* 지금까지 만든것은 게임 오브젝트를 터치로 이동시켜서 게임 오브젝트를 교환하는 부분밖에 없음

* 7-1에서는 게임 오브젝트를 이동시켰을 때 같은 색상의 게임 오브젝트가 3개 이상 나열되어 매칭되면 오브젝트를 삭제하는 기능 구현

```
void GameObject::SlidingCompleteHandler()
{
    int x1 = m_prevBoardX;
    int y1 = m_prevBoardY;
    int x2 = m_targetBoardX;
    int y2 = m_targetBoardY;

    m_pGameLayer->SlidingFinished(x1, y1, x2, y2);
}
```

```
void GameObject::SetGameLayer(GameLayer* pGameLayer)
{
    m_pGameLayer = pGameLayer;
}
```

```
void GameLayer::StartGame()
{
    srand(time(NULL));

    for (int x=0; x<COLUMN_COUNT; ++x) {
        for (int y=0; y<ROW_COUNT; ++y) {

            int type = rand() % TYPE_COUNT;

            GameObject* pGameObject = GameObject::Create(type);
            m_pBoard[x][y] = pGameObject;

            pGameObject->setAnchorPoint(ccp(0, 1));
            pGameObject->setPosition(Common::ComputeXY(x, y));
            pGameObject->SetGameLayer(this);
            addChild(pGameObject, zGameObject);
        }
    }
```

```
    m_bTouchStarted = false;
    m_numOfSlidingObjects = 0;
}
```

* 이제 실제로 게임 오브젝트가 이동을 완료했을 때의 처리를 구현

```
class GameLayer : public CCLayer
{
public:
    void SlidingFinished(int x1, int y1, int x2, int y2);

protected:
    // 이동 중인 오브젝트의 수
    int m_numOfSlidingObjects;
```

* 이동중인 오브젝트의 수는 왜 저장하는걸까? 게임오브젝트를 이동시키면 하나만 이동되는게 아니고 2개의 오브젝트가 이동함. 하지만 한번의 이벤트만 처리하면 되기 때문에 모든 오브젝트의 이동이 완료된 후 한번만 이벤트를 처리하기 위해서.

```
void GameLayer::SlidingFinished(int x1, int y1, int x2, int y2)
{
    --m_numOfSlidingObjects;
    if (m_numOfSlidingObjects == 0) {

    }
}
```

```
void GameLayer::SwapObjects(int x1, int y1, int x2, int y2)
{
    m_numOfSlidingObjects = 2;
    ...
}
```

* 오브젝트 이동이 완료되면 한번만 매칭 로직이 수행되도록 만들었음. 실제 매칭 로직을 계속 만들어보자.

```
void GameLayer::SlidingFinished(int x1, int y1, int x2, int y2)
{
```

```
    --m_numOfSlidingObjects;
    if (m_numOfSlidingObjects == 0)
    {
        if (IsStreak(x1, y1) || IsStreak(x2, y2))
        {
            // 매칭된 줄은 삭제한다.
            if (IsStreak(x1, y1))
            {
                RemoveObject(x1, y1);
            }

            if (IsStreak(x2, y2))
            {
                RemoveObject(x2, y2);
            }
        }
    }
}
```

```
bool GameLayer::IsStreak(int x, int y)
{
    return StreakHorz(x, y) > 2 || StreakVert(x, y) > 2;
}
```

```
int GameLayer::StreakHorz(int x, int y)
{
    if (x < 0 || x >= COLUMN_COUNT) {
        return 0;
    }

    if (y < 0 || y >= ROW_COUNT) {
        return 0;
    }

    GameObject* pCurrentGameObject = m_pBoard[x][y];
    if (pCurrentGameObject == NULL) {
        return 0;
    }

    int streak = 1;
    int temp = x;
```

```
    // 왼쪽으로 검사해나감
    while (CheckType(pCurrentGameObject->GetType(), temp - 1,
y)) {
        --temp;
        ++streak;
    }

    temp = x;

    // 오른쪽으로 검사해나감
    while (CheckType(pCurrentGameObject->GetType(), temp + 1,
y)) {
        ++temp;
        ++streak;
    }

    return streak;
}
```

```
bool GameLayer::CheckType(int type, int x, int y)
{
    if (x < 0 || x >= COLUMN_COUNT) {
        return false;
    }

    if (y < 0 || y >= ROW_COUNT) {
        return false;
    }

    if (m_pBoard[x][y] == NULL) {
        return false;
    }

    return type == m_pBoard[x][y]->GetType();
}
```

```
int GameLayer::StreakVert(int x, int y)
{
    if (x < 0 || x >= COLUMN_COUNT) {
        return 0;
    }

    if (y < 0 || y >= ROW_COUNT) {
        return 0;
```

```
    }

    GameObject* pCurrentGameObject = m_pBoard[x][y];
    if (pCurrentGameObject == NULL) {
        return 0;
    }

    int streak = 1;
    int temp = y;

    // 위쪽으로 검사해나감
    while (CheckType(pCurrentGameObject->GetType(), x, temp -
1)) {
        --temp;
        ++streak;
    }

    temp = y;

    // 아래쪽으로 검사해나감
    while (CheckType(pCurrentGameObject->GetType(), x, temp +
1)) {
        ++temp;
        ++streak;
    }

    return streak;
}
```

