Justin Yeung

305489135

8/25/23

Project 4 Report

**Class Dependency layout**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GraphObject | | | | | | | | |
| Actor | | | | | | | | |
| Humans (any human that moves on the board) | | | Earth | Boulders | Interactables (any item a human can pick up) | | | |
| Tunnelman | Protesters | |  |  | HiddenObjects | | Goodies (non-hidden objects) | |
|  | RegularProtesters | HardcoreProtesters |  |  | Barrels | Gold | SonarKit | WaterPool |

|  |
| --- |
| GameWorld |
| StudentWorld |

**Public member functions**

Note: no pure virtual functions used

* I had several bugs when using the = 0 syntax, and was able to resolve these by defining them as funcName {} or funcName {} = false

1. Actor – used to base all other classes in the actor file
   1. Actor(int imageID, int startX, int startY, Direction dir, double size, unsigned depth);
      1. Constructor used for majority of actors
   2. Actor(StudentWorld\* s, int imageID, int startX, int startY, Direction dir, double size, unsigned int depth);
      1. Constructor used for interactable objects
   3. virtual ~Actor() {};
      1. Simple destructor used for all actors
      2. All ABC’s should have their destructors virtual
   4. virtual void doSomething() {};
      1. virtual function – all actors have a specific doSomething function, this is used to call that in StudentWorld all at once
   5. StudentWorld\* getWorld() { return p; }
      1. Function to return pointer to the StudentWorld, used for all actors to access other actors, such as the tunnelman
   6. virtual bool decreaseHealth(int v) { return false; }
      1. decreases health of any actor
         1. specifically utilized on any humans, but the line “actor->decreaseHealth(2);” in Student world requires it to be declared in Actors
      2. Virtual b/c different actors have different requirements when they lose health
   7. virtual bool canBeSonarRevealed() const { return false; }
      1. sets the status of all items to automatically be unable to be revealed, will be overridden in other classes
      2. virtual – overridden when items can be sonar revealed, such as gold and oil barrels
   8. virtual bool canBeAnnoyed() const { return false; }
      1. automatically sets all actors to be unable to be annoyed
      2. virtual - overriden in the case of humans
   9. virtual bool canPickItems() const { return false; }
      1. automatically sets all actors to be unable to be picked up
      2. virtual – overridden when it’s an interactable or hidden object
   10. virtual void pickGold() {}
       1. virtual function – empty definition that will be overridden by the tunnelmen and actor classes
       2. All 3 final types of humans have a different requirement for picking up gold
   11. virtual bool canActorsPassThrough() const { return true; }
       1. automatically sets all actors to be able to be passed through
       2. virtual - overridden when actors cannot pass through it, such as boulders
   12. virtual void setState(State state) { s = state; }
       1. sets the state of an actor - for example: allows actors to go from a “moving” state to removed from the board
          1. stable = 1,
          2. waiting = 2,
          3. falling = 3,
          4. dead = 0,
          5. pickableByTunnelMan = 4,
          6. pickableByProtesters = 5,
   13. virtual int getHealth() const { return 0; }
       1. will be overridden by the tunnelmen and actor classes, currently automatically returns health of 0
          1. specifically utilized on any humans, but the line “actor->getHealth();” in Student world requires it to be declared in Actors
   14. State getState() { return s; }
       1. returns the state of an actor
   15. void move(Direction);
       1. move in a specified direction
   16. void moveForward();
       1. move in the current direction by 1 block
   17. bool checkBlock(Direction, int, int);
       1. check if the spot has any boundaries, earth, or boulders that may prevent movement
