* Actor
  + Actor(StudentWorld\* world, int imageID, int startX, int startY, Direction startDirection, float size, unsigned int depth) : GraphObject(imageID, startX, startY, startDirection, size, depth), m\_world(world), alive(true)
    - I created this constructor for the Actor class, which provides the correct values to its parent “GraphObject” so that the constructed actor can be graphed on the game screen and the variables can be initialized. This is important to the actor class as all actors have visual aspects and need to be drawn and GraphObject is used to define the parameters for their drawing.
  + virtual void doSomething() = 0;
    - I chose to implement this class as a pure virtual function in Actor as all actor objects need to do something every tick yet they all have to do something different.
  + void die(){alive = false;}
    - I chose to implement this class as a function that can be inherited as all the actors have the ability to “die” and there has to be a function to change that state
  + bool isAlive(){return alive;}
    - I chose to implement this class as a function that can be inherited as there has to be a way to check if they are alive or not so it can be known if they must be deleted and set invisible
  + StudentWorld\* getWorld() const
    - I chose to implement this class as a constant function so that it does not change and all it does is return a pointer to the StudentWorld of the current object. It is in Actor because all actors will need to have a link back to their resident StudentWorld class so that they can be placed into the game world and their states can be altered every tick.
  + virtual ~Actor()
    - I created a virtual destructor for Actor which calls the die function before destroying the actor. I chose to make this function virtual as any classes of which the Actor class will be the base class may need to call both their own destructor in addition to the Actor class’s destructor. If this class was not virtual, there could be a risk of memory leaks
* Tunnelman
  + Tunnelman(StudentWorld\* world, int imageID = TID\_PLAYER, int startX = 30, int startY = 60, Direction startDirection = right, float size = 1.0, unsigned int depth = 0) : Actor(world, imageID, startX, startY, startDirection, size, depth), hit\_pts(10), water(5), sonar(1), nuggets(0)
    - The tunnelman constructor, constructs a tunnelman object with starting hp of 10, water of 5, sonar of 1, and nuggets of 0, which are all important aspects of the game, in addition to using the setVisible() function to display the passed imageID of TID\_Player on the Game Screen
  + void isAnnoyed(int hP);
    - removes HP from tunnelman in addition to checking if he is dead
    - Important to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets
  + virtual ~Tunnelman() //destructor
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
  + void ableToMove(int ch, int dest\_x, int dest\_y);
    - Checks if the tunnelman is facing the direction seeking to move in, if not changes direction to that direction. Important because
  + virtual void doSomething();
    - Gets the users key input every tick: calls different functions depending on which key on the keyboard/direction is pressed. In charge of creating a sonar object, a squirt, moving, dropping gold, ending the level, etc.
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
  + void add(int ID);
    - adds values of specific object type to tunnelman inventory
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets, and other classes may need to alter these values
  + void shoot();
    - Shoots a spurt if space was pressed, creating a new Squirt object in the direction the tunnelman is facing while checking if the squirt object will go out of bounds, hit earth, or hit a boulder first
    - Chosen to be in Tunnelman because the tunnelman object can only tell when the spacebar has been pressed to be able to create this object, through doSomething
  + void decHitPts(int decrement)
    - Removes HP from tunnelman corresponding to the number passed to the function
    - Helper function for isAnnoyed
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets
  + int getHitPts(){ return hit\_pts;}
    - Returns the value of the private variable hit\_pts;
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets and StudentWorld may need to access these private variables
  + int getWater(){return water;}
    - Returns the value of the private variable water;
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets and StudentWorld may need to access these private variables
  + int getSonar(){return sonar;}
    - Returns the value of the private variable sonar;
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets and StudentWorld may need to access these private variables
  + int getNuggets(){return nuggets;}
    - Returns the value of the private variable nuggets;
    - Assigned to tunnelMan because only tunnelMan has access to the private variables hit\_pts, water, sonar, and nuggets and StudentWorld may need to access these private variables
* Earth
  + Earth(StudentWorld\* world, int startX, int startY, int imageID = TID\_EARTH, Direction startDirection = right, float size = .25, unsigned int depth = 3) : Actor(world, imageID, startX, startY, startDirection, size, depth)
    - The Earth constructor constructs an earth object with starting size of 0.25 (which is important to make it 1x1 pixel size) and a depth of 3, so that it is below all other objects in addition to using the setVisible() function to display the passed imageID of TID\_EARTH on the Game Screen
  + virtual void doSomething()
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class albeit not having any functionality
  + virtual ~Earth()
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* GoldNugget
  + GoldNugget(StudentWorld\* world, int startX, int startY, bool pickupableByT, bool pickupableByP, int imageID = TID\_GOLD, Direction startDirection = right, float size = 1, unsigned int depth = 2) : Actor(world, imageID, startX, startY, startDirection, size, depth), tunnelPickup(pickupableByT), protestPickup(pickupableByP)
    - The GoldNugget constructor creates a gold object of imageID TID\_GOLD with a depth of 2 so that it is below protestors. The passed values in addition to the starting position and world include whether or not it is pickupbable by the tunnelman and another bool indicating whether or not it is pickupable by the protester. This is important because if a protester can pick it up, it should be set to visible and start a count for a certain amount of ticks to know when to sink into the oil field.