* 매칭을 검사하는 함수는 만들었으니, 이제 게임 오브젝트를 제거하는 함수인 RemoveObject 함수를 만들어 보겠음.

```
void GameLayer::RemoveObject(int x, int y)
{
    int currentType = m_pBoard[x][y]->GetType();
    int temp;

    std::vector<GameObject*> removedObjects;

    GameObject* pGameObject = m_pBoard[x][y];
    removedObjects.push_back(pGameObject);

    if (StreakHorz(x, y) > 2) {
        temp = x;
```

```
        while (CheckType(currentType, temp-1, y)) {
            pGameObject = m_pBoard[temp-1][y];
            removedObjects.push_back(pGameObject);

            --temp;
        }

        temp = x;

        while (CheckType(currentType, temp+1, y)) {
            pGameObject = m_pBoard[temp+1][y];
            removedObjects.push_back(pGameObject);
            ++temp;
        }
    }

    if (StreakVert(x, y) > 2) {
        temp = y;
        while (CheckType(currentType, x, temp-1)) {
            pGameObject = m_pBoard[x][temp-1];
            removedObjects.push_back(pGameObject);
            --temp;
        }

        temp = y;

        while (CheckType(currentType, x, temp+1)) {
            pGameObject = m_pBoard[x][temp+1];
            removedObjects.push_back(pGameObject);
            ++temp;
        }
    }

    for (int i=0; i<removedObjects.size(); ++i) {
        GameObject* pGameObject = removedObjects[i];
        if (pGameObject) {
            int boardX = pGameObject->GetTargetBoardX();
            int boardY = pGameObject->GetTargetBoardY();
            m_pBoard[boardX][boardY] = NULL;

            removeChild(pGameObject, true);
        }
    }

    removedObjects.clear();
}
```

* removeChild 함수

```
    /**
     * Removes a child from the container. It will also cleanup
all running actions depending on the cleanup parameter.
     *
     * @param child     The child node which will be removed.
     * @param cleanup   true if all running actions and
callbacks on the child node will be cleanup, false otherwise.
     */
    virtual void removeChild(CCNode* child, bool cleanup);
```

* 몇가지 다듬기 작업들

- GetTargetBoardX/Y 는 오브젝트가 이동했을때만 설정되었는데, 이동한적이 없었던 오브젝트를 매칭시키면 크래시
  - 객체 초기화시 설정으로 해결

```
void GameLayer::StartGame()
{
    srand(time(NULL));

    for (int x=0; x<COLUMN_COUNT; ++x) {
        for (int y=0; y<ROW_COUNT; ++y) {

            ...

            pGameObject->SetTargetBoardX(x);
            pGameObject->SetTargetBoardY(y);

            ...
        }
    }

    m_bTouchStarted = false;
    m_numOfSlidingObjects = 0;
}
```

