 판정, 다음 블록, 다음 패널 구현 완료

2 - ④ 테트리스 블록 줄 지우기, 점수, 최고 점수, 게임 오버 추가 구현

먼저 테트리스 블록 줄을 지우기 위해 테트리스 코드에서 한 라인이 채워지면 지우는 함수인 evaluateLine 함수를 참고했으며 map 을 읽어야하기 때문에 MapScript 의 update 함수를 이용해 줄을 지우도록 구현하였다.

```
// 모든 라인이 true인지 확인
bool isLineAllOccupied(int line);

bool evaluateLine(int line);
```

```
bool MapScript::isLineAllOccupied(int line)
{
    Transform* mapTransform = this->gameObject->getTransform();
    int width = mapTransform->getScale().X();

    // 한 라인이 전부 true인지 확인
    for (int i = 0; i < width; i++) {
        if (!map[line*width + i]) {
            return false;
        }
    }
    return true;
}
```

```
bool MapScript::evaluateLine(int line) {
    if (!isLineAllOccupied(line)) return false;

    Transform* mapTransform = this->gameObject->getTransform();
    int width = mapTransform->getScale().X();
    int height = mapTransform->getScale().Y();

    // clean the given line
    for (int i = 0; i < width; i++) map[line*width + i] = false;

    // copy lines above the "line" down below their below lines.
    for (int i = height - 1; i >= 0; i--) {
        for (int j = 0; j < width; j++) {
            // 지워지는 줄은 위의 줄의 데이터들을 가져와 적용
            map[(i + 1)*width + j] = map[i*width + j];
            // 지워진 줄의 윗 줄은 false로 비움
            map[i*width + j] = false;
        }
    }
    return true;
}
```

```
void MapScript::update()
{

    if (!GameObject::Find("mino")) return;
    // 맵에 있는 블럭 정보 불러오기
    GameObject* movingBlock = GameObject::Find("mino");

    // 맵에 있는 블럭이 드랍 판정이 났을 때
    if (!movingBlock->isActive()) {
        Transform* movingBlockTransform
            = movingBlock->getTransform();
        // 놓기
        place(
            movingBlockTransform->getShape(),
            movingBlockTransform->getPosition(),
            movingBlockTransform->getScale().X(),
            movingBlockTransform->getScale().Y()
        );

        int blockHeight = movingBlockTransform->getScale().Y();
        // 한번에 지운 줄 갯수
        int countRemoveLine = 0;
        // 라인들을 판정함
        for (int i = 0; i < blockHeight; i++) {
            if (evaluateLine(movingBlockTransform->getPosition().Y() + i)) {
                countRemoveLine++;
            }
        }
```

이 부분까지 완성을 하면 정상적으로 게임을 즐길 수 있을 정도로 구현이 된다. 그 후로는 추가적인 부분이며 먼저 블록에 next 를 표시하게끔 하기 위해 nextText 라는 GameObject 와 NextTextScript 스크립트에서 잘 표시하게끔 구현하였다.

```
// next를 출력하기 위한 객체
GameObject* nextText = new GameObject(
    "nextText", nextPanel, "text", "Next"
);
nextText->addComponent<NextTextScript>();
objs.push_back(nextText);
```

```
class NextTextScript :
    public Component
{

public:
    NextTextScript(GameObject* gameObject);
    ~NextTextScript();

protected:

    void start();

    void update();

};
```

```
void NextTextScript::start()
{
    // nextPanel의 정보 불러오기
    if (!GameObject::Find("nextPanel")) { return; }
    Transform* nextPanelTransform
        = GameObject::Find("nextPanel")->getTransform();

    // nextPanel 내 적절한 위치 저장
    int x = nextPanelTransform->getPosition().X()
        + nextPanelTransform->getScale().X() / 2;
    int y = nextPanelTransform->getPosition().Y() + 2;

    // nextPanel 내 적절한 위치 적용
    Transform* textTransform = this->gameObject->getTransform();
    textTransform->setPosition(x, y);
    // shape 문자열만 출력하도록 scale 조정
    textTransform->setScale(textTransform->getShape().size(), 1);
}

void NextTextScript::update()
{
    // shape 문자열만 출력하도록 scale 조정
    Transform* textTransform = this->gameObject->getTransform();
    textTransform->setScale(textTransform->getShape().size(), 1);
}
```

