Report – Project 4

1. **A high level description of each of your public member functions**

**Graphical user interface, application

Description automatically generated**

**Actor.h/Actor.cpp**

**Actor class – The ultimate base class**

**Actor(StudentWorld \*world, int imageID, int startX, int startY, Direction dir, double size, unsigned int depth);**

Used to initialize the world the gameObject is in, its image, its starting location coordinates, its initial facing direction, it specified size, and its specified depth

**virtual ~Actor();**

Destructor that basically sets it to invisible; virtual as it is required for inheritance

**StudentWorld \*getWorld();**

Returns the StudentWorld class/object so we can use functions from StudentWorld class

**bool isAlive();**

Returns if the object is alive or not

**void die();**

This kills the object so it gets cleaned up by the cleanUp function

**virtual void doSomething() = 0;**

virtual as every inherited object will have a unique doSomething function

Abstract class as our actor base class doesn’t essentially do anything rather defines all the common functions that will be used by all objects

**virtual void annoyed(int health) {}**

virtual as every inherited object will be annoyed/annoy in some unique ways

Empty for actor as actors can’t be annoyed; only specific game actors can be annoyed

**Earth Class – Class for Earth that fill up the oil field**

Derived from actor class

**Earth(StudentWorld \*world, int startX, int startY)**

Initialize with world, image of earth, starting location, size of 0.25, depth of 3

**virtual ~Earth()**

Empty deconstructor

**Item class – Made for all objects that can be picked up**

Derived from actor class

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

Initialize its size to 1, depth to 2, and initialize tick count to 0

**disappear(int ticks);**

Items can disappear this function enables objects to disappear when a given a tick is specified when it needs to be disappeared

**Man Class – Base class for TunnelMan and protester**

Derived from actor class

**Man(StudentWorld \*world, int imageID, int startX, int startY, Direction dir, int health)**

Initialized with appropriate image of either TunnelMan or reg/hardcore protester, start location, size of 1, depth of 1, and set its private variable of m\_health to appropriate starting health

**int getHealth()**

Returns m\_health/current remaining health of the man object

**void decHealth(int points)**

Decrements the current health by the health points passed in

**virtual void moveDirection(Direction direction) = 0**

Virtual as derived classes can have different/redefined moveDirection function that determines how they move

Abstract as the man class itself doesn’t move/doesn’t make sense to have a moveDirection function.

**virtual void isAnnoyed(int health) = 0**

Virtual as derived classes can have different/redefined isAnnoyed function that determines how they react when they are annoyed by different objects in the game

Abstract as the man class itself cannot be annoyed/doesn’t make sense to have a isAnnoyed function.

**Goodie Class – Inherited from Items Class (For Sonar, Waterpool as they don’t have specific sounds like oil barrels and gold nuggets, and have the same tick count for when they can disappear)**

Derived from pick up

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

Passes in its image world class, image, and start location

**virtual void doSomething();**

Redefine doSomething where

1. It either just doesn’t do anything when the object is dead
2. If it is alive and within a radius of 3 from the TunnelMan it dies, plays the sound got goodie, and then accordingly updates TunnelMan’s item count
3. Disappears if it doesn’t get picked up (max(100, 300 – 10\*current\_level\_number) ticks)

**Boulder Class – Class for the boulders in the game**

Derived from actor class

**Boulder(StudentWorld \*world, int startX, int startY)**

Initialized with the world class, boulder image, starting locations, direction of facing down, size of 1, and depth of 1. It get initialized in a stable state and tick count at 0. Lastly, it removes any earth in the 4x4 it will occupy when it gets placed in the oil field

**virtual void doSomething();**

Refined doSomething for boulder where if there is earth underneath it, it will remain still and if there is no earth underneath it then it will start to fall when it reaches a tick count of 30 ticks and play falling rock sound. If there is earth or another boulder it kills the object. Otherwise it looks for protesters or TunnelMan to bonk/annoy.

**void annoy();**

If the boulder is within radius of 3 of any protesters or TunnelMan when it falls it decrements their health by a 100 (immediately kills). Uses StudentWorld functions (getRadius, getTunnelMan) to ease/lessen duplicated code.