  + virtual void doSomething();
    - Every tick, checks if goldNugget is alive, does not allow it if it is placed by tunnelman over another GoldNugget, checks if undiscovered piece of gold and tunnelman within radius, set visible. It also checks if piece of gold pickupable by the tunnelman, and within range of the tunnelman pick it up. If the tunnelman cannot pick it up and a protester can, check if protester is within radius to be able to pick it up and then bribe it if it is, but if not, start the disappear timer.
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
  + void disappearIn(int ticks);
    - Tracks how much time until the GoldNugget sinks into the mud. Only activated if the nugget is placed by the tunnelman and is pickupable by the protester. Important to this class to know when to destroy the object if it is placed by tunnelman
  + virtual ~GoldNugget()
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* OilBarrel
  + OilBarrel(StudentWorld\* world, int startX, int startY, int imageID = TID\_BARREL, Direction startDirection = right, float size = 1, unsigned int depth = 2) : Actor(world, imageID, startX, startY, startDirection, size, depth), pickupable(true)
    - The OilBarrel constructor creates an object of imageID TID\_BARREL with a depth of 2 so that it is below protestors. The passed value in addition to the starting position and world indicates whether or not it is pickupbable by the tunnelman. This is by default set to true, and the object is by default set to invisible as the tunnelman must search for it.
* Boulder
  + Boulder(StudentWorld\* world, int startX, int startY, int imageID = TID\_BOULDER, Direction startDirection = down, float size = 1, unsigned int depth = 1) : Actor(world, imageID, startX, startY, startDirection, size, depth), stable(true), ticks(0), waiting(false), falling(false)
    - The Boulder constructor creates an object of imageID TID\_BOULDER with a depth of 1 so that it is below protestors but above earth. The bool values initialized (stable, waiting, falling) indicate the state the boulder is in, and the ticks integer counts the ticks so that the boulder can wait a bit before being set to the falling state. Initializes boulder to visible as it is seen at the start
  + virtual void doSomething();
    - Check the state of the boulder every tick and if it should be falling, waiting, or stable currently and update based on the call to the function needsToFall()
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
  + virtual ~Boulder()
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
  + bool needsToFall()
    - Checks if the boulder has either hit the player or the earth or has gone out of bounds and does not need to fall any further. Also checks if boulder has hit protester but will allow the boulder to continue to fall if so (but will fully annoy protester)
* Goodie
  + Goodie(StudentWorld\* world, int imageID, int startX, int startY, Direction startDirection = right, float size = 1, unsigned int depth = 2) : Actor(world, imageID, startX, startY, startDirection, size, depth), currTicks(0)
    - Goodie constructor created since both Sonar Kit and Water Pool share many similarities. This constructor initializes the current ticks to 0 as ticks will be added until the ticks needed to disappear is found
  + virtual void doSomething();
    - Checks what a goodie should do every tick and checks if it was picked up by the player, destroys current object and increases player score and respective object (sonar or water) if so
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
  + void disappearIn(int ticks)
    - Since both children, sonar kit and water pool, both have to be destroyed and also need to be destroyed in the same amount of ticks, this function captures both of their needs and does not need to be specialized and thus belongs in their parent class, goodie.