다음으로는 점수를 구현했으며 GameObject 인 score 과 ScoreScript 를 통해 점수를 출력하고 점수를 저장하도록 구현했다.

```
class ScoreScript :
    public Component
{
    int score;

public:
    ScoreScript(GameObject* gameObject);
    ~ScoreScript();

protected:

    void start();

    void update();
};
```

```
ScoreScript::ScoreScript(GameObject* gameObject)
    : Component(gameObject), score(0)

{
}
```

```
void ScoreScript::start()
{
    // nextPanel의 정보를 받음
    if (!GameObject::Find("nextPanel")) { return; }
    Transform* panelTransform = GameObject::Find("nextPanel")->getTransform();
    Transform* myTransform = this->gameObject->getTransform();

    // nextPanel 내 적절한 위치 저장
    int x = panelTransform->getPosition().X() +
        panelTransform->getScale().X() / 2 - 7;
    int y = panelTransform->getPosition().Y() +
        panelTransform->getScale().Y() / 2 + 4;

    // nextPanel 내 적절한 위치에 지정
    myTransform->setPosition(x, y);

    // 점수 문자열 처리
    string text = this->gameObject->getTransform()->getShape();
    text.append(std::to_string(score));
    myTransform->setShape(text);

    // 문자열 만큼 출력하게끔 scale 조정
    myTransform->setScale(text.size(), 1);
}

void ScoreScript::update()
{
    // 이 객체의 shape를 통해 점수 저장
    Transform* myTransform = this->gameObject->getTransform();
    string scoreBuf = myTransform->getShape();
    score = atoi(scoreBuf.erase(0, 8).c_str());

    // 문자열 만큼 출력하게끔 scale 조정
    myTransform->setScale(myTransform->getShape().size(), 1);
}
```

이 부분까지만 하면 점수를 갱신할 수 없기 때문에 테트리스 블록 줄이 없어지는 것을 판정하는 MapScript 에 점수를 갱신하는 부분과 여러 줄을 한번에 지웠을 때 보너스 점수가 더하고 점수를 갱신하도록 구현하였다.

```
// 줄을 한 번에 지운 갯수에 따른 점수 배열
static vector<int> bonusScore = { 0, 50, 150, 300, 500 };

int blockHeight = movingBlockTransform->getScale().Y();
// 한번에 지운 줄 갯수
int countRemoveLine = 0;
// 라인들을 판정함
for (int i = 0; i < blockHeight; i++) {
    if (evaluateLine(movingBlockTransform->getPosition().Y() + i)) {
        countRemoveLine++;
    }
}

// score 정보를 불러옴
if (!GameObject::Find("score")) { return; }
Transform* scoreTransform = GameObject::Find("score")->getTransform();

// 점수만 추출
string scoreBuf = scoreTransform->getShape();
scoreBuf.erase(0, 8);
int score = atoi(scoreBuf.c_str());
// 점수를 더함
score += bonusScore.at(countRemoveLine);
string text = "score : ";
text.append(std::to_string(score));
// 점수 적용
scoreTransform->setShape(text);
```

그 다음으로는 게임 오버를 GameObject 인 gameOver 와 GameOverScript 를 이용하여 게임 오버의 active 를 true 로 설정하면 모든 게임의 초기화와 동시에 게임이 멈추도록 만들고 Enter 키를 누르면 다시 시작하도록 구현하였다.

```
// Game Over을 나타내기 위한 객체
GameObject* gameOver = new GameObject(
    "gameOver", nullptr, "status", "Game Over"
    ,Vector2(15, 15), Vector2(9, 1)
);
gameOver->addComponent<GameOverScript>();
objs.push_back(gameOver);
gameOver->setActive(false);
```

```
void GameOverScript::update()
{
    // 맵안에 있는 블럭의 작동을 멈춤
    if (!GameObject::Find("mino")) { return; }
    GameObject* movingBlock = GameObject::Find("mino");
    movingBlock->setActive(false);

    // 다음 블럭의 작동도 멈춤
    if (!GameObject::Find("nextMino")) { return; }
    GameObject* nextBlock = GameObject::Find("nextMino");
    nextBlock->setActive(false);

    // 엔터 키를 누르면 새로운 게임 진행
    if (Input::GetKeyDown(KeyCode::Enter)) {
        movingBlock->setActive(true);
        nextBlock->setActive(true);
        this->gameObject->setActive(false);
    }
}
```

