1. **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. For example, “I chose to define a pure virtual version of the blah() function in my base class because all Actors in TunnelMan must have a blah function, and each type of actor defines their own special version of it.”**

*Note: I have provided a brief overview for all functions in Actor.cpp but have not provided pseudocode and that is essentially provided in the spec. However, for the main methods in StudentWorld.cpp, I have provided the pseudocode as I had to design that code myself.*

**Actor:** Base class for all other game objects

**Methods:**

1. virtual bool isAlive() = 0;
2. virtual void doSomething() = 0;
3. virtual void getAnnoyed(int amt, int points) = 0;
4. virtual void setDead() = 0;
5. virtual StudentWorld\* getWorld(){return m\_world;}

The method isAlive() is required for every game object and is therefore virtual. Since the StudentWorld holds a pointer to game objects, and it must check for dead objects at every tick, this method is required.

The method doSomething() is an obvious choice here to be called during every tick to urge actors to move. The spec strongly hints at the need for this function.

The method getAnnoyed() is also mentioned in the spec as a different section for each game object and therefore it makes sense to include it here.

The method setDead is only really used once in this script but its need to be invoked in StudentWorld requires its existence. I could probably do away with it but to save time, I decided to add it and make it a dummy function for most objects.

Finally, when any object is created in StudentWorld, it needs to remember which world it was created in so that its methods can look at other objects in that world. Therefore, getWorld() is required.

**Tunnelman:**

**Methods:**

1. virtual void setDead(){m\_hp = 0;}
2. virtual bool isAlive(){return (m\_hp > 0;)
3. virtual void doSomething();
4. virtual void getAnnoyed(int amt, int points){m\_hp -= amt;}
5. void tryMove(Direction dir);
6. void incSonar(){m\_nSonar++;}
7. void incOil(){m\_nOil++;}
8. void incGold(){m\_nGold++;}
9. void incSquirts(){m\_nSquirts++;}
10. 10. void decHealth(){m\_hp--;}
11. int getOil() const {return m\_nOil;}
12. int getHealth() const {return m\_hp;}
13. int getSonar() const {return m\_nSonar;}
14. int getGold() const {return m\_nGold;}
15. int getSquirts() const {return m\_nSquirts;}

Methods 6-10 are simple functions to modify tunnelMan’s private members throughout the script. Similarly, methods 11 – 15 access those same members. In the first 5, the non-trivial public methods are doSomething and tryMove.

tryMove is a helper function, which is called in doSomething. It checks whether there’s a boulder or border in the specified direction. If there isn’t, then the tunnelman moves in that direction.

doSomething looks at the TunnelMan’s current position and “digs” the surrounding earth in the tunnelman’s 4 X 4 location by deleting any earth that is near him. It plays the dig sound if in each tick any earth was dug. It also looks at the user’s input and acts accordingly. These actions have been specified in the spec. The arrow keys move the tunnelman (here tryMove is called) and other keys make use of goodies or end the level.

**Earth:** The only relevant part of this object is its construction which has been specified in the spec. It doesn’t have any new methods as it’s only controlled by other game objects and StudentWorld.

**From this point onwards, if virtual methods return dummy values and are only written for the sake of making the class non-abstract, they have not been mentioned here.**

**Boulder:**

doSomething(): we have an integer private variable that represents the state of the boulder. We have another integer private variable m\_wait that starts off with a value of 30 and is decremented every tick so that the boulder can wait before falling. It transitions into its waiting state using a helper in StudentWorld “canFall” which sees whether there is any earth right below the boulder. After the waiting period, it will enter its falling state, move down one unit in the y-direction every tick and plays a sound, until it reaches Earth below and dies.

While its in the falling state, in every tick, the helper functions in StudentWorld annoyNearbyTunnelman and annoyNearbyProtestors are called. These check for the tunnelman and protestors respectively in a specified radius and if they are present, call their getAnnoyed functions.