  + virtual ~Goodie()
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* SonarKit
  + SonarKit(StudentWorld\* world, int startX, int startY, int imageID = TID\_SONAR, Direction startDirection = right, float size = 1, unsigned int depth = 2) : Goodie(world, imageID, startX, startY, startDirection, size, depth)
    - Set to visible once it is constructed at a depth of 2 facing right and with the imageID of TID\_SONAR
    - Does not have a complex class as most functionality occurs in parent, Goodie
  + virtual ~SonarKit()
    - Destructor set as virtual to keep proper formatting as the parent class “Goodie” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* Water Pool
  + WaterPool(StudentWorld\* world, int startX, int startY, int imageID = TID\_WATER\_POOL, Direction startDirection = right, float size = 1, unsigned int depth = 2) : Goodie(world, imageID, startX, startY, startDirection, size, depth) {
    - Set to visible once it is constructed at a depth of 2 facing right and with the imageID of TID\_SONAR
    - Does not have a complex class as most functionality occurs in parent, Goodie
  + virtual ~WaterPool()
    - Destructor set as virtual to keep proper formatting as the parent class “Goodie” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* Protester
  + Protester::Protester(StudentWorld\* world, int imageID, int hP, int startX = 60, int startY = 60, Direction startDirection = left, int size = 1, int depth = 0) : Actor(world, imageID, startX, startY, startDirection, size, depth), tickSinceLastTurn(200), tickNoYell(15), stunned(false), leave(false), hitpoints(hP) //constructs protester
    - Constructs protester object, while randomizing numToMove and setting the tickRest to the formula stated in the spec. imageID, depth, startDirection, ticks since the last turn, ticks without yelling, whether or not the protester is stunned or leaving, and hitpoints are all initialized properly in this.
  + int getHP();
    - Returns the hitpoints of the protester
    - Assigned to Protester because only Protester has access to the private variables in this class and StudentWorld may need to access these private variables
  + void decHP(int points);
    - Decrements the hitpoints of the protester
    - Assigned to Protester because only Protester has access to the private variables in this class and StudentWorld may need to access these private variables
  + virtual void doSomething();
    - Updates the protester's actions every tick. If the protester is dead or exiting, do not check the rest of the cases in this tick. If not check if the tickRest is complete (this slows down the protesters actions so it is not as fast as the tunnelman). Check if the protester is within the range of the tunnelman to be able to yell and thus annoy them. If the protester is a hardcore protester, to find out where to move they can track the player if they are within the specified range. If the player is any protester that sees the player within the line of sight, they will immediately go towards them. Decrease num to move and walk in a random direction if not. Check if protester is at an intersection and turn if it has been more than 200 ticks since the last turn. If the protester can't move in that direction set numToMove to 0 so a new direction can be picked.
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + bool isStunned(){return stunned;}
    - Returns if the protester is stunned or not
    - Assigned to Protester because only Protester has access to the private variables in this class and StudentWorld may need to access these private variables
      * Important in implementation of not allowing stunned protester to be hit by boulder or water squirt
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + Direction directionToPlayer();
    - If the player is within the same axis of the protester (same X or Y coordinate), it returns that direction
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + Direction randomDirection();
    - Selects a random direction for the protester to go towards, which is not occupied by earth, out of bounds, or into a boulder
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + bool directPath(Direction direction);
    - If there is a clear path with no boulders or earth in the way to the player from the direction found in directionToPlayer(), this function moves the protester to the player directly
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + bool atIntersection();
    - Determines direction logic for protester if they are at an intersection
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + void turn();
    - Randomly pick a direction to turn to if both can be turned to or turn the other way if it is the only way to turn
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + virtual void bribe();
    - Determines what to do once a protester has picked up the gold dropped by a player
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + void moveInDirection(Direction direction);
    - Moves the protester in a specified direction passed to the function
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + void randomizeNumToMove();
    - Sets the random number of steps to move value as per the formula in the spec
    - Used in construction of Protester
  + void isAnnoyed(int hP);
    - Annoys the protester by a certain amount of hp and checks if they are dead or not and what killed them, acts accordingly
      * Important to be able to let other objects annoy protester and eventually fully annoy them
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + bool isLeaving(){return leave;}
    - Returns if the protester is on their way to leave or not
    - Assigned to Protester because only Protester has access to the private variables in this class and StudentWorld may need to access these private variables
  + void getStunned();
    - stuns the protester and makes it not move for a longer period of time than normal (causing it to stay in place)
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + bool isFacingPlayer();
    - checks if the protester is facing the tunnelman
    - Used in Protester because subsequent children classes can use this class to determine protest-specific movement and actions
  + virtual ~Protester()
    - Destructor set as virtual to keep proper formatting as the parent class “Actor” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks.
