JongWoo kim

503787749

report

actor.h

class Actor : public GraphObject

{

public:

Actor(StudentWorld\* world, int imageID, int startX, int startY)

:GraphObject(imageID, startX, startY)

{

s\_ptr = world;

N\_alive = true;

}

virtual ~Actor(){}; //later implementation

virtual void doSomething() = 0;

-pure virtual since actoris just base class for all derived class

virtual bool Alive();

-hold the information of an actor alived or died

virtual void Kill();

-set an actor to die

private:

StudentWorld\* s\_ptr;

bool N\_alive;

};

(Projectile)

class Shoting : public Actor

{

public:

Shoting(StudentWorld\* world, int imageID, int startX, int startY, bool playerFired)

:Actor(world, imageID, startX, startY)

{

Players = playerFired;

s\_ptr = world;

shoting = this;

}

virtual ~Shoting(){};

virtual void doSomething();

-main dosomething function for bullet and torpedo since they have same movement and features

bool shotplayer(Actor\* shoting);

-see whether player shotted by alien's projectile

bool collidAlien(Actor\* shoting);

-see whether alien is shot by player's projectile

bool fromPlayers() {return Players;}

-get information a projectile is from player or alien

private:

StudentWorld\* s\_ptr;

bool Players;

Actor\* shoting;

};

class Bullet : public Shoting

{

public:

Bullet(StudentWorld\* world, int imageID, int startX, int startY, bool playerFired)

:Shoting(world,IID\_BULLET, startX, startY,playerFired) {}

virtual ~Bullet(){};

//virtual void doSomething();

private:

};

class Torpedo : public Shoting

{

public:

Torpedo(StudentWorld\* world,int imageID ,int startX, int startY, bool playerFired)

:Shoting(world,IID\_TORPEDO, startX, startY,playerFired) {}

virtual ~Torpedo() {};

//virtual void doSomething();

private:

};

class Alien: public Actor

{

public:

Alien(StudentWorld\* world, int imageID, int startX, int startY,int worth, int hp)

:Actor(world, imageID, startX, startY), s\_ptr(world), h\_point(hp) , point(worth), hit(false) {}

virtual ~Alien(){} //later implementation

virtual void doSomething() =0;

-pure virtual since its just base class of aliens do nothing when called and this class is also abstract class

void shotbybullet(Alien\* a);

-alien shot by players's projectile decrement hp of alien

void shotbytorpedo(Alien\* a);

- alien shot by players's projectile decrement hp of alien

int getworth() {return point;}

-get worth point of alien

int getHp() {return h\_point;}

-get energy of alien

bool attacked() {return hit;}

-check alien is attacked by players

void sethit() {hit = false;}

-set hit initialize status

private:

int h\_point, point;

bool hit;

StudentWorld\* s\_ptr;

};

class NachlingBase : public Alien

{

public:

NachlingBase(StudentWorld\* world, int image,int round,int worth,int hp)

:Alien(world,image,rand() % VIEW\_WIDTH,VIEW\_HEIGHT-1,worth,hp), state(0), s\_ptr(world),n\_round(round)

,MDB(-1),HMD(-1),HMR(-1), tick(0), direction(-1) {}

virtual ~NachlingBase() {}

virtual void doSomething();

-main doSomething of nachilngs (wealthy and normal)

private:

StudentWorld\* s\_ptr;

int n\_round, state,tick;

int MDB,HMD,HMR,direction;

};

class Nachling : public NachlingBase

{

public:

Nachling(StudentWorld\* world,int round)

:NachlingBase(world,IID\_NACHLING,round,1000,5\*(0.9 + 0.1\*round)){}

private:

};

class WealthyNachling : public NachlingBase

{

public:

WealthyNachling(StudentWorld\* world,int round)

:NachlingBase(world, IID\_WEALTHY\_NACHLING, round, 1200, 8\*(0.9 + 0.1\*round)), malfun(false), tick(0), rannum(-1), s\_ptr(world) {}

virtual void doSomething();

-do specialized version of wealthy nachling and call nachlingbase's dosomething

private:

int tick, rannum;

bool malfun;

StudentWorld\* s\_ptr;

};

class SmallBots : public Alien

{

public:

SmallBots(StudentWorld\* world, int round)

:Alien(world,IID\_SMALLBOT,rand() % VIEW\_WIDTH,VIEW\_HEIGHT-1,1500,12\*(0.9 + 0.1\*round)), s\_ptr(world),tick(0), n\_round(round){}

virtual void doSomething();

-main movement of small bot

private:

StudentWorld\* s\_ptr;

int tick,n\_round;

};

class Stars : public Actor

{

public:

Stars(StudentWorld\* world) //constructor

:Actor(world, IID\_STAR, rand() % VIEW\_WIDTH, VIEW\_HEIGHT-1)

{}

virtual ~Stars(){}; //destructor

virtual void doSomething(); //later implementation

-star falling down and die at y = 0

private:

};

class ThePlayership : public Actor

{

public:

ThePlayership(StudentWorld\* world) // constructor

:Actor(world, IID\_PLAYER\_SHIP, VIEW\_WIDTH/2,1), N\_Key(0), hp(50), get(false), pre(0), re(false), shot(false)