          1. logic errors in this function that cause issues with movement of Regular and Hardcore protesters
          2. stems from mismatch of what each of the following function interprets the input positions to be
   18. bool checkBound(Direction dir, int x, int y);
       1. checks if the moving from spot x,y in Direction dir will move off the board
   19. bool checkEarth(Direction, int, int);
       1. checks if the moving from spot x,y in Direction dir will hit any earth
   20. bool checkBoulder(Direction, int, int);
       1. checks if the moving from spot x,y in Direction dir will hit a boulder
   21. void setCount(int c) { count = c; }
       1. sets number of ticks for an item to last on the board
   22. bool Timeup() { return count <= 0; }
       1. States when to set an actor to dead
   23. void decrementCount() { count--; }
       1. Decrement the lifespan of a disappearing actor each tick
2. Earth
   1. Earth(int startX, int startY);
      1. Constructor for each earth block
3. Human
   1. Human(StudentWorld\* world, int startX, int startY, Direction startDir, int imageID, unsigned int hitPoints);
      1. Constructor for any humans (tunnelmen + protesters)
   2. Virtual int getHealth() const override { return m\_hitpoints; }
      1. returns the number of health points
      2. Overrides the Actor version
   3. void setHealth(int h) { m\_hitpoints = h; }
      1. sets the healthPoints of any human accordingly – does not need overriding in the future
   4. void removeHuman() { m\_hitpoints = 0; }
      1. sets healthpoints to 0, causing them to have their state set to dead
   5. virtual bool decreaseHealth(int v) override { m\_hitpoints -= v; return true; }
      1. overrides the Actor’s decreaseHealth
      2. decreases the hitpoints according to the input
   6. virtual bool canBeAnnoyed() const override { return true; }
      1. overrides the actor’s canBeAnnoyed() to make anything based off human annoyable
   7. bool canPickThingsUp() const { return true; }
      1. Sets all humans to be able to pick up items such as gold
4. Tunnelman
   1. TunnelMan(StudentWorld\* world);
      1. constructor
   2. void doSomething() override;
      1. takes in user input to instruct tunnelman what to do
   3. void moveAndDig(Direction dir);
      1. moves tunnelman in specified direction and removes any earth in that spot
   4. void fireSquirt();
      1. fires a squirt to potentially annoy protesters
   5. void useSonar();
      1. fires sonar to reveal any hidden objects within a set radius
   6. void dropGold();
      1. drops gold at tunnelman’s given location
   7. virtual void pickGold() override { m\_gold++; }
      1. override’s actor’s pickGold to increment player’s gold count
   8. int getSonar() const { return m\_sonar; }
      1. returns the number of sonars available to use
   9. int getSquirts() const { return m\_water; }
      1. returns the number of squirts available to use
   10. int getGold() const { return m\_gold; }
       1. returns the number of gold nuggets available to use
   11. void increaseSonar() { m\_sonar++; }
       1. used when tunnelman picks up sonar to increment number they can use
   12. void increaseSquirts() { m\_water += 5; }
       1. used when tunnelman picks up Waterpool to increment number they can use
   13. int getNextX(Direction dir) const;
       1. returns the next X position for a given direction
   14. int getNextY(Direction dir) const;
       1. returns the next Y position for a given direction
   15. virtual bool decreaseHealth(int v);
       1. override’s human’s decreaseHealth to also check if player has died
5. Protester
   1. Protester(StudentWorld\* p, int imageID, unsigned int health);
      1. Constructor for any protesters
   2. virtual bool decreaseHealth(int v) override;
      1. override’s human’s decreaseHealth to also check if player has died
      2. Plays specified noise and stuns protesters
   3. void pickGold() = 0;
      1. overrides human’s pickGold
   4. void resetRestingTicks(int v) { restingTickCounts = v; }