**Squirt Class – Class for TunnelMan’s water gun**

Derived from actor class

**Squirt(StudentWorld \*world, int startX, int startY, Direction dir)**

Initialized with the world class, image of water sprut, starting locations, dir that can vary (TunnelMan’s facing location), size of 1, depth of 1, and initialized private m\_travel variable to 4.

**virtual void doSomething();**

Redefined doSomething for squirt where if its m\_travel reaches 0 or squirts a protester, using squirtProtester(), it gets killed. Otherwise, m\_travel gets decremented once. Then goes into a switch statement for up, down, left, and right. For every direction, if it squirts an earth, boulder, or exceeds the boundary, it dies . Otherwise, it can move one coordinate in whatever direction the squirt is facing.

**bool squirtProtester();**

If the squirt is in a radius of 3 from any protester, it decrements the protester’s health by 2 (using isAnnoyed function of protester). Since it squirted a protester successfully it returns true. Otherwise, it returns false. This returned value will be used in the doSomething to determine if the object will be killed/do something.

**Barrel Classs – Class for Oil Barrels**

Derived from items class

**Barrel(StudentWorld \*world, int startX, int startY)**

Initialized with the world class, image of barrel, starting location.

Its initial visible state is false.

**virtual void doSomething()**

Redefined doSomething for barrel. If it is not alive then just return. If it is not visible and player is withinradius of 4 then it sets visibility to true. If the player gets within a radius of 3 then it dies, plays sound of found oil, increase score by 1000, decrement barrel count (all done with StudentWorld class functions)

**Sonar Class – Class for Sonars**

Derived from goodie class

**Sonar(StudentWorld \*world, int startX, int startY)**

Initialized with the world class, image of sonar, starting location

**WaterPool Class – Class for WaterPools**

Derived from goodie class

**WaterPool(StudentWorld \*world, int startX, int startY)**

Initialized with the world class, image of Water Pool, starting location

**Nugget Class – Class for nuggets**

Derived from items class

**Nugget(StudentWorld \*world, int startX, int startY, bool dropped)**

Initialized with world, image of gold, starting location, and initial dropped state of either true or false (depends on if spawned by game or TunnelMan dropped it)

If the initial dropped state is true, the gold’s visibility state is true otherwise it should be visible.

**virtual void doSomething()**

Redefined doSomething. If it is not alive just return. If it is not visible and player is within radius of 4 from the gold, it becomes visible. If it is not dropped and player is within radius of three, it dies, plays sound of gotGoodie, increases score by 10, and add nuggets to TunnelMan’s inventory of items. If it is dropped and there is a protester in a radius of 3 (made sure that only one protester can pick up the gold by selecting the first protester from the vector of protesters return), gold nugget object is killed, plays protester found gold sound, increases score by 25, and sets protester to bribed. If dropped it has a lifetime tick count of a 100 therefore, by using the disappear function from items class make a dropped nugget disappear in 100 ticks.

**TunnelMan Class – Class for the player character**

Derived from Man Class

**TunnelMan(StudentWorld \*world)**

Initialized with world, image of player, initial location of x = 30, y = 60, facing direction of right, starting health of 10. For its inventory, it starts off with 5 units of water, 1 sonar charge, 0 gold nuggets.

**virtual ~TunnelMan()**

Empty destructor

**void doSomething()**

Redefine doSomething where if it is dead, it should simply not do anything.

