CS 207 Programming Methodology and Abstraction

Course Project Report – TankCraft

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Mengmeng Xu, Department of EE, 157223

Yang Liu, Department of AMCS, 158217

30. Nov. 2017

Table of Contents

Table of Figures

1. Game Description

TankCraft is a player vs player tank battle game, where each player manipulates one tank, control movement, rotation and fire action of the tank. The goal is to defeat the other player's tank. Each tank has 4 hit points (HP). An effective hit will cause the other tank lose 1 HP. The game ends when either of the tank's HP reduces to 0.

2. Class Description

a. Scenenodes

We use a tree data structure to store elements in the game. Each element is called a SceneNode. SceneNode class is derived from sf::Drawable, sf::Transformable and sf::Noncopyable. It has function drawCurrent(), which draws itself in game world.

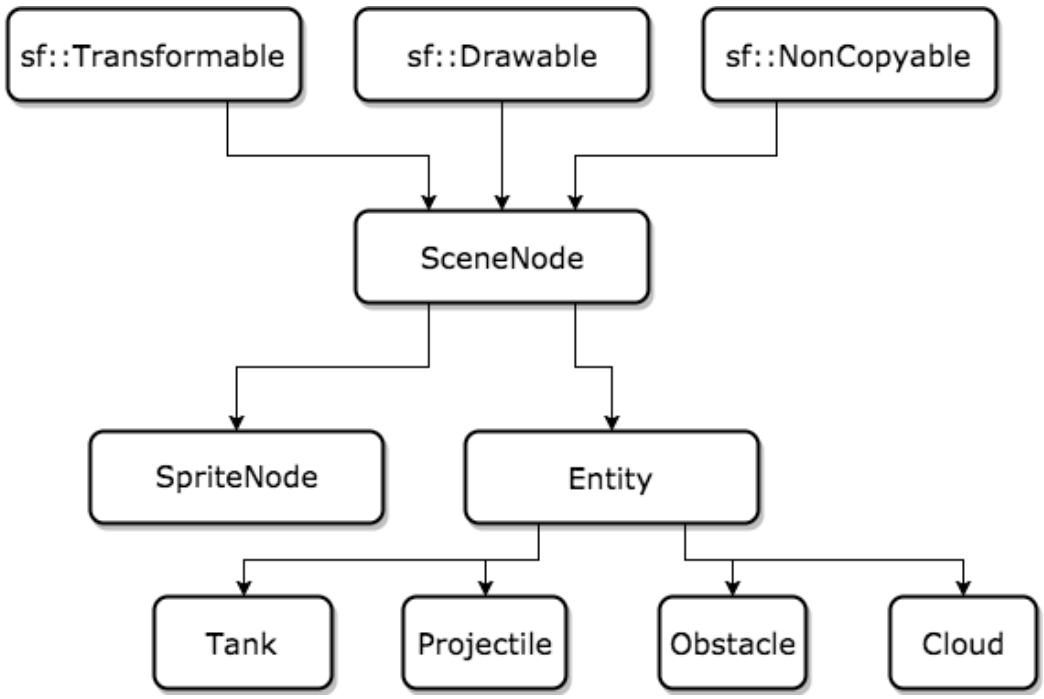

Figure 1 Hierarchical Structure of Classes that inherit Class SceneNode

Entity is inherited from SceneNode Class. It has functions setVelocity(), getVelocity(), setDirection(), getDirection(). Class Tank, Projectile, Obstacle and Cloud all inherit Entity Class. Class Tank set tank's HP, fire cool down time, damage cool down time and some other attributes. When function Fire() is called, the tank fires bullet. Class Tank also has collision detection functions, such as ObstacleTest(), shotTest(). The other three classes are similar to Class Tank but each has its own unique functions.

By using scenenode tree structure, we are able to traverse all the scenenodes in game and then draws it on screen. Also, we only need to know the relative transformation of a node to its parent node to obtain the transformation of this node relative to the root node.

b. World

Class world manages and displays objects in game. mPlayerTank1 and mPlayerTank2 are two tanks. mWorldBound is the border of the map, tanks are not allowed to move out of the map. CommandQ deals with player's inputs. mWorldView controls the viewport position. In function updateView(), the position and size of viewport will update according to the center and positions of two tanks. mSceneGraph has three layers. The first layer is background, which is fixed and draw first on the screen . The second layer is called Air, which contains two tanks, bullets and obstacles. Objects are transformable in this layer and collision detection is done in this layer. The third layer is called Sky, which only contains cloud. No collision is in this layer.

If collision detection function returns a true value, the handleCollision() function will be called to deal with consequence of collision. If two tanks collide, exchange their velocity, then apply an attenuate factor indicating the energy loss. If tank1's bullet collides tank2, tank2's HP decrease 1 and the same for tank2's bullet and tank1. If tank collides obstacle, the tank will be bounced back with a smaller velocity and forced to move to the position of last frame. This circumvents the problem of tank enters obstacle. If bullet collides obstacle, the bullet will be removed from mSceneNode, thus no longer visible. Also the memory will be freed.

c. States

We have designed different screens in the game, such as menu screen, gaming screen and pause screen. Each of these can be represented by a state. In our program, ConfirmState, GameState, MenuState, PauseState, Player1WinState, Player2WinState, StateStack are the classes that pertain to the class switch action.

Class StateStack is a stack container that holds different states. It has private std::vector mStack to store current states in stack, a std::vector mPendingList stores actions(push, pop, clear) to the StateStack, a std::map mFactories which associates StatesID and all created states. IDs are stored in Namespace StatesID. The update() method updates the states in the vector mStack. The draw() method draw all active states from bottom to top.