**Squirt:** Represents a shot of the squirt gun that the tunnelman has.

Squirt has a private member variable m\_dist that tracks how long it’s already traveled and then declares itself as dead when its traveled 4 units.

It’s doSomething has a helper canMove. It checks whether the squirt can proceed moving in its current direction by looking for boulders, earth, and borders one unit forward in that direction. doSomething calls annoyNearbyProtestors in every tick to see if it’s being shot to stun a protestor. If it can move, it moves in the direction its currently facing.

**Goodies:**

Barrels, Gold Nuggets, Sonar kits and Water pools are all classified as goodies as they can all be picked up by the tunnelman and some can be used later during the game. This information is passed on to the tunnelman when he picks them up by incrementing one of his private members (which is why we needed the mutators in the TunnelMan class).

Furthermore, all goodies have a common class:

*bool nearTunnelMan(double dist);*

This helper checks if the tunnelman is present within the specified radius. This is required to determine whether the goodies should be made visible to the tunnelman (oil and gold) and whether they need to be picked up by him and set themselves as dead.

**Barrel:**

Its doSomething() makes the barrel visible if its within 4.0 units of the tunnelman and if its within 3.0 units, it sets itself dead, increments the number of oil barrels the tunnelman has collected (StudentWorld checks if this is as many as there are on this level and proceeds to the next level accordingly), plays a sound and increases the tunnelman’s score.

**Gold Nuggets:**

Some of the gold nuggets functionality is identical to the barrel in that it also makes itself visible and can be collected by the tunnelman. However, it has a few additional private members which determine its visibility and permanence. If the GoldNugget naturally spawns, it behaves almost identically to the barrel. However, once its been picked up the player can press TAB to spawn a temporary nugget at its location. A private integer variable m\_wait handles this by decrementing its tick till it reaches 0 and then deleting itself. A temporary gold nugget dropped in this way, calls a function bribeNearbyProtestors (which is a helper in StudentWorld, which, itself calls a method in Protestor bribe() which simply tells the protestor to leave the oil field.) bribeNearbyProtestors checks for any protestor in a 3.0 unit radius of the goldnugget. If there is one, it returns true and doSomething of gold nugget increases the player’s score and sets itself to a dead state.

**Sonar and water pool:** These work in almost exactly the same way as oil barrels in that they can be picked up by the tunnel man, which increments some private variable and can then be used for some other purpose. If a sonar has been picked up, in Tunnelman’s do something, pressing Z scans a 12.0 unit radius of invisible goodies and sets them to visible if they aren’t already. When water is collected, TunnelMan’s squirt count is incremented by 5 and the tunnelman can then press SPACEBAR to create a squirt object in front of it (if it’s a valid location.) The main difference between these goodies and the previous two is that those are created and hidden away during the initialization of the game while these are always visible and spawn and de-spawn and regular intervals. However, this is done in StudentWorld’s move() so it will be covered there.

**Protestor:**

bool nearTunnelMan(double dist);

This function has been borrowed from the Goodies class. It’s possible to include the protestor as a goodie as well then this function would not have to be duplicated but I don’t think that makes sense since Protestor is not a Goodie.

**The next few functions are helper functions that the protestor needs in his doSomething function. However, they are quite general and can be used in virtually any object. It just so happens that no other object needs it since other than the protestor, the game objects are relatively unintelligent.**

bool isFacingTunnelMan();

Checks the protestor’s current direction. If, for example, it is up then we check whether they have the same x-coordinate and tunnelman’s y-coordinate is greater than the protestors. Repeat this (in a switch statement) to check for left, right and down as well.

bool canReachTunnelMan();

First checks if the tunnelman is in a straight line from the protestors (i.e. returns false if both x and y coordinates are dissimilar.) It also returns false if its within 4.0 units of the TunnelMan (ask the spec specifies). I think this is there to prevent a weird dance-y motion when the protestors get too close to the tunnelman. Then, say they are in a vertical line with x coordinates being the same. It will check the y coordinates from the tunnelman to the protestor for any boulders or earth. If the path is empty it will return true.

void moveInCurrentDir();

A self-explanatory method which is basically the same as TunnelMan’s try move except it will also not be able to move if there is Earth in its way.

bool canMoveInDir(Direction dir);

Again, self explanatory. Uses switch statements to check for boulders, barriers and earth in the direction its facing but **won’t** actually move.