      1. Resets the number of ticks a protester has to rest before moving
   5. bool breadthFirstSearch(int startX, int startY, int endX, int endY, Direction& finalDir, int& legalSteps);
      1. Uses the breadthFirstSearch to find and recommend fastest path from current location to specified end location
   6. void initialBehaviour(bool& hasFinishedResting);
      1. Contains the first half of a Regular and Hardcore Protester’s doSomething – basic decisions until a hardcore protester checks if they are within M moves of tunnelman
      2. Handles the following cases
         1. Protester is dead
         2. Protester is leaving the oilfield
         3. Protester is yelling at the tunnelman
   7. void actionBehaviour();
      1. Contains the second half of a Regular and Hardcore Protester’s doSomething – basic decisions after hardcore protester checks if they are within M moves of tunnelman
      2. Handles the following decisions
         1. Chase tunnelman if they are within direct sight
         2. General movements
            1. Moving in current direction for n squares
            2. Selecting new available directions
            3. Perpendicular turns
   8. void incrementShoutTickCount() { shoutTickCount++; }
      1. used to increment shoutTickCount (private member variable)
   9. void incrementPerpTurn() { perpTurn++; }
      1. used to increment the number of ticks since a forced perpendicular turn has been taken
6. RegularProtester
   1. RegularProtester(StudentWorld\* p);
      1. constructor
   2. virtual void pickGold() override;
      1. calls and overrides protester’s pickGold
         1. Sends regularprotesters off the screen and increments score
   3. virtual bool decreaseHealth(int v) override;
      1. overrides and calls protester’s decreaseHealth
      2. increases score
   4. virtual void doSomething() override;
      1. override’s actor’s do something
      2. calls the initialBehaviour and actionBehaviour functions to move protester
7. HardcoreProtester
   1. HardcoreProtester(StudentWorld\* p);
   2. virtual void pickGold() override;
      1. calls and overrides protester’s pickGold
         1. prevents protesters from moving for set number of turns and increments score
   3. virtual bool decreaseHealth(int v) override;
      1. overrides and calls protester’s decreaseHealth
      2. increases score
   4. virtual void doSomething() override;
      1. override’s actor’s do something
      2. calls the initialBehaviour and actionBehaviour functions to move protester
         1. if tunnelman is close enough, chase player
8. Boulders
   1. Boulders(StudentWorld\* pointer, int startX, int startY);
      1. constructor
   2. virtual void doSomething();
      1. override’s actor’s doSomething
      2. Consider’s boulder’s 3 positions
         1. Stable – no movement
         2. Falling
            1. Determines whether it hits anything that causes it to disappear or hurt any humans
         3. Dead – will be removed
   3. virtual bool canActorsPassThrough() const { return false; }
      1. overrides actor’s canActorsPassThrough() to make it unable to pass through
9. Squirt
   1. Squirt(StudentWorld\* pointer, int x, int y, Direction d);
      1. constructor
   2. virtual void doSomething();
      1. override’s actor’s doSomething
      2. specifies what to do in a tick
         1. specifies what to do if squirt runs out of time or hits specific actors
10. Interactable
    1. Interactable(StudentWorld\* world, int startX, int startY, int imageID, bool visible, State s): Actor(world, imageID, startX, startY, right, 1.0, 2)
    2. double distToTunnelMan()
11. HiddenObjects
    1. HiddenObjects(StudentWorld\* world, int x, int y, int imageID, bool visible, State s) : Interactable(world, x, y, imageID, visible, s) {}
       1. constructor
    2. bool wasDiscovered();
       1. sets discovered to true if tunnelman is in proximity
    3. virtual bool canBeSonarRevealed() const override { return true; }
       1. override’s actor’s canBeSonarRevealed() to allow items to be revealed by sonar
