a. A high-level description of each of your public member functions in each of your classes, and why you chose to define each member function in its host class; also explain why (or why not) you decided to make each function virtual or pure virtual.

StudentWorld:

StudentWorld::StudentWorld();

*   Called when the StudentWorld is constructed
*   Initializes the member counter variable to 0 (for adding insects)

~StudentWorld();

 Deletes all GameObject pointers in the vector to prevent memory leak; clears vector

virtual void init();

*   Initializes a new round of the game
*   Creates the player and the specified number of Protesters in random positions without overlap
*   Store all those GameObject pointers into the vector
*   Made virtual just in case a different kind of world is to be designed later

 virtual int move();

*   Tells all current objects in the world that are still alive to do something

  If any object died during that last change, delete them from the vector

*   Return an integer indicating whether the game should continue

  Made virtual just in case a different kind of world is to be designed later  virtual void cleanUp();

*   Deletes all GameObject pointers in the vector to prevent memory leak; clears vector
*   Made virtual just in case a different kind of world is to be designed later  bool poisonMushroomThere(int x, int y);

void setText();

. Set the information on top of the screen.

void removeDirt();

. This is for the Frackman to remove the dirt when it has dirt before it.

bool isThereDirt(int x, int y);

. To check if there is any dirt supporting the boulders.

Actor\* findActor(int x, int y, int IID);

. Find the desire actor in the game and return that object.

bool compareActor(Actor\* actor, int type);

. Use dynamic casting to check if there is the target I am looking for and return true if

there is and return false otherwise.

void RemoveDeadActors();

. Remove the dead object from the container and the game.

void displayActor();

. This function is for displaying the Gold and the Barrels when Frackman is within the

4.0 radius from them

bool checkRadius(int x, int y);

. To check the radius of a specific object to see if it is near Frackman

player\* getPlayer();

. Return the pointer of the player to other classes to perform some functionalities/

void deBarrel();

. Decrement the number of Barrels when Frackman find them

void setBarrel(int b){barrel = b;}

. Set the number of Barrels at the beginning of the game.

void addGoodies();

. Add Sonar Kits at the left upper corner and Waterpool randomly in the oil field if the

space is available

bool putWaterpool(int x, int y);

. To check if it is ok to put waterpool.

bool pickupAble(int x, int y);

. To check if the Gold and the Barrels within the radius that is pickup able by Frackman.

bool detected(int x, int y);

. When Z or z is pressed, reveal the Gold and Barrels that is within the radius of 12.0

units

void createGold(int x, int y);

. Add Gold to the oil field when TAB is pressed.

bool DirtBlocked(int x, int y, GraphObject::Direction d);

. To check if there is dirt before the actor, when the protesters are wandering and the

squirt is fired.

Actor\* RunIntoActor(int x, int y, int ID, GraphObject::Direction D);

. Return the pointer of the target actor that blocks the current actor’s information that is

passed in

bool BlockSightOfProtester(int x, int y, GraphObject::Direction D);

. To check if there is any dirt in sight of the protesters.

bool BouldersInsight(int x, int y, GraphObject::Direction d);

. To check if there is boulders in sight of the protesters.

void addProtester();

. Add the protesters to the field based on the time and the total number of protesters on

the field

bool IsThereBoulder(int x, int y);

. Use for adding the waterpool when checking not overlapping with boulders

void setCounter(int x){counter = x;}

. Set the number of ticks.

void setNumberToAdd(int x){numberToAdd = x;}

. Set the total number of protesters to add on the field.

void findPath(int x, int y);

. Use a queue base search to find the optimal path to the exit when protesters are fully

annoyed.

GraphObject::Direction exit(int x, int y);

. Return the direction stored in the maze which tells the protesters which direction to

pick when leaving the oil field.

void deNumberToAdd(){numberToAdd--;}

. Decrement the total number of protesters on the field;

bool cellPhone(int x, int y);

. This is for hard core protesters to find Frackman.

GraphObject::Direction findFrackman(int x, int y);

. Return the direction to tell the hard core protesters to find Frackman.

void findPath2(int x, int y);

. Find the optimal path to reach Frackman.

