Project 3: Frack Man Report

1. Public Member functions

StudentWorld public member functions:

Note: All functions were made virtual functions in StudentWorld because it has no derived objects.

StudentWorld(std::string assetDir): GameWorld(assetDir){}

-constructs studentWorld

~StudentWorld();

-deletes frackman, dirt objects, all actors, and any coordinates

virtual int init();

-creates frackman, bouders, gold nuggets, and oil barrels. Sets the Coord\* arrays to nullptrs. returns different status messages based on if the level has been completed, the frackman is dead, or to continue the level.

virtual int move();

-updates the status bar, asks frackman and all other actors to do something, deletes any dead actors,

virtual void cleanUp();

//Mutators

virtual void dig(int x, int y);

-deletes dirt objects from the 2D array representing the oil field

-plays a sound

virtual void setFrackDead();

-set the frackman dead

virtual void incOilFound() {numOilFound++; }

virtual void foundWater();

-increases frackman’s water by 5

virtual void foundSonar();

-increases frackman’s sonar inventory by 1

virtual void foundGold();

-increases frackman’s gold inventory by 1

virtual void revealObjects(int radius);

-sets all objects that are within the radius of frackman to visible

virtual void protestToFrack();

-calls frackman’s annoy function

virtual void addBribe(int x, int y);

-add a temporary gold object to the map at position x ,y

virtual void addSquirt(int x, int y, int dir);

-adds a squirt of the map at position x, y with a given direction

virtual bool annoyAllProtestAt(int x, int y, int radius);

-calls the annoy function for all protesters within a certain radius

void evalLeaveArr();

-uses a queue to update a grid of coordinates that hold the direction a protester need to take to leave the oil field

void evalFrackArr(int fx, int fy);

-uses a queue to update a grid of coordinates (different from the previous grid) that holds the directions a hardcore protester needs to take to get to the frackman.

//Accessors

virtual bool isMoveInField(int x, int y) const;

-returns true if the given position is within bounds (between 0-60). If not, returns false.

virtual bool isMoveOnBould(int x, int y) const; //r<=3

-returns true if the given position is within a radius of 3 from a boulder

virtual bool isDirtUnderBould(int x, int y) const;

-returns true if there is dirt in one of the four squares below the given position specifying a boulder

virtual bool isBouldBelow(int x, int y) const;

-returns true if part of a boulder is below one of the four square below the given position specifying a boulder

virtual bool bHitsFrack(int x, int y) const; //r<=3

-returns true if the frackman is within a radius of 3 from the given position

virtual Actor\* findNearbyFrackMan(Actor\* a, int radius) const;

-if the frackman is within a radius of a, return a pointer to the frackman, otherwise return the nullptr

virtual Actor\* findNearbyProtest(Actor\* a, int radius) const;

-if a protester is within a radius of a, return a pointer to the frackman, otherwise return the nullptr

virtual void playSound(const int sound) const;

-plays a sound with the given soundID

virtual bool isDirtAt(int x, int y) const;

-returns true of there is dirt a the given position

virtual bool isFacingFrack(Actor\* a, int dir) const;

-returns true if the actor is within 4 units away from the frackman in the given direction

virtual bool canMoveTo(int ex, int ey) const;

-return true if there are no dirt objects or boulders in the 4x4 box specified by the given position

virtual int getLeaveDir(int x, int y) const;

-return an integer representing the direction a protester should go in order to leave the oil field

virtual int getFrackDir(int x, int y) const;

-return an integer representing the direction a hardcore protester should go in order to get to the frackman if he is nearby. This function operates slightly different from the last because it stops searching once it hits a certain amount of steps from frackman.

Actor Public Functions:

Actor(int imageID, int sx, int sy, Direction d, double size,

unsigned int depth, StudentWorld\* wldptr); //set visible to true

-constructs an actor. Automatically sets the actor to be visible

virtual ~Actor();

-destructs the actor

//accessors

virtual bool isAlive() const { return m\_isAlive; }

-returns if the object is alive or not. This is used for all actors in the same way, so it does not need to be pure virtual

virtual StudentWorld\* worldPtr() { return m\_world; }

-returns the pointer to the StudentWorld so it can access public functions of StudentWorld. This is used for all actors in the same way, so it does not need to be pure virtual

virtual bool isBoulder() const = 0;

-Returns if the actor is a boulder or not. A pure virtual function because all derived objects will return different values.

virtual bool isProtest() const = 0;

-Returns if the actor is a protester or not. A pure virtual function because all derived objects will return different values.

virtual bool canMove(Direction d);

-returns true if there are no dirt objects or boulders if an object tries to move one position in direction d. All actors use this in the same way, so it is a virtual function.