    4. bool isHidden() { return m\_isHidden; }
       1. sets every hidden object to isHidden
    5. void setStatus(bool status) { m\_isHidden = status; }
       1. sets hiddenObjects to be hidden
12. Barrel
    1. Barrel(StudentWorld\* p, int x, int y);
       1. constructor
    2. virtual void doSomething();
       1. override’s actor’s doSomething
       2. specifies what to do in a tick
          1. specifies what to do if tunnelman picks it up
13. Gold
    1. Gold(StudentWorld\* p, int x, int y, State s, bool b);
       1. constructor
    2. virtual void doSomething();
       1. override’s actor’s doSomething
          1. specifies what to do if tunnelman or protester picks it up
14. Goodies
    1. Goodies(StudentWorld\* p, int x, int y, int imageID);
       1. constructor
    2. virtual void doSomething();
       1. override’s actor’s doSomething
       2. specifies what to do in a tick
          1. specifies what to do if tunnelman picks it up
15. SonarKit
    1. SonarKit(StudentWorld\* p, int x, int y);
       1. Constructor
    2. virtual void doSomething();
       1. override’s actor’s doSomething
       2. specifies what to do in a tick
          1. specifies what to do if tunnelman picks it up
16. WaterPool
    1. WaterPool(StudentWorld\* p, int x, int y);
       1. Constructor
    2. virtual void doSomething();
       1. override’s actor’s doSomething
       2. specifies what to do in a tick
          1. specifies what to do if tunnelman picks it up
17. StudentWorld
    1. StudentWorld(std::string assetDir);
       1. constructor
    2. virtual ~StudentWorld();
       1. destructor
    3. virtual int init();
       1. initialize the gameboard and add all initial actors
    4. virtual int move();
       1. move everything via doSomething each tick, add new elements as needed
    5. virtual void cleanUp();
       1. deletes all actors and returns the memory to the computer
    6. Earth\* getGameboard(int x, int y) const {return gameboard[x][y];}
       1. Gets the value the gameboard at a specified coordinate to determine if earth is there
    7. void RemoveEarth(int x, int y);
       1. deletes earth at a specified coordinate
    8. void CreateSquirt(int x, int y, GraphObject::Direction dir);
       1. launch a squirt from tunnelman
    9. int getTunnelManPosX();
       1. returns the tunnelman’s x coordinate
    10. int getTunnelManPosY();
        1. returns the tunnelman’s y coordinate
    11. int getTunnelManHealth();
        1. returns the tunnelman’s health
    12. void decreaseBarrels();
        1. decrease m\_barrels
    13. void decreaseNumProtesters() { numProtesters--;} ;
        1. decreases the number of protesters
    14. void increaseTunnelManGold();
        1. use pointer to increase tunnelman’s gold
    15. void increaseSonar();
        1. use pointer to increase tunnelman’s sonar
    16. void increaseSquirts();
        1. use pointer to increase tunnelman’s water squirts
    17. void hitTunnelMan();
        1. decrease tunnelman’s health
    18. bool checkBoulder(int, int, GraphObject::Direction);
        1. check if boulder is adjacent to a specific coordinate
    19. void revealHiddenObjects(int, int);
        1. updates hiddenobject’s status to visible if within proximity
    20. void dropGold(int, int);
        1. creates a new gold actor with status pickableByProtesters
    21. bool goldPickedByProtesters(int x, int y);
        1. instructs protesters to pick up gold if within proximity
    22. bool annoyProtesters(int x, int y);
        1. protesters have been hit by water, decrease their health
    23. bool checkForBoulders(int x1, int x2, int y1, int y2);
        1. checks for boulders within a specified area
    24. bool checkEarthArea(int, int, int, int);
        1. checks for earth within a specified area
    25. void hitProtesters(int, int);
        1. kills protesters if hit by boulder
    26. bool checkIfEmpty(int x, int y);
        1. checks if nothing is at a specific coordinate