게임 오버 판정은 MapScript 에 블록이 드랍 판정되었을 때 isLineTrue 함수를 통해 그 블록이 맵의 맨 윗줄에 존재하면 게임이 끝나도록 구현했다.

```
bool MapScript::isLineTrue(int line)
{
    Transform* mapTransform = this->gameObject->getTransform();
    int width = mapTransform->getScale().X();

    for (int i = 0; i < width; i++) {
        if (map[line*width + i]) {
            return true;
        }
    }
    return false;
}
```

```
// 블럭의 맨 위가 다 찼을 경우 게임 오버
if (isLineTrue(0)) {
    // 게임 오버 활성화
    if (!GameObject::Find("gameOver")) { return; }
    GameObject::Find("gameOver")->setActive(true);
    // map 초기화
    start();
```

마지막으로 최고 점수를 구현했으며 GameObject 인 bestScore 와 BestScoreScript 스크립트를 이용해 최고 점수를 출력하고 저장하도록 구현했다. 참고로 BestScoreScript 스크립트는 ScoreScript 스크립트와 매우 유사하게 구현했다.

```
class BestScoreScript :
    public Component
{
    int bestScore;

public:
    BestScoreScript(GameObject* gameObject);
    ~BestScoreScript();

protected:

    void start();

    void update();
};
```

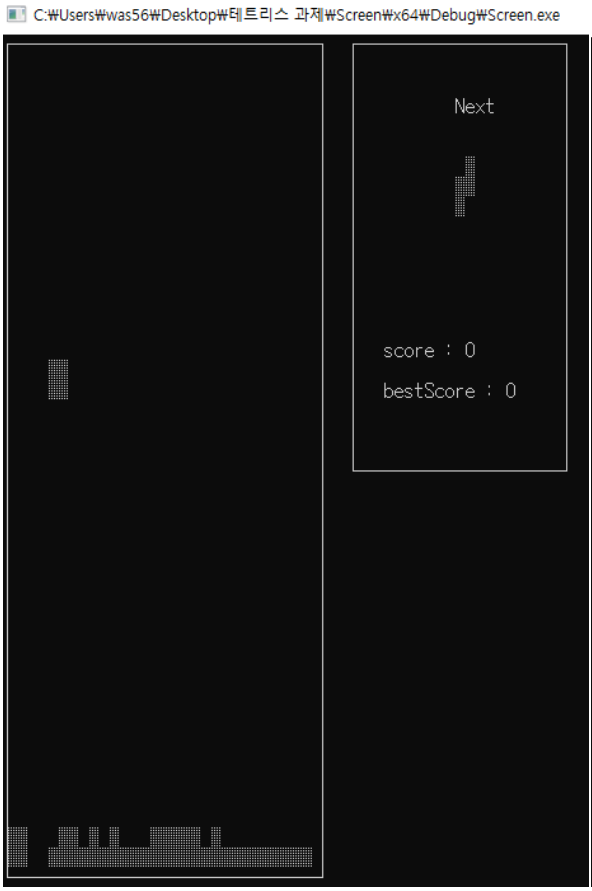
```
BestScoreScript::BestScoreScript(GameObject* gameObject)
    : Component(gameObject), bestScore(0)

{
}
```