Read in the user input:

-For escape: Play sound giveup and die

-For Space: If water in inventory is greater than 0 play squirt sound, decrement water by 1, and squirt water

-Up key: moveDirection upwards

-Down key: moveDirection downwards

-Left key: moveDirection left

-Right key: moveDirection right

-Z or z key: if sonar in inventory is greater than 0 decrement sonar by 1 and activate sonar (sonarActivate Function) and play sonar sound

-tab key: if gold in inventory is greater than 0 add nugget to the oil field but the dropped state should be true and decrement gold by one from inventory

**void moveDirection(Direction direction)**

Redefine moveDirection for TunnelMan; reads in direction the TunnelMan is trying to move or face

switch statement

if direction is up and if the TunnelMan is currently facing up & its y coordinates are less than 60 & boulder doesn’t exist above move in y direction by 1 & remove earth and play dig sound. Otherwise, the TunnelMan should now face up.

if direction is down and if the TunnelMan is currently facing down & its y coordinates are greater than 0 & boulder doesn’t exist below move in y direction by -1 & remove earth and play dig sound. Otherwise, the TunnelMan should now face down.

if direction is left and if the TunnelMan is currently facing left & its x coordinates are greater than 0 & boulder doesn’t exist to the left move in x direction by -1 & remove earth and play dig sound. Otherwise, the TunnelMan should now face left.

if direction is right and if the TunnelMan is currently facing right & its x coordinates are less than 60 & boulder doesn’t exist to the right move in x direction by 1 & remove earth and play dig sound. Otherwise, the TunnelMan should now face right.

if direction is none just break

**void Squirt()**

Get TunnelMan’s direction.

If case is up

* If there are no earth and boulders +4 coordinates in the y direction and doesn’t pass the boundary limits, create a new squirt object at the coordinate +4 above TunnelMan

If case is down

* If there are no earth and boulders -4 coordinates in the y direction and doesn’t pass the boundary limits, create a new squirt object at the coordinate -4 below TunnelMan

If case is left

* If there are no earth and boulders -4 coordinates in the x direction and doesn’t pass the boundary limits, create a new squirt object at the coordinate -4 to the left of TunnelMan

If case is right

* If there are no earth and boulders +4 coordinates in the x direction and doesn’t pass the boundary limits, create a new squirt object at the coordinate +4 to the right of TunnelMan

If direction is none just break

**void isAnnoyed(int health)**

Decrement health by integer passed in. If the health is less than 0 die and play player give up.

**int getWater()**

Returns amount of water TunnelMan has in his inventory

**int getSonar()**

Returns amount of sonar TunnelMan has in his inventory

**void addGoodieItems**

A function that can add different items to TunnelMan’s inventory.

If the image id is a sonar, increase score by 75 and add 2 sonars to TunnelMan’s inventory.

If the image id is a waterpool, increase score by 100 and add 5 water to TunnelMan’s inventory.

**void addNuggets()**

Adds one gold nugget to TunnelMan’s inventory

**int getGold()**

Returns the number of gold TunnelMan has in his inventory

**Protester Class – Class for both regular and hardcore protester class**

Derived from Man Class

**Protester(StudentWorld \*world, int imageID, int health)**

Initialized with world, image of either regular protester or hardcore protester, initial facing direction of left, appropriate health for either regular (5) or hardcore protester (20), leave state as false, last perpendicular count to 200, last yell count to 15, numSquaresToMoveInCurrentDIrection to random number between 8 and 60, and waiting tick count to max(0, 3 – current\_level\_number/4).

**void doSomething**

Redefined doSomething where if it is dead it shouldn’t do anything.

Otherwise, if there are ticks to wait decrement by 1 and return.

If the waiting time/tick is over for the protester reset tickwait with max(0, 3 – current\_level\_number/4). Increment last yell and last perpendicular turn made each tick.

If the protester is in the state of leave the oil field, when it eventually reaches the coordinate (60, 60) it dies, sets itself invisible, the world removes the protester, and returns.

Otherwise the regular protester must move one square closer to (60,60)

Therefore, it calls moveToExit function from StudentWorld that guides the protester to the shortest path to 60,60.

Otherwise, since it is not in the state of leave, it checks if any player is within a radius of 4 from itself and is facing the TunnelMan by using playerWithinRadius and isFacingTunnelMan function.

If the TunnelMan’s position meets these conditions, the protester yells at the TunnelMan if the tick since last yell is greater than 15. Since the TunnelMan got yelled at it annoyes the TunnelMan and decrements its health by 2. It resets the tick count for tick since last yell. Play sound protester yell and return.