Direction pickRandomDir();

Uses rand()%4 to pick a rand integer 0 to 3 and each has a corresponding direction. It also uses the previous helper (canMoveInDir) to keep selecting a random direction until it’s a valid one.

bool isAtIntersection(Direction& dir);

Protestor has a private integer that keeps track of the last time a perpendicular move was made. This method is only called if 200 ticks have passed. Otherwise, the tick count is simply decremented. A vector of directions stores viable directions. When the protestor reaches at intersection, i.e. if its facing right or left, it can move either up or down OR if its facing left or right, and it can move either up or down, it prepares to return true and fills the vector with the viable perpendicular direction moves it can make. If this vector is empty, the function obviously returns false. Other rand%(size of vector) is used to randomly pick a direction for the protestor to turn in.

virtual void getAnnoyed(int amt, int points);

As mentioned before, this method is only called by squirt and boulder when they make “contact” with the protestor. It decrements the protestor’s health by the specified amount and gives the player a certain number of points. It is also in charge of playing the sound of the protestor getting annoyed.

std::string Protestor::pointToPath(int ex, int ey);

This uses a queue based BFS algorithm to convert the current world into a maze and return a string path of the shortest exit from protestors current location to 60, 60. I used a Point struct and a corresponding “prev” char array which shows the directions that led up to a given position. Therefore, by tracing back from the end to the start and appending the corresponding directions to a string, I was able to get the shortest path to the exit.

std::stack<Actor::Direction> Protestor::createExitStack(std::string path);

This uses the above mentioned path as an input, reads each character and stores a corresponding stack of directions, which serve as instructions for the protestor when it has to leave the field.

**virtual void doSomething();**

The exact pseudocode has been given step by step in the spec so here I have focused on describing *how* I have implemented each step rather than what I’ve done.

I created a private variable to track every 4th tick and only call this function then to make sure the protestor moves slowly. Another private variable is initialized with a randomly generated number between 8 and 60 inclusive that represents the number of squares a protestor should walk before randomly changing its direction. Another one checks for the last time it shouted and doesn’t let it shout again before 15 ticks as well as before the tunnelman moves from his location.

If the protestor is in the leave state, the protestor should exit*.* It creates a new stack of directions it should follow to the exit (using the two helpers mentioned above) and then starts on this path. Otherwise,it uses nearTunnelMan and isFacingTunnelMan to determine if it should shout. After it shouts, it resets the last\_shout timer to 15 ticks and decrements it every time after. Otherwise, if it canReachTunnelMan, we’ll set its direction accordingly and make it increments its location one step towards the tunnelman. Otherwise, it will simply continue walking in its direction, decrementing numSquaresToMove with every move and when it reaches 0, randomly selecting a new value. In the case, this movement brings it to an intersection, we use isAtIntersection to get a random viable perpendicular turn to make and change the direction.

**Hardcore Protestor:**

I have implemented the protestor in the same way as I did the regular protestor, the only difference being that when the protestor gets bribed, it gets stunned instead of leaving the oilfield and some values for updating scores needed to change. Other than that, the only additional feature to be implemented was the cellphone tracking. However hard I tried, the game kept freezing when I would implement this feature so I decided to finally comment out that code as I would rather get all my other points even if it means sacrificing a few here.

**StudentWorld**

The student world class is the setting for all actors in the game. As the spec mentions, it handles the initialization of a game, movements and updates during every tick, and cleaning up after a level is over. A lot of the helper functions that I have used here have already been mentioned above and it does not make sense to repeat them as their usefulness is best understood in the context of the game objects. Therefore, the emphasis in this report will be on init(), move() and cleanup() as well as the many methods that these functions call.