//mutators

virtual void doSomething() = 0;

-the object does something during each tick. Each derived object will define its own version.

virtual void setDead() { m\_isAlive = false; }

-sets the object as dead. All derived objects will use this function in the same way.

virtual void annoy(bool byBould) = 0;

-different objects are annoyed by a boulder in different ways. Derived objects will have their own version of this function.

virtual void beBribed() = 0;

-different objects will respond to being bribed in different ways, so they will have their own version of the function.

Squirt Public Functions: No pure virtual functions because it does not have any derived objects.

virtual bool isBoulder() const { return false; }

-returns false because it isn’t a boulder

virtual bool isProtest() const { return false; }

-returns false because it isn’t a protester

virtual void doSomething();

-moves four squares in its designated direction and annoys protesters

virtual void annoy(bool byBould) {}

-does nothing

virtual void beBribed() {}

-does nothing

Dirt Public Functions: No pure virtual functions because it does not have any derived objects

virtual void doSomething() {} //does nothing

virtual void annoy(bool byBould) {} //does nothing

//Accessors

virtual bool isBoulder() const { return false; }

virtual bool isProtest() const { return false; }

virtual void beBribed() {}

-does nothing

FrackMan Public Functions: No pure virtual functions because it does not have any derived objects

//Accessors

int getHitPts() const { return hitPts; }

int getNumWater() const { return numWater; }

int getNumSonar() const { return numSonar; }

int getNumGold() const { return numGold; }

virtual bool isBoulder() const { return false; }

virtual bool isProtest() const { return false; }

virtual void beBribed() {}

-does nothing (he can’t be bribed)

//Mutators

virtual void doSomething();

-immediately returns if the frackman is dead

-sets the frackman as dead if his hit points are below zero

-updates the array of coordinates that specify the direction hardcore protesters should take to follow frackman

-digs through any dirt that frackman is currently on

-moves the frackman

-drops bribes, fires squirts, activates sonars if the right key is pressed

-the next four functions change the private data members of frackman

virtual void annoy(bool byBould) { hitPts -= 2; }

virtual void incNumWater() { numWater += 5; }

virtual void incNumSonar() { numSonar++; }

virtual void incNumGold() { numGold++; }

Boulder Public Functions: No pure virtual functions because it does not have any derived objects

//Accessors

bState getBState() { return m\_state; }

-returns if the boulder is stable, waiting, or falling

virtual bool isBoulder() const { return true; }

virtual bool isProtest() const { return false; }

virtual void beBribed() {}

-does nothing (can’t be bribed)

//Mutators

virtual void doSomething();

-immediately returns if not alive

-if stable, checks if there is dirt below it

-if there is no dirt, changes its state to waiting

-if waiting, checks if enough ticks have passed

-if so, then changes its state to falling

-if falling

-if it hit dirt or another boulder -> sets itself to be dead

-if it his a protester or frackman -> annoys them

-moves down

virtual void annoy(bool byBould) {} //does nothing

void changeBState(bState newState) { m\_state = newState; }

ActivatingObject Public functions: This is an abstract object!

//Accessors

virtual bool isBoulder() const { return false; }

-all derived classes of activating objects are not boulders

virtual bool isProtest() const { return false; }

-all derived classes of activating objects are not protesters

virtual bool canFrackGet() const = 0;

-virtual function because all derived objects return different values. Some activating objects may or may not be pick-upable by frackman.

virtual int getSoundID() = 0;

-virtual function because all derived objects will return different soundIDs when they are picked up.

virtual int getScoreInc() = 0;

-virtual function because all derived objects will increase the score by different amounts when they are picked up.

virtual void beBribed() {}

-does nothing because all activating objects can’t be bribed

//Mutators

virtual void doSomething();

-all activating objects will follow the same general pattern when they are asked to do something. they will:

-return immediately if not alive

-if frackman picks up the object

-play a sound

-inform StudentWorld that it has been obtained (different for each object)

-if frackman is nearby

-make itself visible

-if a protester picks it up

-play a sound

-inform StudentWorld that it has been obtained(different for each object)

-cause some effect on the protester

-if the activating object is temporary

-do something temporary (different for each object)

virtual void annoy(bool byBould) {} // does nothing

virtual void frackFoundObj(int soundID, int scoreInc); //r<=3

-all activating objects will play a sound and increase the score by some amount when they are picked by by frackman.

virtual void frackNearObj() = 0; //r<=4

-pure virtual function because objects act differently when the frackman is near an object.