Actor:

Actor(StudentWorld \*p, int ID, int x, int y, Direction dir, double size, unsigned int depth);

. Base class that other class inherits from.

virtual void doSomething()=0;

. Pure virtual, since no GameObject will be created to move in such a way; derived

classes all do something differently

StudentWorld \*getWorld() const {return m\_world;}

. Fetch and return the pointer to the world it’s in; the world doesn’t change, so it’s

constant

int getHealth() const {return m\_health;}

. Return the life points of the actor.

virtual ~Actor(){};

. Base class destructor, which essentially has nothing to free

void annoyed(){m\_health -= 2;}

. Deduct life points of the actors when it is annyed.

void setAnnoyed(int x){annoy = x;}

. Setter for annoy function.

int getAnnoyed(){return annoy;}

. getter for annoy function.

void setHealth(int health){m\_health = health;}

. Set the life points of a specific actor.

bool isAlive() const {return isalive;}

. Return true if the actor is alive and false otherwise.

void setDead(){isalive = false;}

. Set the actor to die.

virtual void incrementTick(){m\_ticks++;}

. Increment the ticks to control the actors moving speed.

int getCurrentTicks(){return m\_ticks;}

. Return the current ticks at a specific time.

int getState(){return state;}

. Get the state that the actors are currently at.

void setState(int s){state = s;}

. Let the actors enter a desire state.

void clearTicks(){m\_ticks = 0;}

. This is to reset the ticks t make the actors to act properly.

Player:

player(StudentWorld \*p, int x, int y);

virtual void doSomething();

*   Get the user’s input, which might be to move the player, shoot, or do nothing

  If it is an arrow key, pass that into movePlayer function

*   If it is to shoot, create a squirt until it hits something or leaves bounds
*   Change “can shoot” boolean so that the player can only shoot every other tick

void fire();

. Create a squirt in front of the Frackman.

int getWater(){return water;}

int getSonar(){return sonar;}

int getGold(){return gold;}

void deWater(){water -= 1;}

void addWater(){water += 5;}

void addGold(){gold++;}

void deGold(){gold -= 1;}

void addSonar(){sonar += 2;}

void deSonar(){sonar -= 1;}

virtual ~player(){};

. Player destructor, which essentially has nothing to free

Protester:

Protester(StudentWorld \*p, int x, int y,int ID);

virtual void doSomething();

. This virtual function do nothing.

void setNumOfSquaresToMove(int n){numOfSquaresToMove = n;}

int getNumOfSquaresToMove(){return numOfSquaresToMove;}

void deNumOfSquaresToMove(){numOfSquaresToMove -= 1;}

. These there function is for the functionality of the number of squares that the Protesters

can walk.

virtual ~Protester(){};

bool canMoveTo();

. To let the protester not run out of bound.

void chooseDirection();

. If the protester run out of bound or run into an obstacle, randomly set it direction.

void wandering();

. This function is for the protester not seeing the Frackman and walking around in the oil

field

bool huntPlayer();

. This function is for the protesters seeing the Frackman without any things block the

protesters’ sight and move forward to the direction of the Frackman.

bool playerInsight();

. This function is to check the protesters if seeing the Frackman without any things block

his sight.

void exit();

. This is the function tells the Protesters to choose correct direction when leaving the field

Regular Protester:

RegularProtester(StudentWorld \*p, int x, int y);

virtual void doSomething();

. I separate Four states:

First, the protester just walking around the oil field

Second, the protesters see the Frackman and hunt him.

Third, this is the state after the protesters yell at Frackman and stop in a number of ticks

Fourth, when the protesters are attacked and stop

Fifth, the exit state when the life points of the protesters become zero.

virtual ~RegularProtester(){};

Hard Core Protester:

HardcoreProtester(StudentWorld \*p, int x, int y);

virtual void doSomething();

. I separate Four states:

First, the protester just walking around the oil field

Second, the protesters see the Frackman and hunt him.

Third, this is the state after the protesters yell at Frackman and stop in a number of ticks

Fourth, when the protesters are attacked and stop

Fifth, the exit state when the life points of the protesters become zero.

Sixth, looking for Frackman when Frackman is within 16 units.

void searchFrackman();

. Search for Frackman when detecting Frackman

virtual ~HardcoreProtester(){};

Dirt:

Dirt(StudentWorld \*p, int x, int y);

virtual void doSomething();

. The dirt doesn’t need to do anything.

virtual ~Dirt(){};

Barrel:

Barrel(StudentWorld \*p, int x, int y);

virtual void doSomething();

. Check to see the Frackman walk nearby and set it visible. If it is pickup able by the

Frackman then set it to die.

virtual ~Barrel(){};

Boulders

boulders(StudentWorld \*p, int x, int y);

virtual void doSomething();

. There are two states for the boulders

First, stable state and check if there are dirt supporting him.