ONLY FOR HARDCORE PROTESTER

If the protester’s image id is hardcore protester, it will calculate the value M where it is the legal horizontal or vertical moves that the hardcore protester is allowed to move to reach the TunnelMan. It receives the direction it needs to move to reach the TunnelMan (if it can be reached in M moves) from the function senseTunnelMan (in StudentWorld). Then it changes its current direction to face this direction and move one square in that direction.

If direction to TunnelMan is sensed/known, there is a clear path, and TunnelMan is more than 4 radius away, the protester sets its direction to TunnelMan. Then it takes a step toward TunnelMan.

At this point, the protester can’t directly see/sense the protester. It decrements numSquaresToMove by 1. If the numSquaresToMove is less than 0, it chooses a random direction until it gets a valid move/direction. If it finds a valid direction, the protester changes its direction to this direction then random generates a new numSquareToMove. Then it attempts to take one step in the currently facing direction.

If the protester is at an intersection where it allows for a perpendicular turn and hasn’t made a perpendicular turn in the last 200 resting ticks, it finds determines randomly which viable turn to choose, then selects a new value for numSquaresToMoveInCurrentDirection, resets lastPerpendicularTurn tick to 0.

If the protester is walking but cannot move/blocked from taking a step in the currently facing direction, it will set numSquaresToMoveInCurrentDirection to zero.

**void isAnnoyed(int amt)**

Redefined isAnnoyed.

If it is in a state of leave already just return as it can’t be further annoyed.

Otherwise, decrement the health point of amt.

If the health of the protester is greater than 0, it will be stunned and play protester annoyed sound.

If the health of the protester is less or equal to 0, leave state will be true and plays protester give up sound.

The tick for waiting will be set to 0 so it ensures it will do something on the very next tick.

If the amt of health decremented was 100, it means a boulder bonked the protester therefore 500 points will be added. Else if it was a regular protester that got annoyed and left, it increases point by a 100. Else, it means a hardcore protester got annoyed and left, it increase point by 250.

**int getTickWait()**

Returns m\_tickWait (the number of ticks that the protester is in a rest state for)

**bool getLeaveState()**

Returns m\_leave (true if leave state else false)

**void moveDirection(Direction dir)**

If direction passed in is up

* If it already facing up
  + if y coordinate of protester is at 60 set its direction down because we reached the boundary
  + else move up by one
* else set its direction to up

If direction passed in is down

* If it is already facing down
  + if y coordinate of protester is at 0 set its direction up because we reached the boundary
  + else move down by one
* else set its direction to down

If direction passed in is left

* If it is already facing left
  + if x coordinate of protester is at 0 set its direction right because we reached the boundary
  + else move left by one
* else set its direction to left

If direction passed in is right

* If it is already facing right
  + if x coordinate of protester is at 60 set its direction left because we reached the boundary
  + else move right by one
* else set its direction to right

If direction of none was passed in just return

**bool isFacingTunnelMan()**

Compares the x and y coordinate and accordingly and returns true if it is facing the direction of TunnelMan and returns false if it is not facing in the direction of TunnelMan.

**Direction dirToTunnelMan()**

Returns the direction to TunnelMan if they are in a straight line with each other

if the TunnelMans x coordinates and y coordinates match up to that of protester’s, it simply just returns the protesters direction.

If the protester’s x coordinate matches up with TunnelMan’s

* if protester is below TunnelMan return up
* else return down

If the protester’s y coordinate matches up with TunnelMan’s

* if protester is to the left of TunnelMan return right
* else return left

as a default statement return none

**bool is clearPath**

When the direction is up – if there are no boulders or earth (ie validMove) from protester’s y coordinate to TunnelMan’s y coordinate return true

When the direction is down – if there are no boulders or earth (ie validMove) from protester’s y coordinate to TunnelMan’s y coordinate return true

When the direction is left – if there are no boulders or earth (ie validMove) from protester’s x coordinate to TunnelMan’s x coordinate return true

When the direction is right – if there are no boulders or earth (ie validMove) from protester’s x coordinate to TunnelMan’s x coordinate return true