virtual void informObjObtained() = 0;

-pure virtual function because objects act differently when the object is obtained

virtual void tempObjDoesStuff() = 0;

-pure virtual function because not all activating objects are temporary. This result in different functions

Bribe Public Functions: No pure virtual functions because it does not have any derived objects

-the following three functions return values that allow its base class to do something helpful

virtual int getSoundID() { return SOUND\_PROTESTER\_FOUND\_GOLD; }

virtual int getScoreInc() { return 25; }

virtual bool canFrackGet() const { return false; }

virtual void frackNearObj() {}

-does nothing when frackman is near the object

virtual void informObjObtained() {};

-an inventory does not have to be updated when a protester picks up a bribe

virtual void tempObjDoesStuff();

-decreases the amt of ticks the bribe has left

Oil Public Functions: No pure virtual functions because it does not have any derived objects

-the following three functions return values that allow its base class to do something helpful

virtual int getSoundID() { return SOUND\_FOUND\_OIL; }

virtual int getScoreInc() { return 1000; }

virtual bool canFrackGet() const { return true; }

//Mutators

virtual void frackNearObj(); //r<=4

-sets the object to be visible

virtual void informObjObtained();

-tells student world that frackman has fond an oil object

virtual void tempObjDoesStuff() { return; }

-not a temporary object, so it does nothing

Gold Public Functions: No pure virtual functions because it does not have any derived objects

-the following three functions return values that allow its base class to do something helpful

virtual int getSoundID() { return SOUND\_GOT\_GOODIE; }

virtual int getScoreInc() { return 10; }

virtual bool canFrackGet() const { return true; }

//Mutators

virtual void frackNearObj();

-sets the object to be visible

virtual void informObjObtained();

-tells studnetworld to update frackman’s inventory

virtual void tempObjDoesStuff() {}

-not a temporary object, so it does nothing

Sonar Public Functions: No pure virtual functions because it does not have any derived objects

-the following three functions return values that allow its base class to do something helpful

virtual int getSoundID() { return SOUND\_GOT\_GOODIE; }

virtual int getScoreInc() { return 75; }

virtual bool canFrackGet() const { return true; }

//Mutators

virtual void frackNearObj() {} //does nothing (not a temporary object)

virtual void informObjObtained();

-tells studentworld to update frackman’s inventory

virtual void tempObjDoesStuff();

-decreases the number of ticks it has left

Water Public Functions: No pure virtual functions because it does not have any derived objects

-the following three functions return values that allow its base class to do something helpful

virtual int getSoundID() { return SOUND\_GOT\_GOODIE; }

virtual int getScoreInc() { return 100; }

virtual bool canFrackGet() const { return true; }

//Mutators

virtual void frackNearObj() {} //does nothing

virtual void informObjObtained();

-tells studentworld to update frackman’s inventory

virtual void tempObjDoesStuff();

-decreases the number of ticks it has left

Protester Public Functions:

//accessors

virtual bool isBoulder() const { return false; }

-all derived objects are not boulders

virtual bool isProtest() const { return true; }

-all derived objects are not protesters

virtual int squirtAnnoyScore() const = 0;

-each protester adds a different amt to the score if completely annoyed by a squirt

//mutators

virtual void doSomething();

-each protester follows a similar procedure when asked to do something:

-immediately returns if not alive

-rests if it is on a resting tick

-if leaving

-move in the direction calculated by studentWorld’s leaveArray of pointers to coordinates

-if the protester is within 4 units of frackman, annoy frackman

-if a hardcore protester can sense frackman’s cell phone signal

-move in direction of frackman. The direction is designated by StudentWorld’s frackArray of pointers to coordinates

-if the protester is in horizontal or vertical sight

-move towards the frackman

-move in a randomly generated driection

virtual void annoy(bool byBould);

-all protesters are annoyed similarly

virtual void doHardcoreDetect() = 0;

-regular protesters don’t do this step, so this function is pure virtual

virtual void decHitPts(int howMuch) { hitPts -= howMuch; }

virtual void beBribed() = 0;

-pure virtual because regular and hardcore protesters act differently when bribed

virtual void setLeave() { m\_isLeaving = true; }

virtual void setStare(int howLong) { restingTicks = howLong; }

virtual void attemptMove();

-all protesters attempt to make a move in some direction the same way

RegP Public Functions: No pure virtual functions because it does not have any derived objects

virtual void doHardcoreDetect() {} //does nothing

virtual int squirtAnnoyScore() const { return 100; }

virtual void beBribed();

-plays a sound and incrases the score. protester transitions to leaving state

HardcoreP Public Functions: No pure virtual functions because it does not have any derived objects

virtual void doHardcoreDetect();

-move in direction of frackman. The direction is designated by StudentWorld’s frackArray of pointers to coordinates

virtual int squirtAnnoyScore() const { return 250; }

virtual void beBribed();

-stops moving for a certain amount of ticks

1. Functionality and Known Bugs

To my knowledge, there is no functionality that I failed to complete. However, when a squirt is fired at a protester that is within 4 steps from FrackMan, the squirt graphics do not show up.

