1. A high-level description of each of your public member functions in each class
   1. Actor (derived from GraphObject):
      1. The constructor for the Actor class carries the information to be sent over to the GraphObject class, such as the imageID, startX, startY, StudentWorld\*, direction, size, and depth. In this constructor, we also set the private member variables (like the ptr to the Student World and dead or alive bool) to its respective allocations. In addition, I have set the Actor’s visibility to true.
      2. The destructor’s body is empty since all of the deletions of allocated memory take place in cleanUp(), which is not called in my code (and is instead called in the code provided for us).
      3. The doSomething function in this Actor class is pure virtual because it logically does not make sense for a regular Actor to do anything. Instead, each of the derived class has their own version of the doSomething function because they all do different things.
      4. The getWorld function retrieves a pointer to the StudentWorld class for the rest of the derived classes to use when they want to call a function from the StudentWorld class. This is not virtual because it does not need to be overridden.
      5. The isDead function simply returns each of the objects’ dead or alive status, and this is not a virtual function because this is the same code for all of the classes.
      6. The setDead function is used to set the dead or alive status to whatever argument gets passed in. This is the same bool that gets returned in the isDead function. This is once again not virtual because it does not need to be overridden in later classes.
   2. TunnelMan (derived from Actor):
      1. The constructor for this class simply has the pointer to the StudentWorld as its parameter, as this is a necessary argument in the Actor constructor (which needs to be constructed in the TunnelMan initializer list). In addition, the initializer list also includes the allocations of the hit points, water units, sonar charge, and golden nuggets to their respective initial values as specified in the spec. In the actual body of the constructer, the function setOilBarrels is called (explained later below), and the visibility of the object is called as true.
      2. In this class, there are many mini void and int functions that are for the main purpose of incrementing/decrementing a member variable by a certain amount and retrieving that member variable. All of these are not virtual since they will not be redefined later nor are there any more derived classes from the TunnelMan class.
         1. incrementSonar is a void function that increments the value of the private member variable sonarCharge by two.
         2. incrementGold is a void function that increments the value of the private member variable goldNuggets by one.
         3. incrementWater is a void function that increments the value of the private member variable waterUnits by five.
         4. decrementOil is a void function that decrements the value of the private member variable oilBarrels by one.
         5. getSquirts is an int function that returns the value of the private member variable waterUnits.
         6. getGoldNugs is an int function that returns the value of the private member variable goldNuggets.
         7. getSonar is an int function that returns the value of the private member variable sonarCharge.
         8. getOil is an int function that returns the value of the private member variable oilBarrels.
         9. hitPointsDecrement is a void function that decreases the number of hitPoints based on the int argument is accepts.
         10. getHitPoints is an int function that returns the current number of hitPoints.
             1. The above functions are *vital* for accessing the TunnelMan’s private member variables in other classes (like StudentWorld), since private member variables cannot be simply called upon in other classes. Thus, functions meant for the purpose of “setting” and “getting” is a primary part of OOP and is thus very prominent in this project.
      3. The setOilBarrels function is a void function that takes the current level’s required number of oil barrels (a function from the StudentWorld class) and sets that as the oilBarrels private member variable in this class. This is not virtual because it is very specific to this class and will not be redefined. This tracker is very important because it enables the program to know when to stop the level and move onto the next one (when all the oil barrels are found).
      4. The destructor for the TunnelMan is also empty because all of the memory deallocation occurs in the cleanUp function which is called outside of my code.
      5. The virtual void doSomething class (derived from the Actor class’s pure virtual doSomething) codes the main, specific actions of the TunnelMan, including going left, right, up down, facing left, right, up, down, dying, squirting water, using the sonar kit, and dropping a piece of gold. These cases are accommodated for using a switch case, where each input key separates each of these actions. Evidently, these actions are very specific to the TunnelMan. Here is a brief description of the design of this function:
         1. First, I check if the TunnelMan is dead, and if the TunnelMan is dead, I return immediately.
         2. Next, I utilize switch cases to determine which set of code should be called upon:
            1. Direction (left, right, up, down — these are all separate cases but have very similar code so I will write the description of them together): Check if the current direction is already the direction of interest and the coordinate of interest is within the bounds of the field (0 to 61). Then, check if there is a Boulder object in that direction, and if so, you cannot move. If there are no Earth objects (like in the tunnel) in that direction, you don’t have to dig (just go through). Else, you must set all the visible Earth objects in the direction of choice to invisible. If you dug, you must play the digging sound. Then, move towards the direction of choice. Then, you must tell the setChangeWorld function (which will be explained later) that the world has been changed (since the TunnelMan moved and also since it dug up some Earth objects).