**Direction getRandomDir()**

Random number generator 1-4

1 = up

2 = down

3 = left

4 = right

**bool atIntersection()**

If get direction is left or right and there is a valid move either going up or down return true

If get direction is up or down and there is a valid move either going left or right return true

**void viableTurn()**

if direction is up or down

* if there isn’t a validMove to the left setDirection to the right
* else if there isn’t a validMove to the right setDirection to the left
* else it means both left and right are valid moves so randomly choose between the two

else it means direction is left or right

* if there isn’t a validMove up setDirection down
* else if there isn’t a validMove down setDirection up
* else it means both up and down are valid moves so randomly choose between the two

**void bribed()**

if the image id is protester -> goes into leave state

else the image is hardcore protester -> increases score by an additional 25 and basically gets stun/set tick for wait to max(50, 100 – current\_level\_number \* 10)

**Regular Protester**

**RegularProtester(StudentWorld \*world)**

Derived from protester class

Initialized with world, image of protester, and health to 5

**HardCore Protester**

**HardCoreProtester(StudentWorld \*world)**

Derived from protester class

Initialized with world, image of hardcore protester, and health to 20

**StudentWorld.h/.cpp**

**Private data members:**

protester count

barrel count

gold nugget count

sonar count

bool firstTick

tick count

2D array of Earth 64 by 64 all set to nullptr

TunnelMan pointer

Vector of actor pointers

2D array of maze 64 by 64

Struct of coordinate

**StudentWorld(std::string assetDir)**

Initializes GameWorld with assetDir

**~StudentWorld()**

Empty destructor

**int init()**

Initializes the whole game world

from row 0 to 60 and column 0 to 64 construct an earth object

For rows 30 to 33 inclusive, make them invisible

Set boulder count to min(current\_level\_number / 2 + 2, 9)

Set gold nugget count to max(5-current\_level\_number / 2, 2)

Set barrel count to min(2 + current\_level\_number, 21)

add boulder, nuggets, and barrels

add new TunnelMan object

return GWSTATUS\_CONTINUE\_GAME

**int move()**

Moves for each tick

Display the score status/inventory (textFormatter)

Iterate through the actor vector

If the actor is alive doSomething

If the actor is not alive decrease life by one and return GWASTATUS\_PLAYER\_DIED

If the barrel left in the oil field is 0 play sound finished level and return GWASTATUS\_FINISHED\_LEVEL

It then asks the player to do something also adds protesters and goodies

It then iterates again through the actor vectors to remove any actors that are dead

Returns GWASTATUS\_CONTINUE\_GAME

**void cleanUp()**

From rows 0 to 60 and cols 0 to 64, in the earth 2d array, if it is not a nullptr delete it and set the pointer in the 2d array to nullptr. We have finished deleting the whole earth 2d array.

Iterate through the vector array and delete each actor then clear the vectors of pointers.

Delete the TunnelMan.

**void textFormatter()**

Gets level, lives, health, water, gold, sonar, score.

Formats the text.

Use the setGameStatText to place the string text into the game.

**bool removeEarth(int x, int y)**

Get the row and col of the object.

Default removed is false.

From x and y going 4x4 to the right and up if it is not visible or is the tunnel skip

If there was any earth that was visible set it invisible and set removed as true

Return bool var removed

**bool earthUnderneath(int x, int y)**

If the earth is visible 1x4 down one from object’s coordinate return true else return false.

**bool earthExists (int x, int y)**

If the earth is visible 4x4 from the coordinate of the player return true

**bool boulderExists(int x, int y)**

Iterate through the m\_actors vector. If the image id is boulder check if any pixels of the boulder overlap with the object, we are comparing with. Check -3 to +3 for both the boulder and for the object at the coordinate standpoint.