Second, falling state and check if it run out of the boundary and dirt, then set I to die.

Or run in the Actors and set that actor to die.

virtual ~boulders(){};

Gold

Gold(StudentWorld \*p, int x, int y);

virtual void doSomething();

. There are two states for the Gold:

First, check to see the Frackman nearby it and set I to visible. And then, check if it is

pickup able by the Frackman.

Second, when the Frackman drops it on the field and survive in a number of ticks

virtual ~Gold(){};

Squirt

Squirt(StudentWorld \*p, int x, int y, Direction dir);

virtual void doSomething();

. First, check if in its current direction has dirt or boulders if yes set it to die.

Second, if there is protesters within it travel distance and it his them successfully,

then annoy the protesters.

virtual ~Squirt(){};

Sonar:

Sonar(StudentWorld \*p, int x, int y);

virtual void doSomething();

. Pop up in the upper left corner of the field and check it is pickup able by Frackman.

virtual ~Sonar(){};

WaterRefills:

WaterRefills(StudentWorld \*p, int x, int y);

virtual void doSomething();

. Pop up randomly in the field that is not blocked by any thing. And check it is pickup

able by Frackman.

virtual ~WaterRefills(){};

**b. A list of all functionality that you failed to finish as well as known bugs in your classes.**

To my knowledge, I have finished all functionality that is listed in the project specifications. I originally did have some bugs with how the Protesters choose the optimal path when leaving the oil field. And I have hard time on dealing with the boulders when it is in falling state.

**c. A list of other design decisions and assumptions you made, e.g.: It was ambiguous what to do in situation X, and this is what I decided to do.**

The protester class is the most difficult part in the whole project 3. In the project specifications, the protester must choose to rest every other tick as well. I decided to initialize the ticks counter in such a way the protester moves in several ticks passes. Also, when I cannot come up with a good way to write a exit function for the protesters when they are fully annoyed and leaving the field. I ask for help an decide to use a two dimensional array which can store the direction of each grid to tell the protesters to choose the optimal path to exit.

**d. A description of how you tested each of your classes.**

Player

In order to test the player, I spawned the player by itself in the world to test its boundaries. Then I added dirt to make sure that the player could remove the dirt in a proper way. Lastly, I added the boulder in to the field and make sure that the player cannot run into the boulders.

Dirt

The dirt class essentially does nothing except display them self in the screen.

Regular Protester

Basically, first I just test the protester can show up on the game by just create them. And then, I add the moveTo function to test the protesters can walk on in the oil field. And then adding another feature that the protesters cannot walking into the dirt and boulders. After finishing all this thing. I try to add codes to test the Protesters can yell at the Frackman and pick up the gold.

Finally, I add the path searching function to test if the protesters can find the optimal path to leave the oil field when they are fully annoyed.

Hardcore Protester

Basically, I just modify the code I put in the Regular Protesters. And add one more state to it which is searching the Frackman when he is within 16 units.

Boulder

 First, I try to add the boulders in the oil field in the correct place without overlapping any other Goodies in the oil field. Then, I add the features that it will fall after a short period of time when there is no dirt support underneath. Then, I add codes to check if it will disappear when it run into the dirt or another boulder or run out of the boundary. Then, I try to add another features that it will kill the actors when they run into the falling range.

Squirt

I first just add one feature which is if the squirt can move in the direction of Frackman. Then, I add another feature that if it will disappear after it runs into the dirt or boulders or run out of the boundary. Then, I add another features which is if can stop the protesters and successfully annoyed then.

Gold

I just add them to the field first and see if they overlap other Goodies on the field. Then, I add another feature which is that if it will appear when the Frackman is nearby it. Then test it if it is pickup able by the Frackman. Lastly, I test if it can be added to field after it is picked up by the Frackman; also test it can survive a number of ticks when it is added. Lastly, I test if can be picked up by the protesters.

Barrel

First, just add them to the oil field and test it will set itself to be visible when the Frackman is nearby. Also it is pickup able by the Frackman.

WaterRefills

It is very simple, I just need to test if it can be added to the oil field with a specific probability. Also, I test if it can be picked up by Frackman when it is added to the oil field.

Sonar:

This is the same procedures as the WaterRefills class.