All state class in the game are inherited from the class State. This class has function requestStackPush(), requestStackpop() and requestStackClear(), which request modification of the StateStack. Function update() and draw() as mentioned above to update and display the state.

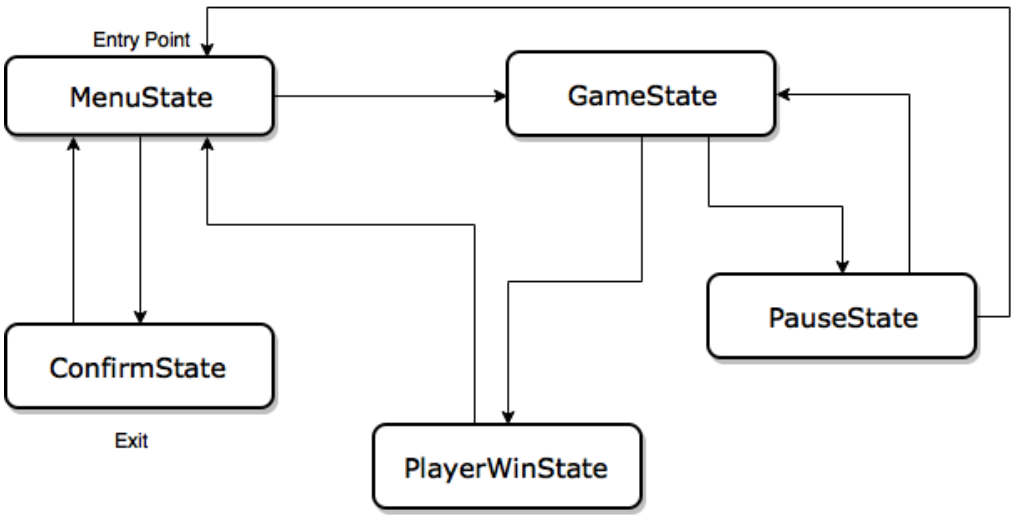

Figure 2 Switch among States

Menustate is pushed to statestack when the game begins. Menustate draws several buttons (via function draw()), which will be illustrared later. If we click the button play, then MenuState is popped and GameState is pushed into stack. The game world will be displayed until one player wins. Then the winnerState will be pushed and specific command will be needed to reload the game. By pressing esc to push PauseState into stack while game is running, the game will pause. From PauseState we can either resume game or go back to main menu by pressing corresponding key. The ConfirmState is added in case someone press the exit button accidently.

d. Button

Class Button is inherited from Class Component, which has Boolean attributes of isActive and isSelected. When active, the button will be highlighted. In our program, we select a brighter texture. If selected, the callback function of the button will be called.

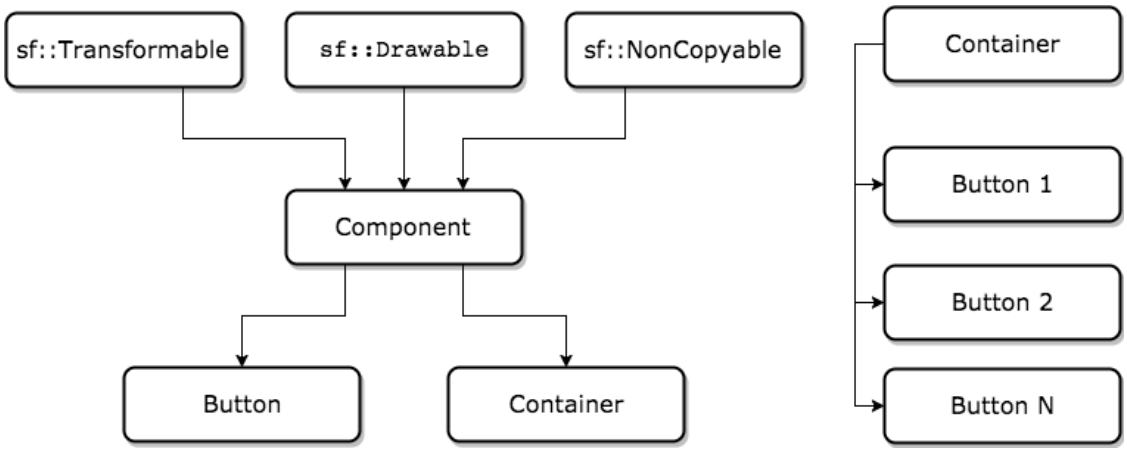

Figure 3 Class Component and Class Button

e. Handle Player Inputs

The keyboard key and game action is bound by std::map in class Player. Check if the pressed key binds to a command, push the command into command queue. Then the command is triggered. We set private member variables on mCategory on both

SceneNode Class and command class. When the program deal with a command, it will traverse all the scene node tree and compare its mCategory. Action will be processed only when the mCategory of command and scene node match.

3. Algorithms

a. Tank Velocity

When tank starts moving, the acceleration is not constant all the time. In the first beginning stage, the acceleration is constant. The velocity increases proportionally with the time the button is pressed. At the second stage, the acceleration decreases with the velocity increases. In the end, velocity reaches a limit.

b. Bullet velocity

The velocity of bullet that tank fires is associated with the movement of tank. The velocity of bullet comprises two parts, the intrinsic velocity of bullet and the velocity of tank. The bullet velocity is faster when tank moves forward than backward.

c. Momentum after collision

The calculation of velocity after collision is basically based on Newton's Second Law. Considering energy loss during collision, we also apply a attenuation factor to the velocity.

d. Friction

In our game setting, there exist friction between tank and ground, which means a moving tank without any operations will eventually stop.

e. Collision Detection

Use the SFML built-in function to get the axis-aligned bounding rectangle of entities. Then use SFML intersect function to test collision. If two rectangle overlaps, they are considered to collide.

Reference

https://www.sfml-dev.org/learn.php