* RegularProtester
  + RegularProtester(StudentWorld\* world);
    - constructor for regular protester, imageID will be passed to it (through Protester constructor)
  + virtual ~RegularProtester(){}
    - Destructor set as virtual to keep proper formatting as the parent class “Protester” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks
* HardcoreProtester
  + HardcoreProtester(StudentWorld\* world);
    - constructor for hardcore protester, imageID will be passed to it (through Protester constructor)
  + virtual ~HardcoreProtester(){}
    - Destructor set as virtual to keep proper formatting as the parent class “Protester” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks
* Squirt
  + Squirt(StudentWorld\* world, int startX, int startY, Direction startDirection, int imageID = TID\_WATER\_SPURT, float size = 1, unsigned int depth = 1) : Actor(world, imageID, startX, startY, startDirection, size, depth), distance(0)
    - Sets squirt as visible upon creation while specifying imageID of TID\_WATER\_SPURT, size of 1, and depth of 1. The startDirection is passed to it depending on where the tunnelman is facing when this object is created. The distance variable is initialized to 0 as the squirt has not moved yet
  + virtual ~Squirt()
    - Destructor set as virtual to keep proper formatting as the parent class “Protester” has a virtual destructor as well. Both need to be virtual in order to avoid memory leaks
  + virtual void doSomething();
    - Updates every tick. Checks if squirt is alive. If it is, checks if the squirt is touching a protester through annoyProtester or has reached its max distance of 4; if so, die(). If it hasn’t then the squirt should move in the direction it was fired in for up to 4 spaces until distance is 4 and then it is destroyed or it comes into contact with a protester and triggers annoyProtester() to return true
    - Chosen to be virtual as this function was a pure virtual function in the parent, Actor, class and should be virtual and must be overridden in this class
  + bool annoyProtester();
    - Determines if a protester is within the radius of the squirt while it is active and if so, stuns it and decrements its hitpoints accordingly
* StudentWorld
  + StudentWorld(std::string assetDir);
    - Constructs the StudentWorld object from parent GameWorld
  + virtual int init();
    - Initializes the game world by dynamically allocating all starting actors. This belongs in the Student World class because the GameWorld class calls this function after the player hits enter on either the welcome screen, life lost screen, or the new level screen
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void initializeEarth();
    - This function is created to initialize the earth objects and is a helper function for init() so that the code looks cleaner
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void initializeBoulders();
    - This function is created to initialize the boulder objects and is a helper function for init() so that the code looks cleaner
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void generateGoodie();
    - This function is created to check if a goodie should be created, which goodie it should be, based off the probabilities given in the spec in addition to where they should be placed. This acts as a helper for the move() function as it is run every tick and is called within it.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + ~StudentWorld();
    - Destroys the StudentWorld object and frees dynamically allocated memory
  + void generateRandomCoordinates(int&x, int&y);
    - generates random coordinates for the oil and gold to be spawned, helper for the init function to help keep things organized and clean
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + virtual int move();
    - This function is called every tick by the Game Controller and must return continue game to go to the next tick, or end current level if a player has lost a life or has beat the level. This function takes in the actions of the player and the actions of the actors every tick.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void createProtester();
    - This function is created to check if a protester should be created, which protester it should be, based off the probabilities given in the spec. This acts as a helper for the move() function as it is run every tick and is called within it.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool moveIsValid(int x, int y);
    - Checks if the move is within bounds and not touching a boulder, used to check tunnelman’s movement in addition to the water spurt from the tunnelman’s squirt
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool newMoveOverlap(int x, int y, Actor\* b);
    - Checks if the new move would overlap the passed actor, helpful for tracking collisions using x and y coordinates only. Used as a helper function for multiple classes that track collisions or overlaying such as goodieOverlap()
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool overlapEarth(int x, int y, Actor::Direction d);
    - Used primarily for the tunnelman’s movement. Checks if moving in the direction passed would cause the tunnelman to come into contact with any earth. Returns true if so false otherwise.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void boulderEarthOverlap(int x, int y);
    - Removes earth around the boulder for initializing by calling boulderShouldFall
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool boulderShouldFall(int x, int y);
    - Determines if there is visible earth under the boulder
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void sonarUsed(int x, int y, int radius);
    - Reveals any hidden oil or gold in the radius passed of which the calculation is detailed in the spec. Only called if z or Z is pressed.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void foundOil();
    - Subtracts one from the total oil counter initialized in the init function
    - Used in StudentWorld because only it has access to the private variables referencing important game data.