- 게임 시작시 매칭되어있는 객체들이 사라지지 않음

- 게임 오브젝트를 처음 배치할 때 매칭되는 경우가 발생하지 않도록 수정

```
void GameLayer::StartGame()
{
    srand(time(NULL));

    for (int x=0; x<COLUMN_COUNT; ++x) {
        for (int y=0; y<ROW_COUNT; ++y) {

            GameObject* pGameObject = NULL;

            do {
                int type = rand() % TYPE_COUNT;

                if (pGameObject != NULL) {
                    CC_SAFE_DELETE(pGameObject);
                    m_pBoard[x][y] = NULL;
                }

                pGameObject = GameObject::Create(type);
```

```
                m_pBoard[x][y] = pGameObject;

            } while (IsStreak(x, y));

            pGameObject->setAnchorPoint(ccp(0, 1));
            pGameObject->setPosition(Common::ComputeXY(x, y));
            pGameObject->SetTargetBoardX(x);
            pGameObject->SetTargetBoardY(y);
            pGameObject->SetGameLayer(this);
            addChild(pGameObject, zGameObject);
        }
    }

    m_bTouchStarted = false;
    m_numOfSlidingObjects = 0;
}
```

- 게임 오브젝트가 사라진 위치에 검은색 빈 공간이 남게되는데 이 부분을 드래그해서 조작하려고 하면 크래시

- 터치된 위치에 게임 오브젝트가 존재하는지 확인하는 부분이 없음 -> 확인 후 객체가 없으면 무시

```
void GameLayer::ccTouchesBegan(CCSet* pTouches, CCEvent*
pEvent)
{
    if (m_bTouchStarted == false) {
        CCTouch* pTouch = (CCTouch*)pTouches->anyObject();
        CCPoint point = pTouch->getLocationInView();

        m_gestureStartBoardX = Common::ComputeBoardX(point.x);
        m_gestureStartBoardY = Common::ComputeBoardX(point.y);

        if (m_pBoard[m_gestureStartBoardX]
[m_gestureStartBoardY] == NULL) {
            return ;
        }

        m_bTouchStarted = true;
    }
}
```

```
void GameLayer::ccTouchesMoved(CCSet* pTouches, CCEvent*
pEvent)
{
    if (m_bTouchStarted) {
        CCTouch* pTouch = (CCTouch*)pTouches->anyObject();

        CCPoint point = pTouch->getLocationInView();

        int boardX = Common::ComputeBoardX(point.x);
        int boardY = Common::ComputeBoardX(point.y);

        if (m_pBoard[boardX][boardY] == NULL) {
            return ;
        }

        if (m_gestureStartBoardX != boardX ||
m_gestureStartBoardY != boardY) {
            if (IsAdjacent(m_gestureStartBoardX,
m_gestureStartBoardY, boardX, boardY)) {
                SwapObjects(m_gestureStartBoardX,
m_gestureStartBoardY, boardX, boardY);

            }

            m_bTouchStarted = false;
        }
    }
}
```

7.2 비매칭 시 복귀

* 복귀 시 처리 방식은 게임 오브젝트 매칭 시 이동처리와 거의 비슷함. 다만 이동 방향만 다름.

```
void GameLayer::SlidingFinished(int x1, int y1, int x2, int y2)
{
    --m_numOfSlidingObjects;
    if (m_numOfSlidingObjects == 0) {
        if (IsStreak(x1, y1) || IsStreak(x2, y2)) {
            // 매칭된 줄은 삭제한다.
            if (IsStreak(x1, y1)) {
                RemoveObject(x1, y1);
            }
```

```
            if (IsStreak(x2, y2)) {
                RemoveObject(x2, y2);
            }
        }
        else
        {
            // 매칭되지 않았을 경우 되돌린다.
            SwapObjects(x1, y1, x2, y2, true);
        }
    }
}
```

```
    void SwapObjects(int x1, int y1, int x2, int y2, bool
bRollback=false);
```

```
void GameObject::Rollback()
{
    CCPoint targetPosition = Common::ComputeXY(m_prevBoardX,
m_prevBoardY);
    CCPoint point = getPosition();

    CCMoveBy* pMoveBy = CCMoveBy::create(0.1f,
ccp(targetPosition.x - point.x, targetPosition.y - point.y));
    CCFiniteTimeAction* pAction = CCSequence::create(pMoveBy,
NULL);

    runAction(pAction);
}
```

7.3 게임 오브젝트 낙하 처리 및 생성

다음분이...