            2. The escape case simply calls the annoyTunnelMan function to annoy the TunnelMan beyond its capability, which essentially immediately sets the TunnelMan to die.
            3. The space case is for squirting water, and there needs to exist at least 1 water unit in order for the TunnelMan to be able to squirt. The player squirt sound is played regardless of whether or not the image of the squirt is made visible. Then, the current direction of the TunnelMan is taken into consideration and is shot towards that way for four squares. If there exists a boulder or Earth object in the way, the image will not show up. The case for when a Protester is in the way is not coded for in the TunnelMan’s doSomething; rather, it is in the Squirt’s doSomething. The waterUnits is then decremented.
            4. If tab is pressed and the number of goldNuggets is greater than 0, it places down a new gold nugget on the ground by calling a function in StudentWorld (which can then add the gold nugget to the vector). This will be explained more later. The number of goldNuggets decrease by one.
            5. If z or Z is pressed and the sonarCharge is greater than 0, it will call a function in StudentWorld that traverses through the vector and sets all the goodies near the the TunnelMan’s location to be visible.
   3. Earth (derived from Actor):
      1. The constructor accepts the x position, y position, and the pointer to the StudentWorld, as these are necessary for the construction of the Actor class, which occurs in the initializer list of this class’s constructor. In addition, the initializer list sets Evisible to true, which is a function I made to easily access the specific Earth class’s visibility (which is later referred to in another function explained below).
      2. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
      3. The doSomething function is a virtual function because it was derived from the Actor class’s pure virtual function. In this class, the doSomething function does exist, but it does not do anything, since the Earth objects logically don’t do anything. To ensure that the Earth object is not an abstract base class, I’ve made the body of this function empty.
      4. As mentioned before, the setEvisible function sets the Earth class’s specific visible bool to whatever argument gets placed in the parameter. This may seem redundant with the setVisible argument already in the GraphObject class, but it is helpful for keeping track of Earth’s own visibility and being able to have a private member variable to be able to check later (and do actions based on it) is very useful. This is not a virtual function because it is very specific to Earth.
      5. Thus, the isEVisibleCheck is designed to return the value of the private member variable mentioned in the previous bullet point. This is very helpful in being able to determine if other Actors can walk through the coordinate and not have to stop. This is once again not a virtual function because it is very specific to Earth.
   4. Boulder (derived from Actor):
      1. The constructor takes in the specific x and y position of the Boulder object as well as the ptr to StudentWorld since it is necessary for the construction of Actor. In the initializer list, the object is set to be visible, have a stable status, and also has its own visibility setter (to keep track of its own member variable that can later be used in if statements to check if certain conditions are met).
      2. The void doSomething function in this class is virtual simply for good practice since that function is derived from the Actor class. First, it checks if it is dead and if so, returns immediately. Else, it checks the status, and if it is stable and there are still Earth objects below, it maintains its status and returns. Else, it changes the status to a waiting state and sets up a tick tracker to increment up to 30 before changing to a falling state, where the sound of falling rock is played. It will continue to fall down (over multiple ticks) until there exists Earth objects below it. When it runs into an Earth object, it sets itself to dead. If there were Protesters and TunnelMan near it as it was falling, it annoys them.
      3. The setBvisible function is a void non-virtual function (since it only applies specifically to the Boulder class) assigns the member variable of the boulder class to visible or not based on the argument. This assists with the maintenance of the boulder’s specific visibility, which is later used in another function to check if the boulder is visible, which helps for other Actors’ maneuver around the boulders.
      4. The isBVisibleCheck function is a bool non-virtual function (only applies to Boulder objects) that returns the private member variable Bvisible, which is mainly applicable to the boulder and thus only exists in this class.
      5. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   5. Goodies (derived from Actor):
      1. The constructor takes in the imageID, x-coordinate, y-coordinate, StudentWorld pointer, a bool to check if the Goodies should be shown or not, the direction, size, and depth. The initializer list then takes this information and passes it onto the Actor constructor. The goodiesVisible member variable is set to the bool that was passed into the constructor (show).
      2. The NearAction function is a non-virtual bool function (it will not be redefined in the Goodies’ derived classes) which checks if the goodies are not visible and the tunnelMan is near using a function from StudentWorld. If so, the Goodies’ visibility is changed to true and the function returns true. Otherwise, it returns false. This is used for when the TunnelMan walks closely to Goodies.
      3. The PickUpAction function is a non-virtual bool function that checks if the Goodies are visible and if the tunnelMan is close enough to pick up the Goodies. Then, it sets the Goodies as dead and returns true. Else, it returns false. This is used when the TunnelMan is close enough to the Goodies to pick them up.