**bool withinRadius** **(int x, int y, int x2, int y2, int radius)**

Use Euclidian/Pythagorean to see if it is within radius

**bool playerWithinRadius (Actor \*a, int radius)**

If the player is within radius of an actor return true otherwise false

**vector<Protester \*> allProtesterInRadius(Actor \*actor, int radius)**

Iterate through the actor vector, if the image id is protester or hardcore protester save it to a Protester pointer variable ptr by dynamic casting from Actor to Protester. If the protester being investigated has a waiting tick of more than 3 (max delay time for protester movement is 3 ticks) continue to investigate other actors. If the protester does have a waiting tick of less than 3 add it to the vector. After it adds all the appropriate protesters within the radius return the vector.

**void sonarActivate(int x, int y)**

Iterate through the actor vector, if the image id is barrel or gold and if it is within radius of 12 setVisible to true for those actor objects.

**bool objectsWithinRadius(int x, int y, int radius)**

Function used to widespread the objects throughout the oil field.

Iterate through the actor vector, if any object in the vector is within radius of a coordinate return true else return false

**bool validMove(int x, int y, GraphObject::Direction dir)**

switch statement

if direction is up

-if y is not 60, earth does not exist above, boulder does not exist above return true else false

if direction is down

-if y is not 0, earth does not exist below, boulder doesn’t exist below return true else false

if direction is left

-if x is not 0, earth does not exist to the left, boulder doesn’t exist to the left return true else false

if direction is right

-if x is not 60, earth does not exist to the right, boulder doesn’t exist to the right return true else false

if direction is none

-always return false

default return false

**void moveToExit(Protester \*Protester)**

First initialize the 2D maze array with all 0s.

Using BFS algorithm with queue, push coordinates.

Trying to go up, down, left, and right until it can reach coordinate of (60,60)

***The way how this all works:***

***Diagram, engineering drawing

Description automatically generated***

In the 2D maze array, it basically is a map to each pixel of the oilfield. We record in each element of the array, how far away from (60,60) it is.

Unlike the examples we did in lecture where we went from a protester’s position to the exit, we go the opposite way – from exit to protester.

When processing the maze, read the current distance to the exit from the integer matrix. Then determine all its valid neighbors that the protester could move to that also must have a distance 0 in the corresponding cell (means it’s unvisited). Write the current distance + 1 into those cells and add them to the queue. At the end, you can then find the shortest path in which the protester just moves to whatever coordinate/element of the 2D array maze has a distance that is 1 less than the distance the protester needs to move to reach the exit.

**GraphObject::Direction senseTunnelMan(Protester \*ptr, int M)**

Same method as the moveToExit but this time the we start from TunnelMan to the Hard Core protester.

For the second step, it is the same idea as the moveToExit, where it chooses the path where distance is -1 from that of the current distance. However, this time rather than moving the protester just return the direction in which if the hardcore protester takes it can move one step closer to TunnelMan.

**void addActor(Actor \*a)**

Simply adds actor to the m\_actors vector

**void addBoulder(int num)**

Adds boulder to the oil field

Initialize filled to 0.

While filled is less than the num of boulders we are adding in for the current level

Get a random col and row between x=0,y=20 and x=60,y=56, inclusive (so they have room to fall). If there are no objects within the radius of 6 and is within the boundary of the oil field, add/make a new boulder object in the randomly selected coordinate and increment filled by 1 as we added a new boulder.

**void addProtester()**

Initialize T to max(25, 200 – current\_level\_number)

Protester can be only added to the oil field after at least T ticks have passed since the last Protester of any type was added

Initialize P to min(15, 2 + current\_level\_number \* 1.5)

Target number P of Protesters that should be on the oil field in the given level of the game

if it is either a first tick OR tickCount since last protester was added is more than T && number of protester in the game is less than P

The probability of the new protester that will be added to the oil field is min(90, current\_level\_number \* 10 + 30) -> add hardcore protester else add regular protester

Now we need set firstTick to false and reset tickCount to 0 and increment protester count for the game by 1.

While we do the above, we keep ticking the clock by incrementing tick count by 1.

**void addGoodies()**

There is a 1 in G chance that a new waterpool or sonar kit will be added to the oil field during any particular tick.