**Init()**

fillEarth(); // creates an array of earth pointers with the required dimensions. It also deletes the earth in a shaft going downwards

creating a new tunnelman and store its pointer as a private member

distributeBoulders(); // uses rand() to find appropriate locations to insert boulders. Uses a dist() function for successively placed game objects to ensure that they are 6.0 units apart. Then, any earth surrounding it is deleted.

distributeOil(); // as specified by the spec, distributes some number of oil barrels randomly hidden in the earth

distributeGoldl(); // as specified by the spec, distributes some number of gold nuggets randomly hidden in the earth

return continue\_game\_status;

**Move()**

setDisplayText(); // uses a helper formatter function that reads values of gold, score, level, oil barrels, squirts left, etc and displays it in a specific manner at the top of the screen.

Iterating through all actors in the vector of game object pointers:

actors[i]->doSomething();

tunnelman->doSomething();

*spawn any new actors as and when required:*

trySpawningGoodies(); // according to some given probabilities, spawn either gold or water pools for a temporary period.

trySpawningProtestors(); // while the number of protestors on the oilfield is less than the limit, if enough time has passed, spawn new protestors at 60,60 with a given probability of them being hardcore protestors.

*deal with any dead actors:*

Iterating through all actors in the vector of game object pointers:

if not actors[i]->isAlive() {

if it’s a protestor, make sure to decrement the private member that tracks count

delete the actor

erase the pointer to that actor from the vector

}

If not tunnelman->isAlive {

Play sound of player giving up;

Decrease live count;

Reset number of protestors to 0;

Reset time last protestor spawned to 0;

return player dead status

}

Calculate the total oil that should be there based on the level.

if (tunnelman->getOil() == totalOil){

play sound of player finishing level;

Reset number of protestors to 0;

Reset time last protestor spawned to 0;

return status of finishing level;

}

else return continue game status;

**CleanUp()**

Iterating through the 2D array of earth objects, if something is not a nullptr, delete it and set it to nullptr.

Delete the tunnelman and set to null pointer;

Iterating through the vector of game objects, delete each actor and erase that pointer from the vector.

1. **A list of all functionality that you failed to finish as well as known bugs in your classes, e.g. “I wasn’t able to implement the Squirt class.” or “My Hardcore Protester doesn’t work correctly yet so I just treat it like a Regular Protester right now.”**

Hardcore Protestor cellphone tracking:

Although I know that this should work using the same BFS algorithm as before, when I try to implement this feature, for some reason, the protestor gets stuck in one of my else if statements and the game freezes. I have left my portion of the code commented out in case I am able to get partial credit for that.

Tunnelman Dig speed:

I think that my tunnelman digging speed is slightly slower than the one in the spec. This could be a hardware issue as I’ve only seen a video of the prof play the game (the sample still doesn’t work for me ☹) However, I don’t think this will be an issue as I got 20/20 in my part 1 of the project.

1. **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.**

Protestor movement after shouting:

The spec mentions that after the protestor shouts, it can’t move until 15 resting ticks have passed. This is what I initially implemented. However, in the video uploaded, I noticed that the protestor also stays in its place until the tunnelman is 4.0 units away from it. Therefore, I finally stuck with that implementation as prof specified that if there’s any ambiguity, always follow the spec.

Game status bar:

Another part that is ambiguous is the status bar. The spec shows an example of one format for the status bar. However, a different format is used in the sample video posted. I prefer the one used in the spec so I’ve stuck with it as the professor said that either is fine.

Goodie spawning

The spec doesn’t mention that sonar kits and water pools should also be 6 units away from other game objects but it gets too crowded otherwise, the sample seems to avoid this crowding and even on piazza prof mentioned we should still do it.