      4. The virtual void doSomething function is a pure virtual function here as well, since the generic “goodies” cannot actually do something. Thus, the specific doSomethings of the goodies are specified in their respective classes.
      5. The setGoodiesVisible is a void non-virtual function that takes in a bool function and sets it to the goodies’ own goodiesVisible member variable.
      6. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   6. SonarKit (derived from Goodies):
      1. The Sonar Kit constructor takes in an x-coordinate, y-coordinate, and a pointer to the StudentWorld. In its initializer list, it passes on the image of a sonar, x, y, the pointer to StudentWorld, true as the bool for visibility, and depth and size. It also sets the ticks number to 0 (which is important for the doSomething function).
      2. The Sonar Kit’s virtual void function (simply good programming habits since it’s derived from its base class) checks if the sonar kit is “dead” and if so, promptly returns. Else, if the PickUpAction (TunnelMan is close enough to pick it up) function is true, the sound got goodie is played and the tunnelManInventory (a function in StudentWorld) is called, where the inventory of the TunnelMan is stored (explained later). Finally, the score is increased by 75 points. The sonar kit is also required to disappear after some time, so the number of ticks is tracked and if it is greater than the number of ticks it is supposed to last for in that level, it is promptly set to dead. Otherwise, the ticks increment.
      3. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   7. BarrelOfOil (derived from Goodies):
      1. The constructor takes in the x-coordinate, y-coordinate, and studentWorld pointer. In its initializer list, it passes in the image of the barrel, x, y, the studentWorld pointer, false for being visible, right (direction), and the size and depth.
      2. The virtual (simply good programming habits since it’s derived from its base class) void function doSomething checks if it is dead and if so, returns. Then, it checks if the TunnelMan is near the object using the NearAction function explained in the Goodies’ section and sets itself to visible. Then, if the TunnelMan gets close enough to pick it up, it becomes dead, plays the found oil sound, increases the score by 1000, passes the oil barrel into the tunnelMan inventory, and checks if the level is completed.
      3. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   8. GoldNugget (derived from Goodies):
      1. The constructor takes in the x-coordinate, y-coordinate, studentWorld pointer, a bool determining if it should appear (depending on if the TunnelMan drops the gold or it already exists in the world), a bool determining if it can once again be picked up by the TunnelMan, and a bool for the current state (permanent: hasn’t been picked up by the TunnelMan, temporary: has been dropped by the TunnelMan). In its initializer list, it passes on the image of the gold, x, y, the bool for whether or not it should appear, the direction (right), and the size and depth. In addition, it sets the bools pickUppableTM to the bool in the constructor (true or false depends on the context, as explained earlier), permanentState to the state in the constructor, and the ticks to 0, which is important in the doSomething() function.
      2. The doSomething function is a virtual (simply good programming habits since it’s derived from its base class) void function that first checks if it’s dead and if so, returns immediately. If not, if the current state of the gold nugget is not pick-uppable by the TunnelMan and the protester is near it, the gold nugget is set to dead and the sound of the protester finding gold is turned on, and the retrieveGold function in StudentWorld is called, which causes the protester to be able to pick up the gold and increment the points correspondingly (more will be explained later). If the gold nugget is in its temporary state and a protester wasn’t near, the ticks will increment and immediately return. Otherwise, if the ticks passed and no Protester picked it up, it will be set to dead. If the nugget is not in a temporary state and the TunnelMan is near it, it will set itself to be visible. If the TunnelMan is close enough to it to be picked up, it will cause the TunnelMan to play the sound got goodie, increase score by 10, and place the hold nugget in the TunnelMan’s inventory.
      3. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   9. WaterPool (derived from Goodies):
      1. The WaterPool’s constructor takes in a x-coordinate, y-coordinate, and studentWorld pointer. In its initializer list, it passes in the image of the water pool, x, y, studentWorld pointer, direction (right), and size and depth. The ticks for this goodie is initialized to 0.
      2. The virtual void doSomething (simply good programming habits since it’s derived from its base class) first checks if the water pool is dead, then checks if the TunnelMan is near enough to pick up the water pool and if so, the sound got goodie gets played, the Water pool gets placed in the TunnelMan’s inventory, and the score is increased by 100. If the TunnelMan is not near enough to pick it up and the maximum ticks pass by, the TunnelMan is set to dead. Otherwise, the ticks increment.
      3. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   10. Squirt (derived from Goodies):
       1. The constructor takes in a x-coordinate, y-coordinate, studentWorld pointer, and the direction. In its initializer list, it constructs Goodies with the image of the water sport, studentWorld pointer, bool for visibility, direction, and depth and size. In addition, the private member variables m\_tDistance (used to keep track of how many squares the squirt passed by), the m\_direction of the squirt, and ticks are initialized.