Therefore we initialize G to int G = current\_level\_number \* 25 + 300

If the game landed on that 1 in G chance

* There is a 1/5 possibility that this addition of goodie will be a sonar
* Add a new sonar object to 0,60 coordinate in the game
* Else there is a 4/5 possibility that this addition of goodie will be a waterpool
* Randomize row and col from 0 to 60 for both until there is no earth and no objects within radius of 6
* add new waterpool object in the valid randomized coordinate

**void addNugBar(int num, char type)**

Initialize filled to 0

while filled is less than the num of nuggets of barrels we will be adding to the game

randomize coordinates so that it is within boundary of oil field

If there are no objects within radius of 6 of the coordinate and the coordinate is not part of the tunnel, we either add a barrel or a gold nugget to the game. Increment filled by 1 whenever there was a successful addition of barrel or gold to the game.

\*If we added a barrel, we increment the barrel count by one for barrels left in the game\*

**void decBarrels()**

Decrement number of barrels left in the game by 1

**TunnelMan \*getTunnelMan()**

Return the TunnelMan pointer

**void removeProtester()**

Decrement protester count by 1

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

There is no functionality that I have failed to finish as well as bugs that I know of.

1. **A list of other design decisions and assumptions you made**

Design choice was like what the professor suggested except that I further divided the goodie class the professor had as I saw similarities only between the sonar and water pool. The oil barrels and gold nuggets had specific functionalities that needed to be implemented therefore couldn’t combine them into a single class. Functions were made whenever similar actions and checks were required. For major functions such as doSomething, the functions were implemented as how the pseudocode for the function is written in the spec.

I assumed that squirting water at an earth or boulder will not show up but still decrement the water count in TunnelMan’s inventory as this is the behavior the sample game showed.

I assumed that squirting water outside the boundary will not show up as this is what the windows sample game demonstrated/clarified on piazza.

I assumed my status bar was correct in format by following the direction from the spec and comparing the status bar with that of the mac sample game.

I assumed/clarified on piazza with the professor that a stunned protester can’t be stunned again from a squirt. Neither can a stunned a stunned protester be harmed by a falling boulder. (Piazza post 990)

I assumed/clarified on piazza with the professor that a squirt cannot annoy multiple protesters.

I clarified on piazza with the professor that a falling single boulder can annoy multiple protesters at once.

1. A description of how you tested each of your classes (1-2 paragraphs per class)

**StudentWorld** – Made the sonar be able to setVisible every object within a radius of a 100 compared to the original radius of 12. Then I counted the boulders, oil barrels, gold nuggets to make sure a correct amount of them were placed at differing levels. I easily switched levels by making a setLevel function in GameWorld.h. Also, made sure that the protester would spawn immediately as a level started. Lastly, I made sure that whenever I press escape there were no errors thrown from cleanup and played until level 13 to make sure no memory leaks were happening.

**TunnelMan** – Controlled the TunnelMan. Saw if it can move and dig earth properly with minimal lag. Saw if it didn’t walk outside the boundary by changing the status bar to display TunnelMan’s X and Y coordinates. Saw if it could squirt water by pressing space. Saw if it could drop gold when it had one in its inventory by pressing tab. Saw if when it picked up or discovered an item or annoy a protester and made them give up it appropriately increased scores. Saw if it could activate sonar. Saw if it made the appropriate sound for each action.

**Earth** – Changed the status bar to display TunnelMan’s X and Y coordinates then I walked to the start of the Tunnel column and end of the Tunnel column to make sure it was the right location and width. Made sure they turned invisible when a boulder was placed or when TunnelMan digged earth.

**Boulder –** Made sure that correct number of boulders were spawned for each level by using the formula. Made sure that Earths were removed properly when they were spawned. Made sure that they fell when there was no earth underneath. Made sure that they annoy/immediately kill the TunnelMan and all protesters by either going underneath the rock or luring protesters to go underneath a falling rock. Easily tested this by going to level hundred by making a function, setLevel, and luring many protesters under a boulder. Made sure that when a boulder hits a boulder it disappears. Made sure that boulder doesn’t fall outside the boundary of the game (i.e. disappear when it reaches the game boundary).

