NCTU 3D Game Programming

Mini First-Person War Game

Zhang Zhexian; Student ID: 0545080; Assignment: 3; Email: zhangzhexian@outlook.com

[Yes, this is my own work ☺]

Introduction

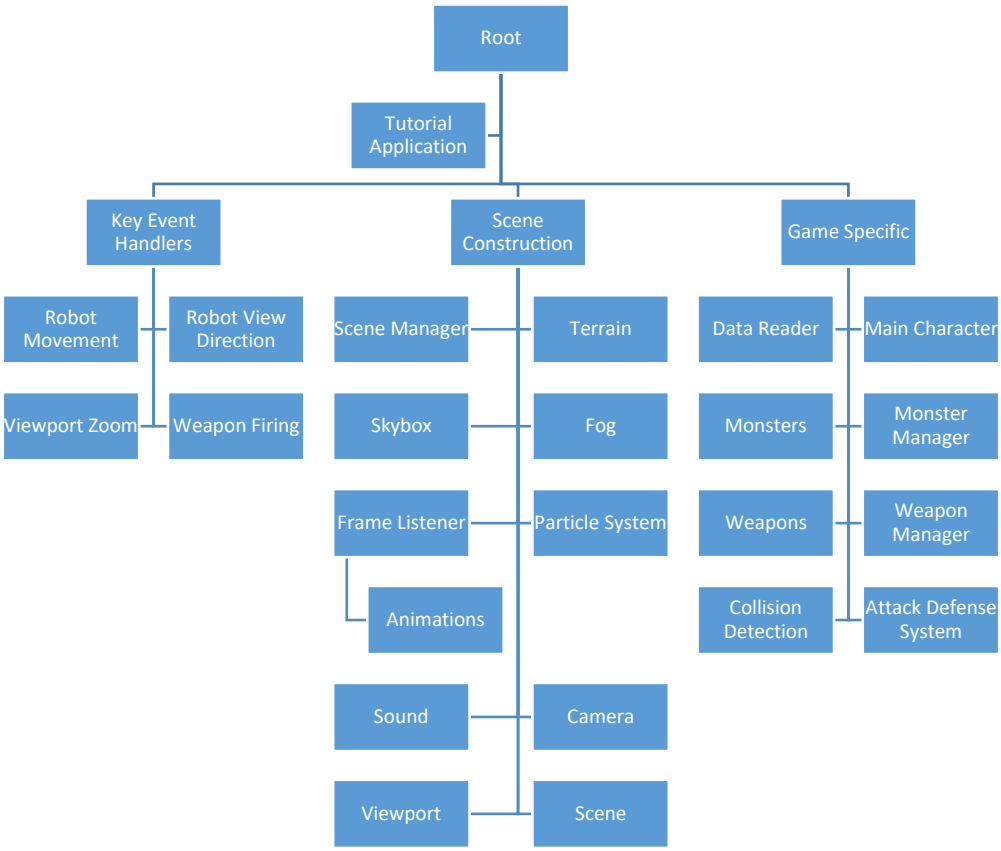
In the last assignment of the course, we are tasked to create another mini game. In this game, we will review and practise the basic concepts such as scene node and entity, movement, animation, and collision resolution. In addition, several new techniques are implemented in the game, including:

- File reader: customize parameters, used for game options
- Physics: trajectory, up and down, energy loss
- Dual viewport: mini map with zooming functions
- Multiple key assignment
- Cooldown time for game tools
- Time to live for game tools

Specifically, we are to create a world with **terrain** on water surface. The main **character** is **walking** from the edge of the terrain to the centre, where a **big sphere** is moving up and down on the spot. Filled on the terrain are many **monsters**, all facing towards the main character and moving towards it. They **move** in a special way that they move close to the main character when far away, but move away from it when the main character is near. The main character can fire its **weapon**, where bullets will be shot out. When the bullet drops onto the terrain, it will **bounce** off; when it collides with the big sphere, a **sound** is played to indicate that the target has been hit.

(word count: 210)

System Architecture



The system consists of three main parts:

1. Key Event Handlers: this is where the user's direct interaction with the game is defined. The user not only can exit the game by pressing "Esc", she may control the movement of the robot using keyboard, control view direction using mouse, and control weapon firing using specific key press. She may also control the zoom ratio of the mini map via key press.
2. Scene Construction: this part defines how the game world is constructed and how the basic animation is achieved. The view is constructed by setting up the scene, terrain, skybox, and fog etc. The camera, viewport, sound, and particle system is tweaked to achieve the various visual and audio effect in the game.
3. Game Specific functions: in this part, a few classes and functions are used to implement functionalities specific to game play. Data reader class is used to allow easy manipulation of game parameters via data reading and writing. Separating game roles into main character, monster, and weapon allows easy customization while re-using the basic building blocks of game objects. The collision detection and attack defence system creates independent game physics modules that may be reused.

(word count: 199)

Method

Visibility Mask and Flag

To achieve the effect of showing game object in one viewport but not the other, visibility mask maybe used with visibility flag set carefully on the game object to achieve selective display.

Set the visibility mask as : vp->setVisibilityMask(mask), where vp is the viewport pointer and mask is the mask of the viewport. This mask is useful in filtering the objects that you don't want them to be rendered on this viewport.

When the main character should be invisible in a certain viewport, the visibility flag of the main character need to be set so that it produces a 0 instead of 1 when “AND” with the visibility mask.