  + void display();
    - Displays the game text, properly formatted
    - Used in StudentWorld because only it has access to the private variables referencing all actors and game text data.
  + bool goodieOverlap(int x, int y, Actor\* b);
    - Checks if the coordinates passed would overlap with any actor in the objects vector, used so goodies don’t spawn on top of any existing actor. Also checks that the objects vector item is not referring the passed Actor\* b, this prevents objects from being placed on top of their own selves after they were added to the vector (ie. gold nugget from tab).
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool goodieOverlap(int x, int y);
    - Checks if the coordinates passed would overlap with any actor in the objects vector, used so goodies don’t spawn on top of any existing actor.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + Tunnelman\* getPlayer();
    - Return a pointer to the player character so protesters within “Actor.cpp” are still able to access the player character by calling this function after getWorld().
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + Protester\* protesterInRadius(Actor \* actor, int radius);
    - Checks if there is a protester within the passed radius of the actor passed and if so returns a pointer to that protester.
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool isThereBoulder(int x, int y);
    - Checks if there is a boulder in the passed coordinates
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool isThereEarth(int x, int y);
    - checks if there is a piece of visible earth assuming a 4x4 actor, given the coordinates provided
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool checkIfOutOfBoundsOrEarth(int x, int y);
    - Checks if the passed coordinates will go out of bounds or hit earth, used in order to know when the squirt object should be deleted
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool withInRadius(int x1, int y1, int x2, int y2, int radius);
    - Checks if the passed coordinates are within ‘radius’ of each other
    - Used in StudentWorld because all actors may have to use this function to determine overlaps
  + void decProtester();
    - Decrements the number of protesters currently in the game world
    - Used in StudentWorld because only it has access to the private variables referencing important game data
  + GraphObject::Direction leaveGame(Protester \* pr);
    - Creates the optimal path for the protester to leave the game. In this class since only StudentWorld has access to the Earth 2d array and position of the boulders and can help the protester find the path for it to leave (reach 60,60) without running into earth or boulders.
  + bool ableToMove(int x, int y, GraphObject::Direction direction);
    - Checks if it is possible to move in the given direction, considering out of bounds, if there is a boulder, or if there is earth
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + bool isPlayerInRadius(Actor \*actor, int radius);
    - Checks if the player is within the given radius of the passed actor and if so returns true else returns false
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + virtual void cleanUp();
    - Deletes all dynamically allocated instances, inherently resetting the game world, while staying on the same level
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + GraphObject::Direction hardcoreProtesterPath(Protester \* pr, int M);
    - Determines the optimal path for the hardcore protester to find the player character, within the radius M (calculated as per the spec)
    - Used in StudentWorld because only it has access to the private variables referencing all actors.
  + void newObject(Actor\* a);
    - Pushes a new actor into the objects vector
    - Used in StudentWorld because only it has access to the private variables referencing all actors.