       2. The virtual void doSomething (simply good programming habits since it’s derived from its base class) checks if there is a protester near and if there is, it annoys the protester (a function in the StudentWorld class). If the m\_tDistance (maximum of 4) is 0 (the squirt has gone a distance of 4 squares), then it should be set to dead. Then, there are four if statements that check what direction the squirt is going, and it checks if there are any Earth or Boulder objects in that direction and if there are, it gets set to dead.
       3. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   11. Protester (derived from Actor):
       1. The constructor takes in the imageID, x-coordinate, y-coordinate, studentWorld pointer, hit points, direction, size, depth, and passes in the corresponding Protester’s imageID, x, y, studentWorld pointer, direction, size and depth. In addition, the initializer list also includes initializing the hitPoints, the bool for leaving the oil, ticks for keeping track of resting perpTicks (keeping track of how many ticks since last perpendicular motion), bool for keeping track of if the protester shouted, a tick for if the protester is stunned, and initializes the number of squares to move in the current direction.
       2. The Protester class also has many mini void, bool, and int functions that sets and gets certain private member variables without the class having access in it. These functions are all non-virtual since they do not need to be re-defined anywhere else. The private member variables include the variable keeping track of the resting ticks, stunned status, ticks in between shouting, ticks in between perpendicular moves, shout status, hit points, leaving oil, and number of squares to move in current direction.
       3. There is also a virtual bool function in this class called isRegular that is set to return true in this class, but will not return true in the HardcoreProtester (which is why this needs to be a virtual function, it will become re-defined elsewhere).
       4. The virtual void doSomething (simply good programming habits since it’s derived from its base class) first checks if the protester is dead and if so, returns. If not, it checks if it’s in the resting state and if so, just increments the resting tick and returns. Otherwise, if the stunned status is true, the stunned status will need to become false and would need to reset the resting ticks. Then, the resting, shouting, and perpendicular ticks increment. Then, we must check if the Protester is in the leave the oil status, and if so, it will follow an algorithm (coded in the StudentWorld) that leads it to the exit (unless it’s already on (60, 60)). Otherwise, if the TunnelMan is near the TunnelMan and facing the TunnelMan, it will shout at the TunnelMan and annoy them. If the protester is not regular (is hardcore), then they will also have the option to do the cell-phone reception, which uses more functions from StudentWorld that will be explained below. Next, if the protester can see the TunnelMan in a vertical or horizontal way (no Earth or Boulder objects blocking), then the Protester will chase the TunnelMan. If the Protester can’t see the TunnelMan, they will decrement the number of squares to go in that direction and choose a random direction to go to and if they can’t go in that direction (blocked by Earth or Boulder), they will choose another random number. Otherwise, the Protester will try to go take a perpendicular turn and reset the number of squares to move in the current direction after turning. Finally, after the direction has been set, the Protester will finally move in that direction unless that direction is blocked again, to which it would set the number of squares to move in the current direction to 0.
       5. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   12. RegularProtester (derived from Protester):
       1. The Regular Protester’s constructor takes in the x-coordinate, y-coordinate, and studentWorld pointer. It then passes on the image of the protester, x, y, student pointer, how many hit points (5), left, and the size and depth to Protester.
       2. It does not have its own doSomething function because the Protester base class already accounts for its action.
       3. It also does not have its own redefinition of the isRegular because it was already returned as true in the parent class.
       4. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   13. HardcoreProtester (derived from Protester):
       1. The Hardcore Protester’s constructor takes in the x-coordinate, y-coordinate, and studentWorld pointer. It then passes on the image of the hardcore protester, x, y, student pointer, how many hit points (10), left, and the size and depth to Protester.
       2. It does not have its own doSomething function because the Protester base class already accounts for its action, considering that the Hardcore Protester’s isRegular function returns false.
       3. Its isRegular function is virtual bool (simply for good programming habits since it’s redefining a function declared in the parent class) that returns false, as this protester is not regular.
       4. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
   14. Point:
       1. The constructor of this class requires the x-coordinate, y-coordinate, and the distance. These values are then stored into private member variables.
       2. There are also multiple mini void and int functions (all non-virtual since no classes are derived from this nor is this class derived from any class) that act as “setters” and “getters” that interact with the column, row, and distance variable.
   15. StudentWorld (derived from GraphObject):
       1. The StudentWorld constructor creates the GameWorld, sets the didWorldChange as false, and sets the protesterTick as 0 (for the adding of protesters in the move function)
       2. The destructor once again does not have anything in it because all of the memory deallocation occurs in the cleanUp function.
       3. The following functions are not virtual because they are not re-defining a function nor do they need to be redefined later:
       4. The getEarth function returns the Earth object at the coordinate x, y that is passed in.