,t\_count(0), n\_torpedo(0), s\_ptr(world) {}

virtual ~ThePlayership(){} // destructor

virtual void doSomething(); //later implementation

-getting keyboard information and make player ship do movement or fire projectile

void keyboard(bool t,int key);

-getting key board information from student world class

int tcount() {t\_count++; return t\_count;}

-use tcount for if there is no insert key for 2 tick, make previous key to initilize

void setpre() {pre = 0;}

-set previous key to initialize

int gethp() {return hp;}

-get hpoint of player ship

int getTorpedo() {return n\_torpedo;}

-get number of torpedo

void damage(Shoting\* s);

- decrement players hp

void collision();

-check player collid with alien

void fullenergy() {hp = 50;}

-restore energy when get goodie

void picktorpedo() {n\_torpedo = n\_torpedo+5;}

-+5 torpedo when get goodie

private:

StudentWorld\* s\_ptr;

bool get, re,shot;

int hp, n\_torpedo, ships;

int N\_Key,pre, t\_count;

};

class Goodie : public Actor

{

public:

Goodie(StudentWorld\* world, int imageID, int startX, int startY,int round)

:Actor(world,imageID,startX,startY), s\_ptr(world), tick(0) ,lifetime((100/round) +30),tickleft((100/round)+30) {}

virtual void doSomething();

-main movement and features of all goodie

virtual bool pick();

-check player pick the goodie

virtual void dogood() = 0;

-base class is abstract class

private:

int tick,lifetime,tickleft;

StudentWorld\* s\_ptr;

};

class FreeShipGoodie : public Goodie

{

public:

FreeShipGoodie(StudentWorld\* world, int startX, int startY,int round)

:Goodie(world,IID\_FREE\_SHIP\_GOODIE,startX,startY, round), s\_ptr(world) {}

virtual void dogood();

-make playership lives +1

private:

StudentWorld\* s\_ptr;

};

class EnergyGoodie : public Goodie

{

public:

EnergyGoodie(StudentWorld\* world, int startX, int startY,int round)

:Goodie(world,IID\_ENERGY\_GOODIE,startX,startY,round), s\_ptr(world){}

virtual void dogood();

-restore players energy full;

private:

StudentWorld\* s\_ptr;

};

class TorpedoGoodie : public Goodie

{

public:

TorpedoGoodie(StudentWorld\* world, int startX, int startY,int round)

:Goodie(world,IID\_TORPEDO\_GOODIE,startX,startY,round), s\_ptr(world){}

virtual void dogood();

-make +5 torpedo

private:

StudentWorld\* s\_ptr;

};

class StudentWorld : public GameWorld

{

public:

StudentWorld() : key(0), n\_alien(0), n\_round(1), e\_shoting(0), n\_des(0)//constructor

{}

virtual ~StudentWorld() //destructor

{

delete create;

vector<Actor\*>::iterator it ;

for (it = actors.begin();it != actors.end() ;)

{

delete \*it;

it = actors.erase(it);

}

}

void UpdateDisplay();

-call setvisible for all actors

void AddActor();

-add actors to space field

void MakedoSomething()

-make all actors to do something

bool RemoveDead();

-remove died actors

void UpdateStatus();

-update status line every tick

void addshoting(int ID,bool players,Actor\* ptr);

-add projectiles

void tickcounter();

void decailen() {n\_alien--;}

-decrease number of alien

void wealthydrop(Actor\* a);

-drop goodies from wealthydrop

void smallbotdrop(Actor\* a);

-drop goodies from small bots

void incdest() {n\_des++;}

-increase number of destroyed alien

ThePlayership\* getplayer();

int getRound() {return n\_round;}

-get round stage

vector<Actor\*> getActors() {return actors;}

-sent vector of Actor to actor.h to check information of actors

int getEshoting() {return e\_shoting;}

-get number of alien's shoting;

virtual void init()

{

create = new ThePlayership(this);

actors.push\_back(create);

}

virtual int move()

{

bool t = getKey(key);

create->keyboard(t,key);

//add new aliens or stars

AddActor();

//give each actor a chance to do something

MakedoSomething();

//remove newly dead actors after each tick

if (RemoveDead())

{

playSound(SOUND\_PLAYER\_DIE);

decLives();

n\_des = 0;

return GWSTATUS\_PLAYER\_DIED;

}

//update status

UpdateStatus();

//set visible

UpdateDisplay();

// This code is here merely to allow the game to build, run, and terminate after hitting enter a few times

//decLives();

return GWSTATUS\_CONTINUE\_GAME;

}

virtual void cleanUp()

{

if (!actors.empty())

{

for (unsigned int i=0;i < actors.size();)

{

Alien\* a = dynamic\_cast<Alien\*> (actors[actors.size()-1]);

if (a !=NULL)

{

n\_alien--;

}

Shoting\* s = dynamic\_cast<Shoting\*>(actors[actors.size()-1]);

if (s != NULL && !s->fromPlayers())

{

e\_shoting--;

}

delete actors[actors.size()-1];

actors.pop\_back();

}

}

}

private:

int n\_alien, key, n\_round, e\_shoting, n\_des;

ThePlayership\* create;

vector<Actor\*> actors;

};