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

Tunnelman:

All of the tunnelman’s movement is related to other classes, that’s why most of the description has been given elsewhere. Briefly, I checked all the input keys and whether they do the right thing. As well as pressing escape, losing lives, finishing levels and quitting the game. I approached boulders from all 4 sides to ensure there was no overlap in any possible situation. I saw that its health was being correctly reduced when being shouted at by protestors and when it dies to a boulder. Further, it could correctly move to all the borders as required by the spec.

Earth:

I tested this by digging around the earth from corner to corner, checking that there are no bad access errors, listening to and comparing the digging sound and the visual of digging the earth, digging 1, 2, and 4 earth objects in a single tick and ensuring that only one sound is being played. Other testing involves making sure that objects like squirts and protestors are being appropriately blocked by earth and earth is appropriately covering hidden objects like gold and oil.

Boulder:

Seeing that the correct number of boulders are being spawned at each level. (This was the hardest part because the game is genuinely not easy.) Spawning 30 – 40 boulders at a time to see that they are appropriately spaced, don’t land in the mine shaft, don’t overlap, and clear the earth around them when being spawned it. I checked the falling feature by digging the earth underneath it, listening to the sound and verifying that it dies at the correct tick. Also, I verified it effectively kills the tunnelman and protestors. Like earth, it appropriately blocks game objects.

Squirt and Water:

I checked that the squirt animation and sound works appropriately. I checked that it lasts for the correct amount of 4 units before dying. I verified its effect on the protestors. I checked that it gets blocked by earth and boulders and dies prematurely if fired too close to them.

I checked that water pools spawn in correct locations (un-earthed regions) and last for the appropriate time. I also roughly checked the frequency distribution between sonar kits and pools which seems to be 20/80 as it should be but sometimes is a little higher and leans towards water pools. I checked that its appropriately deleted when approached by a tunnelman but not a protestor and increments the squirt count by 5, which is displayed on the status bar and also reflects in the tunnelmans ability to shoot more squirts.

Sonar kits

As described for water pools, I tested for spawning frequency. I also checked for the tunnelman’s ability to scan the 12 radius region for hidden objects. However, this was slightly difficult to test as I had to make sure all hidden objects are really being revealed which is hard to know since the objects are hidden! Finally, I figured out a way to cout the objects that *should* be revealed and comparing it with those that are.

Oil Barrels

Like the other goodies, oil barrels with very straightforward to test for. Other than the usual picking up and deletion, I just needed to check that the correct number are being spawned and it reflects in the status bar how many are left. Additionally, once all have been picked up, the player automatically progresses to the next level.

Gold Nuggets

Other than spawning, being collected and increase a count, gold nuggets have an additional feature. Therefore, I tested dropping them as bribes for protestors multiple times. They were picked up normally by protestors, after which the protestors immediately left the oilfield. They also could not be picked up by the tunnelman. I felt like the wait time before despawing was too little but I followed the spec and didn’t have the sample to test it so I’ve decided to let this stay the way it currently is. I also verified that no gold nuggets can be dropped if the tunnelman has 0 on him.

Protestors

I checked that they spawned at the correct location, at the correct frequency and under the correct limit for each level. I verified that protestors are moving currently, following the tunnelman when they can see him, otherwise moving in a random direction for a random number of ticks and taking perpendicular turns occasionally. Within 4.0 units it moves towards the tunnelman and within 3.0 it will shout if it hasn’t before in the last 15 seconds. I also tested that the tunnelman freezes after shouting until the tunnelman moves again. With 5 hp and 2 hp per squirt, the tunnelman gets stunned and then dies after 3 squirts. A boulder falling on the protestor kills the protestor but doesn’t destroy the boulder. Lastly, I tested the exit strategy by annoying the protestor in differenct places and different oilfield configurations to see whether he exits correctly.

Hardcore Protestors

There wasn’t much to test here as I’ve only added the bribe feature which was relatively easy to test and the fact that instead of 3 squirts, I know need 10 squirts to fully annoy the protestor. Again, I could not test the cellphone tracking feature as I did not implement it.