       5. The squirtAction function simply pushes the Squirt object into the respective vectors.
       6. The sonarView function loops through the vector of Actors to see if any of them are nearby the x and y arguments and if so, sets them as visible and plays the sound sonar.
       7. The tunnelManInventory takes in an int (that is predefined as a more easily readable name, like PICK\_UP\_GOLDNUG) and increments the corresponding goodies’ count for the TunnelMan to use.
       8. The isCompletedLvl checks if the number of Oil that the TunnelMan needs to get is 0 and if so, return true. Otherwise, returns false.
       9. The blockingBoulder function checks if there exists a visible Boulder in the 4x4 space based on the x and y coordinates passed in and returns true if there is and false if not.
       10. The blockingEarth function checks if there exists a visible Earth object at the x and y coordinate passed in and returns true if there is and false if not.
       11. The facingTunnelMan function checks if the x and y coordinate of the Protester and the direction the Protester is facing the TunnelMan based off four if statements for each direction. If it is facing the TunnelMan, it returns true and if not, returns false.
       12. The annoyTunnelMan function takes in const ints that are pre-defined in a more readable way (like REG\_PROTESTER\_SHOUT) that decrements the hit points of the TunnelMan based off the action.
       13. The annoyProtester is a function similar to the annoyTunnelMan, but it decrements the hitPoints of the Protester instead of the TunnelMan for each pre-defined int that represents an action.
       14. The retrieveGold function is used for the protesters to pick up the gold nugget that is dropped off by the TunnelMan. If it is a Regular Protester, it will set to leave the oil, but if it is a Hardcore protester, it will simply leave them “stunned” and not moving for a couple ticks.
       15. The tunnelManNear function is used to see if there exists a TunnelMan within a 4 unit radius based on the x and y coordinates passed in. If there is a TunnelMan nearby, it return true and if not, false.
       16. The canProtesterTurn is very specific to the Protester’s perpendicular turns, as I pass in specific values so it always scans the 4x4 image from the bottom left. Ultimately, this uses a similar guideline as the checking for Earth objects, and it returns true if there are none blocking and false if there is a blocking Earth object.
       17. The tunnelManPickUp function calls the tunnelMan to pick up the goodies that is at the x and y coordinate passed in.
       18. The tunnelManHorizontalYLine checks the entire horizontal line that the TunnelMan is currently on, and it is used by Protesters to follow the TunnelMan. The same logic applies for the tunnelManVerticalXLine, except it checks the entire vertical line that the TunnelMan is currently on.
       19. The regProtesterTM checks the position of the Protester with the TunnelMan and returns a GraphObject::Direction.
       20. The setExitDirDisArrays function utilizes the maze-searching algorithm using queues taught in the stacks and queues lecture. The class Point assists with this search. I used a BFS to assign each coordinate in a 2D array of direction and another 2D array for distance a distance and direction that is best suitable to get to the exit. The same logic applies to the setTMDirDisArrays function, except the target location is now the TunnelMan’s coordinates.
       21. The getTMx and getTMy returns the TunnelMan’s x and y for the Actor.cpp file.
       22. The tunnelManHere function checks if the TunnelMan is at the x and y coordinate passed in.
       23. The removedeadGameObjects function removes all the objects in the game that have been set to dead in the current tick.
       24. The setDisplayText formats the structure of the display text for the scoreboard.
       25. The setChangeWorld checks if the world has changed (TunnelMan moved, Earth has been dug, boulder fell) and uses that information to check if the distance and directions array have to be reset.
       26. The getDirectionToExit and getDirectionToTM returns the direction in the x and y coordinate passed in from the 2D array that was initialized with the best direction to the exit and the TunnelMan respectively.
       27. The eraseEarth function erases the Earth objects by setting them to not be visible.
       28. The init function sets out all of the Earth objects, TunnelMan, random number of goodies, and sets the protesterTick to 0 so the move function knows to add a protester right away.
       29. The move function increments throughout each level, and new Protesters are added, the Actors’ doSomething gets called, and random Water Pools and Sonars get added as well.
       30. The cleanUp function deletes all dynamically allocated memory by going through the vectors, pointer, and arrays to delete them.
2. A list of other design decisions and assumptions you made, e.g.:
   1. The spec did not mention the speed of the protesters increasing as it left the oil arena (unlike the demo), so I decided to follow the specification and not change the speed of the Protesters leaving the game.
   2. The spec did not mention that the Protesters had to stop when yelling at the TunnelMan in a row (unlike the demo where the Protester stopped moving when yelling at the TunnelMan consecutively). I decided to follow the specification and not limit the Protester from moving around in between the shouting.
   3. The spec mentioned that “the TunnelMan cannot occupy a square that is less than or equal to a radius of 3 away from the center of any Boulder.” However, this is a bit misleading, as a shape with a radius of 3 must be a 6x6 image, but the Boulder is a 4x4 image, so the radius that the TunnelMan cannot occupy would be greater than the image of the Boulder (which also differs from the demo). Thus, I coded that the entire image of the 4x4 Boulder is inaccessible.
3. A description of how you tested each of your classes (1-2 paragraphs per class)