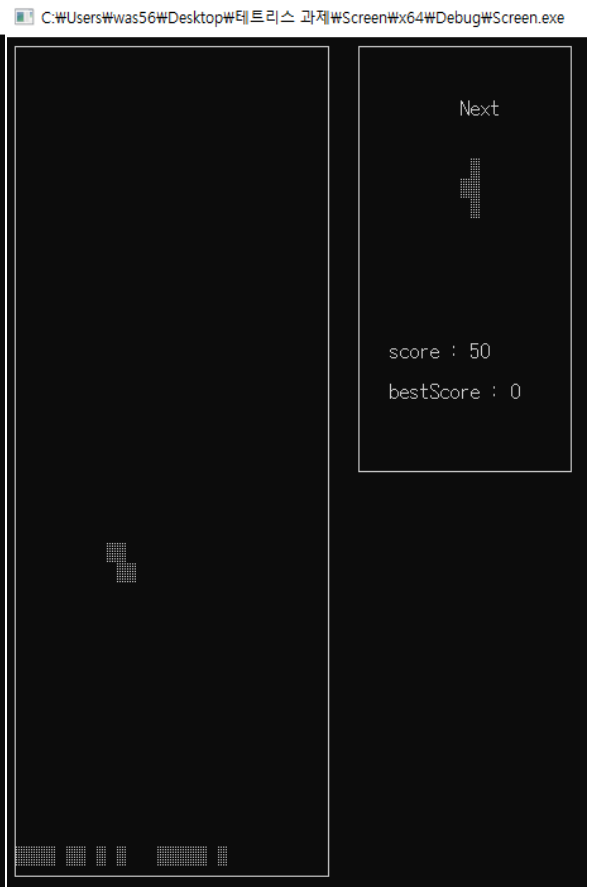
```
void BestScoreScript::start()
{
    // nextPanel의 정보를 받음
    if (!GameObject::Find("nextPanel")) { return; }
    Transform* panelTransform = GameObject::Find("nextPanel")->getTransform();
    Transform* myTransform = this->gameObject->getTransform();

    // nextPanel 내 적절한 위치 저장
    int x = panelTransform->getPosition().X() +
        panelTransform->getScale().X() / 2 - 7;
    int y = panelTransform->getPosition().Y() +
        panelTransform->getScale().Y() / 2 + 6;

    // nextPanel 내 적절한 위치에 지정
    myTransform->setPosition(x, y);

    // 점수 문자열 처리
    string text = this->gameObject->getTransform()->getShape();
    text.append(std::to_string(bestScore));
    myTransform->setShape(text);

    // 문자열 만큼 출력하게끔 scale 조정
    myTransform->setScale(text.size(), 1);
}

void BestScoreScript::update()
{
    // 이 객체의 shape를 통해 bestScore 저장
    Transform* myTransform = this->gameObject->getTransform();
    string bestScoreBuf = myTransform->getShape();
    bestScore = atoi(bestScoreBuf.erase(0, 12).c_str());

    // 문자열 만큼 출력하게끔 scale 조정
    myTransform->setScale(myTransform->getShape().size(), 1);
}
```

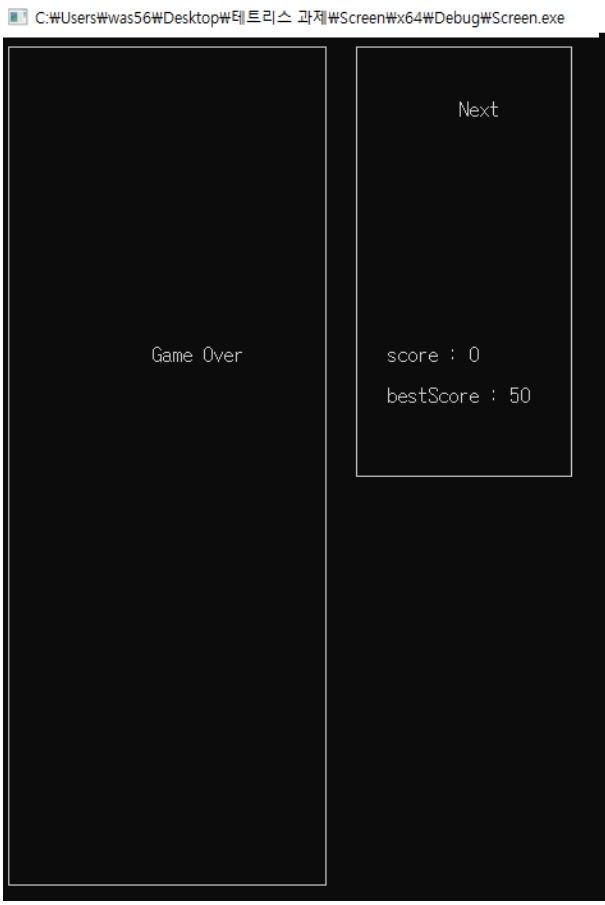
최고 점수도 마찬가지로 이 부분까지 구현하면 최고 점수를 갱신할 수 없기 때문에 게임 오버를 판정하는 MapScript 에 게임이 끝나는 지점에서 최고 점수를 갱신하도록 구현하였다.