* I was able to complete all functionality from what I can tell.



* It was ambiguous what to do when the protester is stunned and also hit by a boulder so I decided to make it pass through the protester (without destroying the boulder) allowing it to damage other protesters or the player
* It was relatively ambiguous whether or not multiple protesters should be able to be hit by the same squirt, so I decided to not allow it to happen and only allow unstunned protestors to be hit and affected by the squirt.



* StudentWorld
  + StudentWorld was probably the most important class for this project and had to be tested thoroughly because it controlled the flow of the game and ran everything behind the scenes basically. I had to ensure the game knew what to do at initialization by creating all the actor members needed after pressing enter. I tested that the tunnelman, the earth field, the oil barrels, the gold, and the first protester were all immediately created. I had to ensure that all the private count variables were initialized correctly in the initialization function as well since they would determine the efficacy of the program in terms of ticks, ensuring the correct number of a certain object are on the screen at one time, etc. I tested this by playing multiple levels; when a new level is created the initializer function is called again, and if something is not re-initialized correctly, it will not run correctly in the subsequent levels (as there should be a greater number of certain actors and a smaller number of others) or if escape is pressed and the level is reset (everything should be the same as you are on the same level). The move function was another very important one as it dictated what happened each second. In order to test that this program was working effectively, I checked if I was able to move my tunnelman through keyboard input and if that was the case, then I knew the doSomething() was being called in move() every tick effectively. Every tick, I also had to check if there were no Oil Barrels left to pick up to know whether or not to move onto the next level. The cleanup function was another very important function I had to test. When a new level is made or a new life is being used, I had to ensure that all objects were properly deleted so that all the dynamically allocated memory was properly freed so no memory leaks would occur once I played 5+ levels (which is how I tested that no memory leaks occurred).
* Actor
  + I had to test a lot of inheritance with the actor class. This became evident in the objects vector I created in StudentWorld which stored many actors. I tested if the variables in the actor class were passed correctly by calling die() and isAlive() within nearly all of my classes and all of them were able to follow the Actor structure through my testing. When I ran the game and did actions that was supposed to call the die() function of the Actor at hand, I could tell that it was able to set all of them to invisible and their alive variable to false which in turn led the StudentWorld world to destroy that object. This made efficiency much more effective rather than putting die() or isAlive() in each of the children classes of actor and thus Actor was a very effective abstract base class for all the subsequent classes I created.
* Tunnelman
  + I had many situations to test for the tunnelman. I first had to ensure that he was loaded in correctly with the correct starting direction in the middle and top of the game area with the correct image ID, size, and depth, and correct starting hitpoints, water, and nuggets which I tested by looking at the game screen and then reading the game text at the top of the screen. I then had to ensure that I could move around with the tunnelman through the arrow keys, wasd, and the numpad . Once this was accomplished, I had to make sure the 4x4 image of the tunnelman could not leave the bounds of the game and that anytime the tunnelman overlapped any earth object, the earth object was destroyed and the tunnelman played the sound of digging. I had to test all the different key inputs for the tunnelman as well. If I pressed space, the tunnelman had to check if he had more than 0 of the water variable and then if so create a squirt object in the direction the tunnelman was facing and decrement. If tab was pressed, the tunnelman had to check if he had collected any gold and then if so, drop one which the protesters could pick up and decrement the amount of nuggets the tunnelman has by 1. If escape was pressed, the game would immediately go to the life lost screen and the player give up sound would be played and then the player would have to press enter to reinitialize everything with 1 less life. If either lower or upper case z were pressed, the tunnelman would have to know to check if there were any sonars in the inventory and then reveal the location of objects around the tunnelman within a radius detailed in the spec, playing the sonar sound, and decrementing tunnelman’s sonar by 1. I would be able to verify this by going to different parts of the game field where there is a lot of earth surrounding the tunnelman and pressing z, watching gold objects and oil barrels (which I should not normally be able to see at that position) become revealed and 1 sonar being taken away from the game text at the top of the screen while the sonar sound is played.