For the Actor class, there was no easy function to call upon, since the “action” function (doSomething) is pure virtual, since it does not logically make sense for a general Actor to do something. Thus, I was only able to test if this class works by testing the functions that are specific to this class, like getWorld and isDead/setDead.

I could tell that these functions worked because these few functions are the glue to my entire project, as the getWorld function enabled the Actor class’s derived classes to be able to use functions from the StudentWorld to utilize the vector of Actors, Earth objects, and the TunnelMan. In addition, without the isDead/setDead, my game would be extremely slow and the Actors would not go away as time went on.

For the TunnelMan class, I checked if the TunnelMan was able to move around and dig accordingly. The TunnelMan should not be able to leave the grid beyond the 64x64 area, and it should not go through (over) any Earth or Boulder objects. Thus, I thoroughly tested all the nooks and crannies that the TunnelMan could not go to, and the TunnelMan was unable to go. In addition, I checked if the TunnelMan was able to pick up a Sonar and have that be reflected in the display text (2 sonar objects incremented), a gold nugget and have that be reflected in the display text (1 gold object incremented), water pool (5 water pools incremented), oil (oil count decremented by 1), and have its hitPoints decremented when it is annoyed by the boulder or a protester.

In addition, I checked if the escape case worked, which caused the TunnelMan to lose a life and restart the current level. The space case was checked by checking if the image flew out in the direction of the TunnelMan, and also if the TunnelMan was close to an Earth or Boulder object, the image of the Squirt would not show. If the TunnelMan was close to a Protester, the image would not show and would also annoy the Protester. The tab case was checked by dropping the gold and having it go away after a couple of seconds, having a Regular Protester pick it up (bribed), and having a Hardcore Protester pick it up (stall). The z and Z case was checked by clicking it near a largely unsearched area and checking if Goodies become visible around it.