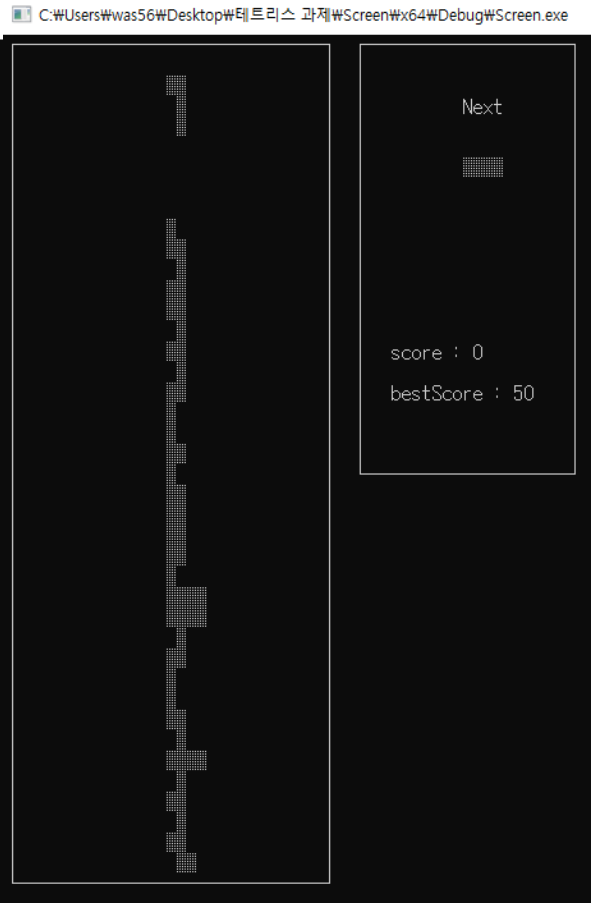
```
// 블럭의 맨 위가 다 찼을 경우 게임 오버
if (isLineTrue(0)) {
    // 게임 오버 활성화
    if (!GameObject::Find("gameOver")) { return; }
    GameObject::Find("gameOver")->setActive(true);
    // map 초기화
    start();
    // 최고 점수 정보 받음
    if (!GameObject::Find("bestScore")) { return; }
    Transform* bestScoreTransform
        = GameObject::Find("bestScore")->getTransform();
    string bestScoreBuf = bestScoreTransform->getShape();
    int bestScore = atoi(bestScoreBuf.erase(0, 12).c_str());

    // 최고 점수보다 높은 점수일 경우 최고 점수 갱신
    if (score > bestScore) {
        bestScore = score;
    }
    string bestScoreText = "bestScore : ";
    bestScoreText.append(std::to_string(bestScore));
    bestScoreTransform->setShape(bestScoreText);

    // 점수 초기화
    scoreTransform->setShape("score : 0");
}
```

이 부분까지 구현하고 컴파일한 결과 아래 사진들과 같이 잘 작동하는 것을 볼 수 있다.









테트리스 플레이 영상은 아래 사이트에 업로드해 놓았다.

https://was564.github.io/study_html/Tetris.html

참고 사이트:

https://github.com/beomjoo90/OOP/commit/3ccf371b4b9bffed226b781ab17029090580c23b