File Input / Output

To read the data file, the readData function is called: DATA_READER::readData();

To set the pointers of the objects so that the other functions can use the objects:

```
bt_Init(mTrayMgr, mSceneMgrArr[0], mCameraArr[0]);
```

Monsters

An array of monsters is created (number of monsters created is the maximum number defined). Based on the number of monsters actually needed, only those monsters will be set visible, the others will remain invisible and not alive (they are already created though).

```
for (int i = 0; i < mNumMonsters; ++i)
{
    if (i<mCurMonstersNum) {
        mMonstersArr[i]->setVisible(true);
    }
}
```

Weapons

Similar to the array of monsters, a pool of weapons are created as well. The difference is that a list of used weapons is kept, so once a weapon is filed, it’s index will be marked in the list of used weapons, and the used weapons cannot be fired again. In the context of the game, it means only a certain number of weapons may be fired at most in the game.

```
for (int i = 0; i < mMaxNum; ++i) {
        if (mUsedWeaponsArr[i] == false) continue;
        WEAPON *g = mBulletsArr[i];
        g->update(evt);
}
```

There is cool down time (a timer kept in weapon manager class) that disable continuous firing.

```
if (mCoolDownTimeForFiring > mMaxCoolDownTime)
{
        mCoolDownTimeForFiring = mMaxCoolDownTime;
}
```

(word count: 326)

Discussion

Object-Oriented Programming (OOP)

The prime purpose of C++ programming was to add object orientation to the C programming language. We started to implement OOP that greatly helped our game development process in this assignment.

Class and Object

Several classes were created, each with its specific purpose and implementation.

- MAIN_CHAR
- MONSTER_MANAGER
- MONSTER
- WEAPON_MANAGER
- WEAPON

For classes such as MAIN_CHAR and the MANAGER classes, only one instance is generated per game, while for MONSTER and WEAPON, many instances (objects belonged to the same class) are created as copies of similar game objects.

Inheritance

All classes are inherited from the GAME_OBJ class, with default parameters and functions defined such as mVelocity, setVisible(bool flg) and update() that are universal across all derived classes.

Polymorphism

Even though all inherited classes share the same common functions, the exact implementation of these functions are different (overwritten with new implementation), or new functions may be created that is specific only to the derived class. For example, the setWalkForward() function is only implemented in MAIN_CHAR, not in MONSTER nor WEAPON.

Data Hiding and Encapsulation

Encapsulation allows greater data security and integrity. By putting variables in public class only when it is necessary, but put in protected or private section and accessed via getter and setter, allows data to be changed only in its defined class. This improves the debugging process as well when we are sure that the data cannot be modified outside the class.

Message Passing

Message passing may be achieved by calling public functions of one class from another class, or by passing a pointer as parameter when instantiating an object from class. The former is encouraged to use more than the latter, as adding extra parameters as shown in the latter case may unnecessarily complicate the class.

I/O file reader

To resolve the bugs in demo program, data reader is modified based on the data category:

For categories such as NUM_BULLETS that are numbers, we take the double number following the category identifier as the data; for categories such as MESH_NAME, we take the std::string following the category identifier as the data.

Game Physics

Big Sphere Up-and-Down Movement

In frameStarted function, the following are called once per frame:

```
mToggle += evt.timeSinceLastFrame; //time used to toggle sphere up and down
Ogre::Real yPosition = sin(mToggle); //sine function for smooth movement
mSN_Sphere->translate(Vector3(0, yPosition, 0)); //translate the sphere
```

Collision Detection

Collision is detected by first compute the positions of A and B as p0 and p1 (both are Vector3). The slop connecting the two points, are computed as Vector3 mm = p0 - p1;

Get its length Real d = mm.length();

Compare d with the sum of radius of A and B to decide if the two objects have collided:

```
if (d < 50) {
        // handle collision
}
```

Trajectory Motion

The basic equation is newPosition = iniialPosition + v0 * deltaT + a * deltaT * deltaT,

where v0 is the velocity when weapon is fired, deltaT is the time since firing, and a is acceleration (the result of air resistance and gravity), net acceleration should be pointing downwards).

Implementation:

```
Vector3 position = this->getPosition();
Vector3 velocity = this->mVelocity*7;
Vector3 a = Vector3(0,-1,0);
velocity.y = -this->mVelocity.y*0.9;
position = (this->getPosition())+velocity*timeSinceBulletFiring+a*timeSinceBulletFiring*timeSinceBulletFiring;
if (projectScenePointOntoTerrain_PosDirection(position))
{
        velocity = velocity * 0.8;
        position = (this->getPosition())+Vector3(velocity.x, -velocity.y, velocity.z);
}
```

Game Play Implementations

In game play, several parameters are implemented such as m*aximum number of bullets* that affects the weapons to be used at most*; cool down time for firing* paused firing until certain time is passed*; life time for fired bullets* will hide the fired bullets instead of let them cluster in the view.

Multiple Event Listeners

For keyboard events, both pressed and released states need to be implemented (with action function and unset function), so that the action will not keep firing in a loop once a key is pressed.

For a specific key handling, the setWalkBackward() function is implemented based on the setWalkForward(), with the only difference that is

mSceneNode->translate(-d) (in walking forward it is translate (d))

and

Vector3 e = -actualDirection*20; (in walking forward it is -e)

So the movement is exactly the revers as walking forward, both in magnitude and direction.

(word count: 689)

Conclusion

Looking back to the start of the course, I am grateful that I learnt so much over a term.

It is my first time trying C++, although I am still having troubles with pointers, at least I am able to independently write complicated programs in assignments and quizzes. I love the basic game design theory taught in the course, and that we can implement the knowledge almost real time in projects and assignments. I am thankful that Ogre platform is chosen as game engine, instead of unity where a lot of implementation details are hidden (good for user experience, but bad for understanding development details). Doing the path finding quiz is my first time implementing an algorithm too, and I enjoy the sense of achievement.

One more quiz to go before presenting the final project, wish I can learn more and improve constantly.

Thank you very much my professor, TA, and classmates, for making the experience so enjoyable!

(word count: 158)