* Earth
  + I tested that the earth would be spawned in the correct amount and correct 1x1 size when creating the 64 x 64 2d array of them by running the program by counting the pixelated earth rows and columns and ensuring that it aligned with both the tunnelman’s movement and the spec requirements. Furthermore, I checked that the earth objects would be set to invisible if they were in the top 4 rows, in the 4x56 middle tunnel at position 30 in the middle of the game world. Additionally if they were under a boulder at initialization or if they were dug by the tunnelman during any tick, they would be set to invisible. I tested these by simply running the game, looking at the starting earth layout and moving around with the tunnelman ensuring that the earth was being set to visible and the sound was playing once per movement of the tunnelman in which earth is destroyed.
* GoldNugget
  + I had to test that the goldNuggets which were pickupable by the tunnelman were invisible at first by starting the game and seeing all of them being invisible. Once I dug near them and was within a radius of 4, I was able to test if I could see them or not. Once visible, I checked that the correct imageID was loaded with the correct starting direction, depth, and size. When I got within a radius of 3, I had to check if my tunnelman could pick up the gold and if it would increment the gold counter in game text and play the picked up goodie sound. If that was the case, I continued by testing if I could press tab to drop the gold and see a gold object appear. This particular GoldNugget I should be unable to pickup as I was the one who had placed it and thus it should only be able to interact with a protester. The tunnelman character display should also be overtop of it as the tunnelman class has a more dominant depth than the GoldNugget. I then had to test if I could place a second GoldNugget right on top of that one. I had to ensure another object was not created as this should not be possible and that my gold counter in the game text remained the same. Once a protester got near this gold item, it would disappear as it had been picked up by the protester. However, if a protester had not picked it up, I had to make sure that the gold would be destroyed after the required amount of ticks specified in the spec so that it goes along with the instructions.
* OilBarrel
  + I had to test that the OilBarrel objects started off as invisible when I first started to run the game. In the same way as undiscovered GoldNuggets, they had to be set to visible once my tunnelman character was within a radius of 4. Once visible, I checked that the correct imageID was loaded with the correct starting direction, depth, and size. Once my tunnelman was in a radius of 3, I would check if the OilBarrel object had been set to invisible and the number of oil left to be found in the game text at the top of the screen was decremented by 1. I also had to check if my score changed accordingly and increased by 1000 at the top.
* Boulder
  + I had to test that the boulder would spawn in a 4x4 location empty of earth randomly and the amount of boulders spawning was consistent with the spec. I had to test that the boulders would start out in stable position facing down and would thus not be moving as they should have earth below them but once the tunnelman digs the earth under them they fall until either they go out of bounds, they hit the tunnelman, or they hit earth again. I had to ensure that hitting earth and having earth below the boulder meant even 1 1x1 earth object under it and that meant it could not fall or be destroyed if it were in a falling state. I also had to test that the boulder waited 30 ticks before falling after all the earth underneath it in a 4x1 row under its 4x4 image were destroyed. I had to test that the boulder falling on the player would lead the player to die and a life to being lost. I had to test that a boulder falling on a protester would lead them to leave the game and that a boulder would not disappear once hitting a protester but could fully annoy multiple protestors with one fall.
* Goodie
  + I checked if the objects (water pool and sonar kit) effectively implemented the doSomething() function of Goodie by checking that if either were within a radius of 3 of the tunnelman, they would disappear and the picked up a goodie sound would by played. The score would increase by 75, and the respective goodie would be added to the player’s inventory on the game text at the top of the screen. I also had to test if both objects disappeared (were set to invisible and destroyed) after the amount of ticks specified.
* SonarKit
  + I had to make sure that the SonarKit would only spawn at 0,60 and was the correct size, and imageID. I also had to make sure it correctly implemented the functions it inherited from the Goodie class by checking if the tunnelman could pick up the sonar kit and see the sonar on the top of the game screen increase by 1 while playing the picked up goodie sound. The other test I had to conduct was to see if the SonarKit would be set to invisible after a certain amount of ticks specified in the spec had passed. I also had to make sure that no SonarKit would spawn on top of another SonarKit, so if one had already been created in the game, a new one cannot be made on top of it.