**Squirt** – Froze each frame to make sure the squirts were moving the way it should while comparing it to the sample game (also froze each frame of the squirt moving). Made sure that three squirts will make a regular protester give up (because 3 squirts = 6 health decrease when regular protester is 5 in health). Made sure that 10 squirts will make a hard-core protester give up (10 squirts = 20 health decrease when hardcore protester is 20 in health). Made sure that squirts did not pierce through earth or boulder by shooting at the earth and boulder. Made sure that TunnelMan started with 5 squirts. Made sure that each time TunnelMan shoots water, the water in the inventory decreased by 1. Lastly, made sure that squirt doesn’t get created or show when it goes out of boundary by shooting up, down, left, right at each boundary of the oil field.

**Oil Barrels**

Made sure that correct number of barrels were placed in the game by looking at the status bar. Made sure that oil barrels became visible when TunnelMan comes close to it simply by playing a lot. Made sure that oil barrels incremented score by a 1000. Made sure that when all barrels were collected the game progress to the next level.

**Gold Nugget**

Made sure that they start off invisible and correct amount were placed at the start of the game. Made sure that gold nugget can be picked up by both protester and TunnelMan but not both. Tested this by picking up gold nugget and dropping it. Then I walked through it with the TunnelMan and made sure it didn’t get picked up, but the protester could. Made sure that they turned visible when TunnelMan is within a radius of 4 from gold by playing the game a lot. Made sure TunnelMan played the right soundgotgoodie when it picks up a gold nugget by picking it up in a game. Made sure when a gold was picked up it increases the score by 10. Made sure that the inventory was updated correctly when a gold was picked up. Made sure that only one protester would get bribed when multiple protesters are within range of the nugget by waiting for protesters to align and dropping gold in front of the protesters. Made sure that when a regular protester picks up the gold, it increases the score by 25 by dropping a gold nugget in front of the regular protester. Made sure that when a hard core protester picks up the gold, it increases the score by 50 by dropping a gold nugget in front of the hard core protester. Made sure that the gold disappears when it was dropped by a TunnelMan after a certain time.

**Sonar Kit**

Made sure that they spawn at the upper left corner of the oil field. Made sure that they can be picked up. Made sure when picked up the TunnelMan’s score increased by 75. Made sure it played soundgotgoodie when picked up. Made sure that the inventory gets updated correctly when sonar is picked up. Made sure that they can change objects to a visible state when within a radius by changing the radius from 12 to a 100.

**Water Pool**

Made sure that they spawned in places of the oil field where there are no earths or any other objects. Made sure that they can be picked up by the TunnelMan. Made sure that they correctly increase TunnelMan’s water inventory by 5. Made sure it increases the TunnelMan’s point by a 100 when it gets picked up. Made sure the soundgotgoodie plays when water is picked up. Made sure that the water pool disappears after their lifetime has elapsed by simply observing.

**Protester**

**Regular Protester –** Made sure that they spawn at the upper right corner of the oil field. Made sure that they have the right image id of regular protester/TID protester. Made sure that they stop and change directions when faced an obstacle (earth, boulder). Made sure that they yelled at the TunnelMan when he came into radius of 4 units of the protester. Made sure that they don’t keep yelling but yell after each 15 ticks. Made sure that after each yell, the TunnelMan’s health gets decremented by 2/20%. Made sure that by placing TunnelMan and Protester at a clear vertical or horizontal line of path, the protester would recognize the TunnelMan and start walking towards him. Made sure that protester doesn’t always make a perpendicular turn by observing. Made sure that it can make a perpendicular turn. Made sure that it doesn’t always choose one direction for its randomized direction. Made sure that when the TunnelMan makes the protester give up it walks back to the spawn location by carving out multiple paths and seeing if it will take the most optimal one.

**Hardcore protester –** Same as above but made sure that it can sense TunnelMan’s cellphone by configuring GameWorld.h and making a setLevel function. With this function I set it the game level to a 100 and made sure that the hardcore protester would immediately make its way toward TunnelMan. After testing, I deleted this function.