Design of the 3D Tetris Game

I. Preview of the game.

1. General view, what stays in the left-top corner is the score board.

2. Game Over

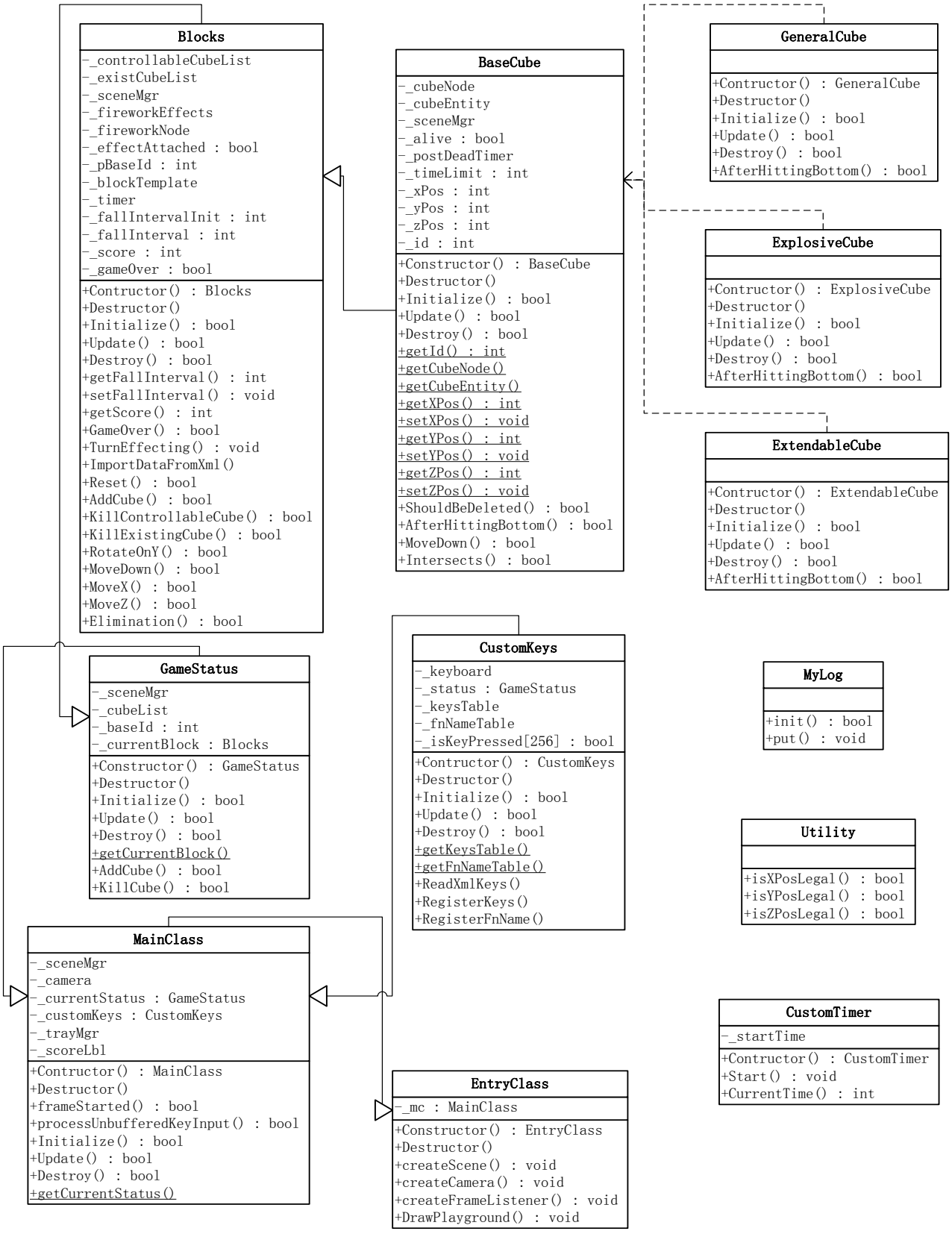
3. Effects when you eliminate one side successfully, and plus 100 point in the score