* Water Pool
  + I tested that any water pool would only be randomly spawned in a 4x4 location, at the correct size and imageID, where the earth was completely cleared by running the game for some time and checking the cleared earth locations. I also tested that the water pool properly implemented the Goodie function for disappearing so that after the specified amount of time in the spec it would “dry up” and then be set to invisible. I made sure that the water pool would not spawn in a location on top of any other Game Object by playing multiple levels and monitoring the spawning behavior. I also ensured that the probability of spawning water pools to spawning sonar kits made sense in the 4:1 ratio.
* Protester
  + I tested if any protester would move directly in a straight line if the player was seen no matter what type of Protester it was. I also tested if a protester could leaveGame correctly and follow an efficient path to directly get to the top right corner after it was annoyed to the max. I tested that both protesters could be stunned by the squirt. Furthermore, I checked that protestors could be immediately fully annoyed by a falling boulder and that I also would be able to cause multiple protesters of any type to leave with one boulder if they are hit within succession. I checked that both types of protester would not be able to pick up or interact with any actor after they began to leave the game. I checked that both types of protester could pick up a gold nugget. I checked that both types of protesters would be able to randomly move if they could not sense the player in any way, whether that be through turning or determining logic at intersections. I ensured that the protesters could not move through boulders or the earth by being in a straight line under, above, to the right, and to the left of them and watching their movement. I made sure that any protester would be slower than the tunnelman and any protester would stop to yell at the tunnelman if he was within their radius. I also ensured that as I played more levels, more and more protesters were being added to the game.
* RegularProtester
  + I first tested if the correct protester image would show up when constructed and added to the objects vector in StudentWorld. I tested if the regular protester was spawned in more times in the earlier levels then more so in the later levels based on the hardcore protester probability. I tested that this type of protester would successfully get bribed and then leave the game when the player drops a gold nugget for it that it is within a radius of 3 of. Furthermore, once this protester picks up a gold nugget, the I’m rich sound should be heard and the score of the player should increase by 25. I made sure that as I played more and more levels, less and less regular protesters were being made and more hardcore protesters were being created. I tested that this type of protester would be fully annoyed and set to leave if I were to cause a boulder to fall on top of it, incrementing the score of the player by 500 (I checked the game text at the top of the screen for this). I tested that the hitpoints of the protester were 5 by squirting it 3 times to cause it to leave, with its actions being stunned each time I squirted it.
* HardcoreProtester
  + I first tested if the hardcore protester was spawned in the correct amount of times based on the probability calculated in StudentWorld. I then tested that the correct image would spawn in whenever the hardcore protester was created. Along with the ways I tested the regular protester, I had to test whether or not the protester could perform the hardcore protester function through hardcoreProtesterPath(). I did this by moving within “M” of the hardcore protester to check if it would find an open path to my tunnelman. I tested different situations with this: if there was earth or a boulder in the way and it should not come through those objects to get to me but should make turns to get to me, and also if the path is clear and it should come directly to me. I also tested if gold nugget bribery would not work on the HardcoreProtester but would only make them stop for a bit and increase player score by 50. I also checked that it would take more squirts (10 total) to fully annoy the hardcore protester compared to the regular. I tested that this type of protester would be fully annoyed and set to leave if I were to cause a boulder to fall on top of it, incrementing the score of the player by 500 (I checked the game text at the top of the screen for this). I made sure that as I played more and more levels, less and less regular protesters were being made and more hardcore protesters were being created.
* Squirt
  + I tested to see if the squirts would not go out of bounds of the game world when created by pressing space when facing out of the bounds. I further tested Squirt by squirting the protesters and ensuring that the correct amount of hp was decremented. I also tested the squirt object by ensuring that it would get destroyed if it came into contact with earth or boulders when I shot at either of those objects. I tested that the squirt would last for the correct distance by counting how many pixels it was launched. Additionally, I tested that the squirt would be oriented in the correct direction when the tunnelman player was facing different directions. I tested that the squirt object would not be destroyed by the first protester it encountered if that protester were stunned as it would have to be able to hit another protester behind it before disappearing. Otherwise, the squirt would hit the first unstunned protester it encountered, stun them, and disappear or disappear after traveling a distance of 4 if it hits no protesters.