Gajan Nagaraj

904937784

Project 3 Report

1. StudentWorld Class Public Functions:

StudentWorld(std::string assetDir); //constructor must be public

virtual ~StudentWorld(); //destructor must be public and virtual (derived from GameWorld)

virtual int init(); //game calls init to initialize every level. Must be virtual because it is pure virtual in GameWorld. Called by the game so it must be public.

virtual int move(); //game calls move every tick of the game to play out the game. Must be virtual because it is pure virtual in GameWorld. Called by the game so it must be public.

virtual void cleanUp(); //game calls cleanup when level or game is over in order to clear all dynamically allocated data. Must be virtual because it is pure virtual in GameWorld. Called by the game so it must be public.

void addItem(Actor\* x) { m\_v.push\_back(x); } //when called, appends a pointer to an Actor class to the Actor\* vector. This is effectively adding an element into the game. For example, this is called by Actors when they fire a projectile and a new projectile is introduced and therefore must be public.

NachenBlaster\* getNach() const { return m\_nach; } //return the pointer to the nacheblaster. Used for example by villain class to locate nachenblaster and fire projectile. Therefore must be public.

void collisionOccur(Actor\* p1); //check if an actor is colliding with another actor during a tick. This must be public because it is called by every actor during every tick.

Actor Class Public Functions:

Actor(int imageID, int startX, int startY, StudentWorld\* world, int damage = 0, int startDirection = 0, double size = 1.0, int depth = 0); //constructor must be public

virtual ~Actor() {} //destructor must be public and virtual (has classes derived from it)

virtual void doSomething() = 0; //Every actor has a doSomething that does something specific to them. Therefore, soSomething must be pure virtual as to redefine the behavior of this function specifically to every actor in the game.

bool getState() const { return m\_state; } //return a boolean stating whether actor is alive or dead. Called by StudentWorld so it should be positive.

void changeState() { m\_state = false; } //change actor to dead when it is killed in collision or offscreen. Called by StudentWorld so it should be public.

StudentWorld\* getWorld() const { return m\_world; } //return a pointer to the studentworld. Must be public because it is called by derived classes.

void offScreen(); //check if an object is offscreen, if so kill it. Called by derived classes so it must be public.

virtual bool collidable() const { return true; } //state if object can collide with other objects. This is true by default, but virtual because some actors such as stars and explosion cannot collide with other objects.

void collision(Actor\* p); //calls StudentWorld collisionOccur. Public because it is called by derived classes.

virtual bool isEvil() const { return false; } //state that identifies villians and villain projectiles. Virtual because this Boolean changes amongst all actors. Called by StudentWorld and derived classes so it must be public.

int getHealth() const { return m\_health; } //return health of actor. Must be public because it is called by StudentWorld and derived classes

void setHealth(int health) { m\_health = health; } //set health of actor. Must be public because it is called by StudentWorld

int getDamage() const { return m\_damage; } //get damage actor does to other actors. Must be public because it is called by StudentWorld

virtual bool damageable() const { return false; } //boolean that determines if actor can take damage. Virtual because not all actors can take damage. Different then collidable, because projectiles can collide, but cannot take damage.

virtual int type() const { return 0; } //return an int that identifies the type of actor. Virtual because all actors have different types. Default type is 0 for stars and explosions.

virtual int score() const { return 0; } //return the score associated with each actor. Default is zero because stars and explosions do not have scores, but other actors do. These scores vary so this function is virtual.

virtual void action() {} //if an actor performs a special action during its collision. This function is virtual because not all actors perform a special action during a collision.

Star Class Public Functions:

Star(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Star() {} //destructor must be public and virtual because it is a derived class

virtual void doSomething(); //doSomething is moving the stars to the left and changing their state if they go offscreen, must be virtual because pure virtual in base class

virtual bool collidable() { return false; } //cannot collide with stars. Must be public because it is redefining a function in base class

Explosion Class Public Functions:

Explosion(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Explosion() {}; // destructor must be public and virtual because it is a derived class

virtual void doSomething(); //do something is performing explosion graphic and changing the state of the explosion. Must be virtual because pure virtual in base class

virtual bool collidable() { return false; } //cannot collide with explosion. Must be public because it is redefining a function in base class

Craft Class Public Functions:

Craft(int imageID, int startX, int startY, StudentWorld\* world, int health, int damage, double size, int depth); //constructor must be public

virtual ~Craft() {} //destructor must be public and virtual because it is a derived class

virtual void doSomething() = 0; //do something is pure virtual because nachenblaster and villains have different do somethings

virtual bool damageable() const{ return true; } //craft can take damage. Must be public and virtual because it is redefining behavior from base class.

NachenBlaster Class Public Functions:

NachenBlaster(StudentWorld\* world); //constructor must be public

virtual ~NachenBlaster() {} //destructor must be public and virtual because it is a derived class

virtual void doSomething(); //do soemthing controls the nachenblaster from user input during a tick. Virtual and public because it is redefining base class function

virtual int type() const { return 1; } //return type that identifies nachenblaster. Virtual and public because it is redefining base class function.

int getCabbage() const { return m\_cabbage; } //get amount of Cabbages nachenblaster has. Public because it is used in StudentWorld.

int getTorpedo() const { return m\_torpedo; } //get amount of Torpedoes nachenblaster has. Public because it is used in StudentWorld.

void setTorpedo(int amt) { m\_torpedo = m\_torpedo + amt; } //set number of torpedoes nachenblaster has. Public because it is used by torpedo goodie class.