I also would have liked to decrease the number of times I update the 2D arrays holding the directions the protesters should move towards the exit or towards FrackMan, but I didn’t have the time to. It still works, but it updates more times than I would have liked.

1. Design Decisions

The spec says that water should not show up where any dirt is, but it does not say that it cannot show up in the same position as a boulder. My program has water pools showing up in the same position as boulders. (FrackMan can still technically obtain water pools in the same position as boulders by letting the boulders fall and then picking up the water.)

1. Tests for Each class

Here is how I implemented each class:

Actor

FrackMan

Dirt

Boulder

Activating Object

Oil Barrel

GoldNugget

Bribe (a temporary object with the same ID as a gold nugget)

Sonar Kit

Water Pool

Protester

Regular

Hardcore

Squirt

StudentWorld Class:

In general, I developed this class incrementally throughout the project to gradually incorporate the implementations of each actor. After I had implemented each actor, I incorporated it into the game with StudentWorld. Since it is difficult to tell whether or not some actors were actually added to the game because they start out as invisible, I often tested if StudentWorld was doing the correct thing by outputting some message once an actor was added or removed from the game.

I tested the path-finding data structure implemented in StudentWorld by first implementing a mock-program by modifying the mazequeue problem done for homework. Once I figured out how to find the shortest path using a simplified maze, I wrote a similar function for StudentWorld. I tested this by killing the protesters (by squirting and with boulders) and verifying if they did leave on the shortest path. For hardcore protesters, I output a message if they were in range from FrackMan and watched them find the shortest path to FrackMan.

FrackMan Class:

I also developed this class incrementally throughout the project because it interacts with every other actor created. I first implemented dirt objects and FrackMan’s dig function and tested if the dirt objects were properly removed by outputting a message. I then implemented the boulders and tested if FrackMan was properly set dead when boulders fell on him by outputting a message when he was removed from the game.

I then tested FrackMan’s interaction with ActivatingObjects by outputting a message when he picked them up, or made them visible. I also output messages once a squirt was created by FrackMan.

Dirt Objects:

Since Dirt Objects don’t really do anything besides get deleted, there wasn’t really much to test. I could visually see if they were deleted properly by playing the game, but I also outputted a message when they were deleted.

Boulders:

I tested this by outputting a message every time a boulder’s state changed, and once it was deleted. I compared the outputted messages with the boulder’s actual behavior, and tested if it was properly deleted in different situations (falling on dirt, falling on a boulder, or falling to the bottom of the field). Since level 0 doesn’t have many boulders, I modified StudentWorld to initialize many boulders just to test them.

Activating Objects:

Since oil barrels, gold nuggets, sonar kits, and water pools all have similar functions, I made them to be derived objects of another class I called ActivatingObjects. I also chose to make another class for a temporary gold nugget called bribe. Because of their similar behavior, I tested all of these objects in the same way. I output a message if an activating object was created, destroyed, made visible by a sonar or nearby FrackMan, or picked up by the FrackMan or protester.

Hardcore and Regular Protester:

The protester class is an abstract base class from which regular and hardcore protesters are derived. I couldn’t directing test an abstract class, so I only implemented the regular protesters at first. I tested the regular protester’s random movement by outputting a message with what kind of action it was performing each tick (leaving, shouting, moving in direct vertical or horizontal sight of FrackMan, making a perpendicular turn, or moving in a legal random direction). I compared these outputted messages with what the protester was actually doing and just watched the protesters move for a long time. I further tested the protester’s leaving function by the method described in StudentWorld.

Hardcore protesters are behave exactly the same as regular protesters, but they also can sense the FrackMan if they are a certain amount of legal steps away. I tested this behavior by outputting a message once the hardcore protester could sense FrackMan and watched the hardcore protester chase FrackMan.

Squirts:

I first tested if these squirt objects were created in the right direction relative to the FrackMan by shooting squirts in every direction. Then, I tested if squirts where correctly deleted once they hit dirt, boulders, or any protesters by outputting a message.