Project 3 Report

Jingjing Nie 304567417

1. Descriptions of public member functions

**class Actor :**

//I use pure virtual function here because each derived class has its own implementation of doSomething. And I also want to prevent abstract class Actor being called and force the implementation of function doSomething

virtual void doSomething()=0;

//The below functions are not set to be virtual because they are used by all derived classes in similar ways

int getID() const; //return what object it is

int getPoints() const; //return object's life points

void setPoints(int points) ; //set object's points

void beingDamaged() ; //set object's status to "suffered damage"

bool isAlive() const ; //return true if the object is alive

void setDead() ; //set object's status to dead

bool offScreen(int x, int y) const; //return true if the object is off screen

void resetSuffer() ; //reset the object to "not suffered damage" after each tick

StudentWorld\* getStudentWorld() const ;

//get the StudentWorld so that functions in StudentWorld can be accessed

bool sufferDamage() const ;

//return true if the object has suffered damage

**class NanchenBlaster :**

//NanchenBlaster does not have derived classes, so I didn't put vitrtual functions

void increaseTorpedoes() ;

//increase the number of torpedoes NanchenBlaster has

int getTorpedoes() const ;

//return the number of torpedoes NanchenBlaster has

int getEnergyPoints() const ;

// return the energy points (cabbages) NanchenBlaster has

virtual void doSomething(); //NanchenBlaster's version of doSomething

**class Alien :**

//obtain information

int getMDir() const ;

//return the flight path direction of each Alien (direction marked by numbers)

int getFlightPlan() ; //return the flight path length of each Alien

double getSpeed() ; //return the flight speed of each Alien

//set variables

void setMDir(int d) ; //set the flight path direction of each Alien

void setFlightPlan(int Fplan) ; //set the flight path length of each Alien

void setSpeed(double speed) ; //return the flight speed of each Alien

void doSomething(); //Alien's version of doSomething

void move(); //let Alien move accordingly

void Dead (Actor\* a);

//let Alien act accordingly when it is dead. I did not use virtual because the aliens act similarly

**class Star :**

virtual void doSomething(); //star’s version of doSomething

**class Projectile :**

void doSomething(); //Projectile's version of doSomething

int getOwner() const; //return the owner of each projectile released

**class Explosion :**

virtual void doSomething(); //Explosion's version of doSomething

**class Goodies :**

void checkCollision(Actor\* a); //let goodies act accordinly if they collide with NanchenBlaster

virtual void doSomething(); //goodies' version of doSomething

**class StudentWorld :**

int init(); //initialize the Student World, and the objects in the world

int move();

//Start the game and let each object do something in each tick

void cleanUp(); //clean and free all objects in the world

void addObjects(Actor\* act); //add objects to the world

void removeDeadGameObjects(); //remove dead objects from the world

void updateDisplayText();

//update the displayed text based on game status

bool collide(Actor\* a1, Actor\* a2) const;

//return true if the two objects collide with each other

bool projectileCollide(Actor\* projectile, int Owner = 0);

//return true if the projectile collide with aliens or NanchenBlaster, or if NanchenBlaster collides with aliens

void alienDestroyed() {numAlienDestroyed++; numCurrAlien--;}

//update the number of aliens destroyed so far and the number of current aliens in the world

void alienOffScreen() {numCurrAlien--;}

//decrease the number of current aliens in the world if it flies off the screen

int getAlienDestroyed () const ; //return the number of aliens destroyed

bool needNewAlien ();

//return true if the formula suggests that new alien needs to be added

void addAlien(); //add aliens if needed

bool mayAttack(Actor\*) const;

//return true if the alien is close to NanchenBlaster so that it may attack NanchenBlaster

void incTorpedoes() ;

//increase NanchenBlaster's number of torpedoes when it picks up torpedo goodies

void attack(Actor\* a1, Actor \*a2); //perform attack actions

void incPoints(); //increase NanchenBlaster's life points when it picks up goodies

1. Functionalities that failed to finish

My code finished all functionalities and implementations, and there are no errors.

1. Other designs and assumptions

There is not an end for this game, so the player should be able to reach the next level as long as he or she successfully finishes the current level. The game only ends when the player loses all the lives, or presses quit.

1. How to test each class

**Class Star:**

When init() function is called, there should be 30 stars, with random sizes, displayed on the screen, and when move() function is called, random number of stars will appear. The stars will drift to the left.

**Class Alien:**

When the game starts, the number of aliens in the world should corresponds to the level number and the relevant equation. Each alien object then should follow random motions or its own special pathways as described in the spec, and when NanchenBlaster is to the right and is close to them, they should possibly shoot corresponding projectiles to attack NanchenBlaster. Also, when they collide with NanchenBlaster or are attacked by NanchenBlaster’s projectiles, they should loose appropriate life points and eventually explode. Aliens are dead after they fly off the screen from left, at which they should be removed and disappear.

**Class NanchenBlaster:**

NanchenBlaster should be able to move in correct directions when player enters different keys. And it should be able to shoot cabbages when it has more than zero energy points. When NanchenBlaster collides with aliens or projectiles shot by aliens, it should loose corresponding life points, and also destroys the alien. Also, when it picks up goodies, it should act accordingly, such as increasing the number of torpedoes it has, increasing the life points, or increasing life.

**Class Projectiles:**

Each projectile should have its own special movement patterns (such as rotation, moving direction, and speed) as specified in the spec. Also, each projectile has its designated owner. Then, when it collides with its enemies (which are determined by their owners), it the attacked object should have its life points decreased or destroyed.

**Class Goodies:**

Aliens, when they are dead, should randomly drop different types of goodies based on their own object types. The goodies will then follow random pathways and then be marked dead when flying off the screen. When the goodies collide with NanchenBlaster, they should be able to enable Nanchenblaster to receive the benefits, such as increasing the number of torpedoes it has, increasing the life points, or increasing life.

**Class Explosion:**

Explosions should occur when an alien has been destroyed by projectiles, or NanchenBlaster collides with an alien. Explosion objects should expand their sizes after they appear and then disappear after four ticks. Also, there should be sounds together with the occurrence of the explosions.

**Class StudentWorld:**

StudentWorld should be able to use the init() function to initialize all the objects in the game when the game first starts, and use cleanUp() function to clean up all the objects and free up the memory usages in the game when the player quits or the game ends. Then, during each tick, the move() function should be able to let each object to do something in its specific ways, introduce new objects and remove dead objects.

Also, StudentWorld should enable functions in Actor classes to successfully obtain or set necessary information since “Actor.h” has included a pointer variable pointing to a StudentWorld. It should also be able to check for collisions between different objects, and let each collision acts in appropriate ways.