Villain Class Public Functions:

Villain(int imageID, int startX, int startY, StudentWorld\* world, int health, double travelSpeed, int flightPath, int damage); //constructor must be public

virtual ~Villain() {} //destructor must be virtual and public because this is a derived class

virtual bool isEvil() const { return true; } //villians are evil. This is public and virtual because it redefines a base class function

virtual int type() const { return 2; } //type that identifies villain class. This is public and virtual because it redefines a base class function

virtual int score() const { return 250; } //score when destroyed (changed for snagglegon). This is public and virtual because it redefines a base class function. Also it is redefined for the base class snagglegon which gives a bigger score.

void changeTravelDir(int travelDir) { m\_travelDir = travelDir; } //change the travel direction. This is public because it is called by derived classes.

virtual bool notSnagg() const { return true; } //bool determines if villain is a snag. This is public and virtual because it is redefined by derived classes.

virtual bool smoregon() const { return false; } //bool determines if villain is a smoregon. This is public and virtual because it is redefined by derived classes.

virtual void doSomething(); //defines the action performed by the villains in a given tick. Public and virtual because it is redefining the doSomething in the base Actor class.

virtual void action(); //perform villains special collision action for snagglegon and smoregon. This is public and virtual because it redefines the action class in Actor

Smallgon Class Public Functions:

Smallgon(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Smallgon() {}; //destructor must be virtual and public because this is a derived class

Smoregon Class Public Functions:

Smoregon(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Smoregon() {} //destructor must be virtual and public because this is a derived class

virtual bool smoregon() const { return true; } //bool that states this is smoregon class. Public and virtual because this redefines a base class function

Snagglegon Class Public Functions:

Snagglegon(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Snagglegon() {} //destructor must be virtual and public because this is a derived class

virtual int score() const { return 1000; } //score specific to snagglegon. Must be virtual and public because this redefines the score function in the actor base class.

virtual bool notSnagg() const { return false; } //bool that states this is snagglegon. This must be virtual and public because it redefines the notSnagg function in the villain base class

Projectile Class Public Functions:

Projectile(int imageID, int startX, int startY, StudentWorld\* world, bool side, int startDirection = 0, int damage = 0); //constructor must be public

virtual ~Projectile() {}; //destructor must be virtual and public because this is a derived class

virtual bool collidable() const { return false; } //projectiles cannot collide with objects, only the other way around. this prevents double collisions. This must be virtual and public because it is redefining the collidable function in actor base class

virtual int type() const { return 3; } //return the type of a projectile. This is public and virtual because it redefines a base class function

virtual bool isEvil() const { return m\_side; } //return whether a projectile is evil or not to prevent villains damaging villains. This is public and virtual because it redefines a base class function

virtual int typeOfProj() const = 0; //return type of the projectile being fired. This is public and pure virtual, because the type of the projectile changes for every projectile so this function is redefined in base classes

virtual void doSomething(); //perform the projectiles action during a tick. This is public and virtual because it redefines the doSomething function in the actor base class.

Cabbage Class Public Functions:

Cabbage(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Cabbage() {} //destructor must be virtual and public because this is a derived class

virtual int typeOfProj() const { return 0; } //return type associated to cabbage. This is virtual because it redefines the pure virtual function in the projectile base class

Turnip Class Public Functions:

Turnip(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Turnip() {}; //destructor must be virtual and public because this is a derived class

virtual int typeOfProj() const { return 1; } //return type associated with turnip. This is virtual because it redefines the pure virtual function in the projectile base class.

Torpedo Class Public Functions:

Torpedo(int startX, int startY, StudentWorld\* world, bool side); //constructor must be public

virtual ~Torpedo() {} //destructor must be virtual and public because this is a derived class

virtual int typeOfProj() const { return 2; } //return type associated with torpedo. This is virtual because it redefines the pure virtual function in the projectile base class.

Goodie Class Public Functions:

Goodie(int imageID, int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Goodie() {}; //destructor must be virtual and public because this is a derived class

virtual int type() const { return 4; } //return type of actor. This is public and virtual because it redefines a base class function

virtual void doSomething(); //perform action for the goodies during a tick. Public and virtual because it is redefining the doSomething in the base Actor class.

virtual bool collidable() const { return false; } //projectiles cannot collide with objects, only the other way around. this prevents double collisions. This must be virtual and public because it is redefining the collidable function in actor base class.

ExtraLife Class Public Functions:

ExtraLife(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~ExtraLife() {}; //destructor must be virtual and public because this is a derived class

virtual void action(); //perform special collision action for extra life. This function is virtual and public because not all actors perform a special action during a collision and this is a redefinition of base class function.

Repair Class Public Functions:

Repair(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~Repair() {}; //destructor must be virtual and public because this is a derived class

virtual void action(); //perform special collision action for repair. This function is virtual and public because not all actors perform a special action during a collision and this is a redefinition of base class function.

TorpedoGoodie Class Public Functions:

TorpedoGoodie(int startX, int startY, StudentWorld\* world); //constructor must be public

virtual ~TorpedoGoodie() {}; //destructor must be virtual and public because this is a derived class

virtual void action(); // perform special collision action for torpedo goodie. This function is virtual and public because not all actors perform a special action during a collision and this is a redefinition of base class function.

2. I did not fail to implement any functionality in the game. I believe there are no bugs in my classes.

3. My design decision was to divide all the actors into subclasses that is specific to common functionality. I believe I achieved this. This provides my project with less repetition and also better organization. I believe that the spec sheet covered all topics thoroughly and I was not confused and therefore did not have to assume anything.

4.