II. Game Manual

1. There are three kinds of cubes.
   a) General cube, applied with *SphereMappedRustySteel* material, and it has no special effects
   b) Explosive cube, applied with *OgreSpin* material (which you'll see OGRE heads spinning around), and when it hit the bottom (or other cubes), it'll eliminate the 6 cubes around it if there's one in the specified position, and eliminate itself.
   c) Extendable cube, applied with *SceneCubeMap1* material (it is like a mirror), and it'll extend another 6 cubes if these positions are empty when it hit the bottom (or other cubes).
2. You can use *'U', 'J', 'H', 'K'* to control the block moving on the surface, *'Z'* to rotate the block (you can only rotate it, yawing or pitching functions are forbidden), and *SPACE button* to speed up the falling.
3. The cubes will be eliminated only if when the surface of the scene is fully filled.
4. Just try it. :D

III. Design of the Game

1. Class View



2. Class Interpretation
    a) The architecture of the game confirm to the architecture described in class: View-Logic-Application.
    b) Application part is the *ExampleApplication* which is inherited by the *EntryClass*.
    c) View part is the *MainClass* which inherits the *FrameListener*, here I call it View class because I mainly take its *frameStarted* function as my drawing function, where most of the drawings are called or completed in this function.
    d) The other Classes are all Logic part.
    e) *GameStatus* class mainly controls the whole scene's logic; it has the *CustomKeys* class to help maintaining the key function.
    f) *Blocks* class is the controllable set of cubes, it has shapes which are imported from the xml file, it can fall down, move left, right, behind and front. It is the main part of the logic.
    g) There are three kinds of cubes, all inherit the *BaseCube* class, with the differences in the materials and the event after it hit the bottom (or other cubes).

IV. Detailed Implementation

1. Cubes --Inheritance
   a) *BaseCube* class is implemented to take control of the basic cube's properties and operations, such as the position (x, y, z), and initialization, update, destroy events. Three kinds of cubes have their own features, such as material, event after hit the bottom (or other cubes). This is implemented with the feature of C++, using such method:

      *ParentClass** pc = **new** *ChildClass*(…)

      Then, all the virtual functions in ParentClass will be override.
   b) Some xml scripts are imported before the game start: GDA03Cubes.xml and GDA03Keys.xml.
      i. GDA03Cubes.xml contains the kinds of the controllable blocks, in the example below, the set 01 contains the block consists of four cubes whose start positions are (1, MAX_Y, 1), (2, MAX_Y, 1), (1, MAX_Y, 2), (1, MAX_Y, 3), and the Types are (General, General, Explosive, General). The scripter can add some new kinds of blocks.
      Here is how the xml file looks like:

```
<Blocks>
    <Set name="01">
        <Cube>
            <X>1</X>
            <Z>1</Z>
            <Type>General</Type>
        </Cube>
        <Cube>
            <X>2</X>
            <Z>1</Z>
            <Type>General</Type>
        </Cube>
        <Cube>
            <X>1</X>
            <Z>2</Z>
            <Type>Explosive</Type>
        </Cube>
        <Cube>
            <X>1</X>
            <Z>3</Z>
            <Type>General</Type>
        </Cube>
    </Set>
    ……
</Blocks>
```

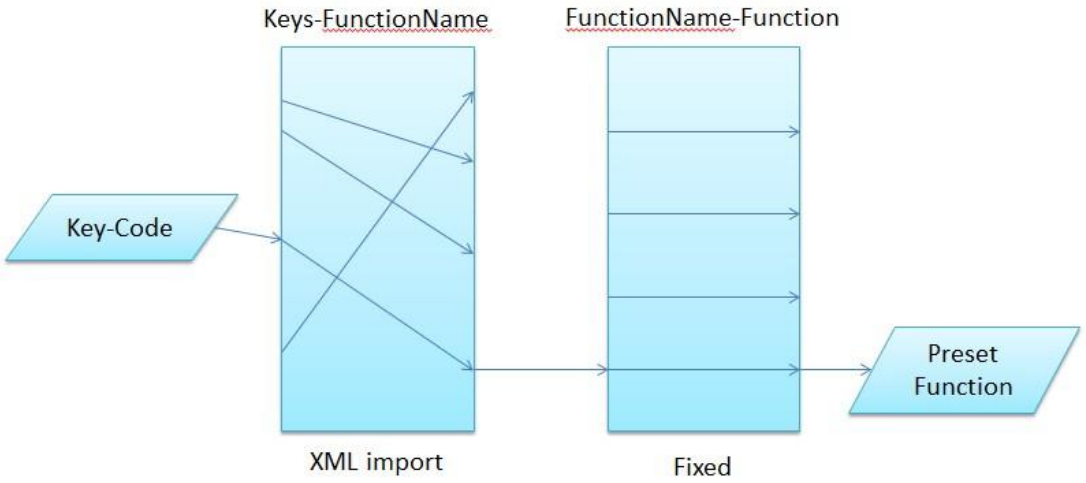
      ii. GDA03Keys.xml contains the information of keys functions, here I use button 'U' to control the block move on X axis in the positive direction. You can set rotate function on the ESC button if you like :D. The preset function-strings in the games are:

MoveXPositive	***MoveXNegative***
MoveZNegative	***MoveZPositive***
RotateOnY	***BlockFall***

Here is how the xml file looks like:

```
<KeysFunctions name="keys_functions">
    <Key name="KC_U">
        <KeyCode name="key_code">0x16</KeyCode>
        <EventName name="event_name">MoveXPositive</EventName>
    </Key>
    ……
</KeysFunctions>
```

How this is implemented:



c) Completely standard interfaces

i. Initialize function, to do all the initialization operations excluding the assignment on some variables (this should be completed in Constructor).

ii. Update function, update functions in each class will be called before the frame is drawn, so do all the frame-oriented update operation here.

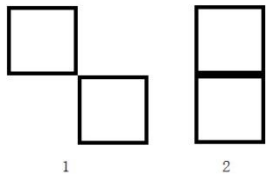
iii. Destroy function, to do all the garbage collecting work of this class, include de-allocate the memories.

Comment: I have this part here because the first edition of the code is completed screwed up due to the lack of standard interfaces, and it becomes a totally mass that I don't know where the function should be called. Then I delete it all, and start a new day, to start a new dream.

d) Custom AABB collision operation

Ogre3D has its own AABB operation, however, there are some problems when implementing with it.

i. Ogre3D think these situation to be a collision:



However this is not what I want, what I want is to set #1 to be not collided while the #2 is collided. First thought is that I can move the upper cube down a little to test the collision; however, I found the _getWorldAABB function has a comment:

/** Gets the axis-aligned bounding box of this node (and hence all subnodes).

@remarks
Recommended only if you are extending a SceneManager, because the bounding box returned from this method is only up to date after the SceneManager has called _update.
*/

Which means, if I move the cube down a little before _getWorldAABB (of course before the _update function), that wouldn't work.

So the custom AABB collision function needs to be implemented.

ii. Implementation

```
bool Intersects(BaseCube* other, MovingDirection dir, double offset = 0.1)
{
    if (!other->Alive()) {
        return false;
    }
    Ogre::Vector3 min1Corner = this->getCubeNode()->_getWorldAABB().getMinimum();
    Ogre::Vector3 max1Corner = this->getCubeNode()->_getWorldAABB().getMaximum();
    Ogre::Vector3 min2Corner = other->getCubeNode()->_getWorldAABB().getMinimum();
    Ogre::Vector3 max2Corner = other->getCubeNode()->_getWorldAABB().getMaximum();
    switch(dir)
    {
    case eMovingYNegative:
        min1Corner.y -= offset; max1Corner.y -= offset;
        break;
    case eMovingXPositive:
        min1Corner.x += offset; max1Corner.x += offset;
        break;
    case eMovingXNegative:
        min1Corner.x -= offset; max1Corner.x -= offset;
        break;
    case eMovingZPositive:
        min1Corner.z += offset; max1Corner.z += offset;
        break;
    case eMovingZNegative:
        min1Corner.z -= offset; max1Corner.z -= offset;
        break;
    default:
        break;
    }

    if ((max1Corner.x < min2Corner.x) || (min1Corner.x > max2Corner.x)) {
        return false;
    }
    if ((max1Corner.y < min2Corner.y) || (min1Corner.y > max2Corner.y)) {
        return false;
    }
    if ((max1Corner.z < min2Corner.z) || (min1Corner.z > max2Corner.z)) {
        return false;
    }
    return true;
}
```

Difference is that I add some offset to the position, and then test the collision.

V. Problems

I have a question:
For example, here we have two classes, A and B, A has a B's pointer (already allocated memory and initialized). Now some operations in B need some A's variables to take part in, so I pass these A's variables' addresses to B through B's constructor or the function arguments.
I think this method is not good, or is it a possible method?
If this is not a good method, any other ways to deal with this situation?

VI. Conclusion

This assignment makes me understand that we have to make fully preparation before getting start to write the code.