The Earth class was tested by being able to have the tunnel at the beginning of each level (which are null pointers in the 2D earth pointer array) and fill in the remaining except the top four rows in the display. In addition, it was tested to see if it would effectively disappear when the TunnelMan dug through the Earth objects. Otherwise, since there is no doSomething function, there was nothing that an Earth could really “do”.

The Boulder class was tested by seeing if the TunnelMan or Protester could not go through it. In addition, I tested if the Boulder was able to disappear as soon as it hit the bottom row or if it hit an Earth object but not a Protester object. In addition, I tested if the level would stop and reset (and decrement a life from the player) if the Boulder fell on the TunnelMan. Finally, I tested if the Boulder would cause any protester (Regular or Hardcore) to give up immediately and leave the oil field.

The Goodies class is another “basic” class, as it encompasses all the goodies classes. I was able to test this despite this not having a doSomething (pure virtual) by implementing the Goodies’ functions into the other goodies classes’ implementations of the doSomething. One such example is the NearAction, where the Goodies should show up as visible even if they previously aren’t when the TunnelMan goes by them. Another is the PickUpAction, which sets the Goodies to dead and allows the TunnelMan to pick them up.

The Sonar Kit was mainly tested to see if it was able to be picked up by the TunnelMan and have the inventory of the TunnelMan’s Sonar Kit to increment by 2. In addition, the sonar kit should allow the TunnelMan to be able to see the objects that are near to it. Also, the Sonar Kit should disappear after a certain amount of ticks, and it must not be pickable by the Protesters and should always spawn at the top left.

The BarrelOfOil class was tested simply by having the TunnelMan find the oil and making sure that finding the correct amount would make the level increase. There is no other real purpose to the BarrelOfOil class other than acting like the main reason for the game to continue.

The GoldNugget class was tested by having it be initialized as not visible at the start of each level, then as the TunnelMan approached the area, the GoldNugget would appear and be pickable by the TunnelMan. Then, to check if it is able to be dropped off by the TunnelMan, I would press TAB and then make sure that the object is visible and the TunnelMan cannot pick it up again. In addition, I made sure that it could bribe the Regular Protester to leave and the Hardcore Protester to stall, and if the Protesters couldn’t find it, it would disappear and be set to dead.

The WaterPool class was tested by utilizing the randomly dispersement of the WaterPool throughout the game in the move() function. It should be able to be picked up by the TunnelMan and increment its water units count by 5, and those water units should be able to be used towards squirts. The WaterPool objects cannot spawn on Earth or Boulders, so I made sure to keep an eye out for those, and there were no spawns in that area.

The Squirt class was tested in these ways: squirting near an Earth or Boulder (sound plays, water units decrement, but no image shown), squirting on a Protester (stuns regardless of Regular or Hardcore, the number of Squirts to cause them to give up is different) and making sure the image also doesn’t show in this case, and squirting in an empty area (image shows).

The Protester class (inclusive of the Regular, since the Regular Protester is almost purely identical to the Protester class) was tested by having the Regular Protester take squirts from the TunnelMan, and after 3 shots, it leaves. This is another thing that I had to check: the Protesters leaving the array (checking the maze-search algorithm), and it was able to successfully leave. In addition, it cannot walk through boulders or Earth objects. Also, it should be able to turn throughout the oil grid, and it should be able to shout at the TunnelMan. It should also be able to chase the TunnelMan if there exists no Boulder or Earth objects in the way in the horizontal and vertical line.

The Hardcore Protester class is almost identical, except it should also be able to have the cell-phone capability of finding the shortest distance and going that distance if there are no Earth or Boulder objects in the way.

The StudentWorld class is the backbone of the entire game. In particular, I checked that the init function worked by ensuring that the set-up of the Earth objects was correct, the Goodies were randomly dispersed and unable to be seen, the TunnelMan was created in the correct location, and the Protester occurred in the very first tick in the correct, top right location. In addition, I checked that the move function worked, by seeing that Protesters would drop in every so often (according to the level), each actor would do something, etc. Finally, the cleanUp function was checked by looking at the status of the next level. If there were no remnants from the last level, the cleanUp function did the job well.