**Known bugs**

1. checkBlocks
   1. the check relies on 3 different functions: checkBoundary, checkEarth, and checkBoulders. I think that they each utilize their inputs of dir, x, and y differently, so some of the functions check if the spot adjacent (dir) to x,y is blocked, while others check if that exact spot is. This causes logic issues with this function, in particular in the protester’s automated movements in my actionBehaviour, which in turn impacts how my RegularProtester and HardcoreProtester move
2. Protester::actionBehaviour
   1. As the check for blocks has a logic error I was unable to resolve in time, the actionBehaviour function is unable to properly determine if actors are blocked. Depending on the parameters I put in, the protester would think they were blocked too early, or would not realize they were blocked when reaching a block of earth or boundary. This was inherited into RegularProtester::doSomething() and HardcoreProtester::doSomething()

**Assumptions**

It should be noted that I was unable to run the sample code on my laptop, and all assumptions are made via Professor Ambrosio’s video demo and the Project 4 guidelines.

1. Direct eyesight of a protester requires the exact same x or y coordinate
   1. This means that if the head of the tunnelman shares the same coordinate of as the foot of the protester, the protester will not chase. If the player is only at the exact same x or y coordinate as the protester during a resting tick, the player will not be chased. However after rewatching and going very slowly through the example, video, I realized otherwise, and changed this in my code.
2. Order of operations of digging
   1. It does not matter whether the move or dig occurs first, so therefore, I move before digging the earth
3. Perpendicular turns
   1. Perpendicular turns that are not caused by the 200 tick perpendicular turn requirement do not count towards this, and do not reset the perpendicular turn tick counter
4. Items
   1. There is no limit to the number of squirts, gold, or sonar charges a tunnelman can collect at once, so long as they have properly spawned and been collected from the board
5. Boulders
   1. If a boulder falls, and in the process overlaps 1-2 pixels with another boulder, it will be set as dead

Class Testing

The following classes are the ones I tested. Any others were the abstracted bases of these classes, and therefore, tested through my classes via inheritance:

1. Actor
   1. The majority of its functionality is tested through its inheriting classes. In this section, I will be focusing on the checkBlock and the 3 functions it entails
      1. checkBoundary: To guarantee that my character could not go out of bounds, I repeatedly tried to move the tunnelman out of bounds, and when it did not react or crash, I confirmed that this function works as intended.
      2. checkBoulder: To check this, I moved my character into boulders from all sides to ensure that all 4 directions do not allow the character to enter a boulder
      3. checkEarth: I dug away part of the upper half of the gameboard. In doing so, I allowed the protesters to have more space to walk around, with occasional chunks of earth in the way. If it did not proceed into these, it worked properly
   2. However combining these into checkBlock appears to have a bug. This was specified above, but I realized I may have used the x, y, and dir inputs differently. Some of these functions interpret it as the current spot, while others check the next spot. Therefore, this causes issues when combined for the walking aspects of protesters.
2. Tunnelmen
   1. To confirm the tunnelman functions as necessary, I tested two components of the tunnelman. The rest were related to other classes, which are tested below
      1. Movement
         1. I confirmed that all the required user inputs (keyPressed) worked, by individually checking each one
         2. I confirmed that my tunnelman moved for each input, and if the tunnelman turns, that counts as the tick, and does not move immediately
         3. I confirmed that by moving in to any area with dirt, it would be dug out and removed
      2. checkBlocks
         1. checkBoundary
            1. I confirmed the boundaries worked by walking my tunnelman into each corner from both directions, ensuring that the tunnelman could not go off the board or cause any memory allocation errors
         2. checkBoulder
            1. To confirm that checkBoulder had the right boundaries, I walked into the boulder from each direction, at different x and y coordinates relative to the boulder. Initially, my boundaries were off, allowing the tunnelman to enter the boulder, but this was resolved by finetuning the checkBoulder’s logic statements’ boundaries. Once I was able to remove each pixel 1 by 1 without entering the boulder, I had confirmed that the function worked as necessary
3. RegularProtesters
   1. To test that protesters worked as necessary, I broke it into several components
      1. Spawning
         1. I ensured that all of the protesters spawned in the upper right corner, and that they were able to respawn if a protester had been eliminated
      2. Initial behaviour
         1. I ensured that the protesters were removed if at 60, 60 and in the state “dead”, by squirting a protester who was unable to move from the initial position
         2. I ensured that the call to breadthfirstsearch worked by squirting a protester until they immediately went back to the spawn spot
         3. To test their shouting, I stood in all 16 surrounding positions to a protester to confirm when the shouting would occur
      3. Action behaviour
         1. Chasing a player: to confirm that a protester would chase the tunnelman, I would dig paths underneath them to confirm that they would turn towards me. In doing so, I went in a loop around the protester to confirm that it would chase me regardless of the direction they are facing
         2. Selecting a new direction to move: To observe that they would select different directions after running out of numSquaresToMoveInCurrentDirection, I utilized print statements and breakpoints to confirm that the counter had run out, and that new directions were being selected. I used the same method for the perpendicularTurns requirement, and confirmed this worked
         3. Blocked directions
            1. By using a breakpoint on both the check for whether the direction was blocked and the inside lines, I was able to determine that the if statement was successfully reached, but there was a logic error that did not return the proper value all the time. Due to time constraints, I decided to submit the code with the version that had less issues with the checkBlock function, but still results in a bug at this step.
      4. Breadthfirstsearch
         1. To ensure that the breadthfirstsearch worked, I would lure protesters to various parts of the maze that I had dug, with various widths, and confirmed that upon eliminating them, they were sent back to the spawn position to eliminate them.
4. HardcoreProtesters
   1. To test that protesters worked as necessary, I broke it into several components
      1. Spawning
         1. I made sure that hardcoreprotesters did not spawn every time as the protester added. Aside from that, the remainder of the functionality was the same as a regularProtester (with the exception of hitpoints, which I tested by squirting one more).
      2. Regular movements
         1. These were tested in the regular protester, and therefore skipped for hardcore protesters
      3. Breadthfirstsearch
         1. To ensure that the breadthfirstsearch worked on locations aside from the spawn location, I also would lure the hardcoreprotesters through the maze, making them take perpendicular turns and going through u-shaped tunnels
5. Earth
   1. This was a relatively simple class to test. In part 1 of the game, I just had to make sure that when the game was initialized, all the required positions were filled in with Earth blocks. By running it multiple times, I confirmed the earth actors were populated properly. Additionally, to confirm that earth was dug properly, I made sure have my player dig through the entire board, to ensure that all pieces could be removed. Once boulders were implemented, I made sure that upon initializing, there would be no earth in the locations that boulders were spawned.
6. Boulders
   1. There were 4 pieces to testing the boulders.
      1. First, I ran the game multiple times to ensure that boulders are spawned in the proper areas (not the top row or center column), and not on top of another object. Additionally, I made sure that all earth in that spot was removed
      2. Next, I made sure that the boulder was stable and did not move/get removed unless required. In the beginning of my testing, the boulder would be removed after several seconds in the game. I realized this was because there was a bug in the checkEarth function, which caused it assume no earth was below, removing it from the game. Once this was fixed, I made sure to test how the boulder would react when 3, 2, or 1 small blocks of earth were supporting the boulder.
      3. Once the boulder was stable, I tested how the boulder would fall. I did so by removing the earth below it, and seeing how it would fall and react to hitting protesters, tunnelmen, or smaller blocks of earth that stick out.
      4. Finally, I tested when it would directly hit another boulder (exact x coordinates), hitting a protester, hitting myself, and hitting a boundary or earth. Each of these could be done by simple tunnelman movements except for the first, in which I manually set the spawns of the boulders to test this.
7. Barrels
   1. There were 2 pieces to testing the oil barrels.
      1. First, the spawning of them on the board. I first set them to always be visible, to ensure that they were spawning in proper locations. Then I set them to invisible so that as I approached them, I knew they would become visible.
      2. Next, I tested the tunnelman’s interactions with them. If a tunnelman picked it up, it would disappear from the board increase the score, and decrement the number of barrels left to pick up. If we removed all barrels, I confirmed the play to the next level. This was done in a board without any protesters added.
8. Gold
   1. There were 4 pieces to testing the gold.
      1. First, the spawning of them on the board. I first set them to always be visible, to ensure that they were spawning in proper locations. Then I set them to invisible so that as I approached them, I knew they would become visible.
      2. Next, I tested the tunnelman’s interactions with them. If a tunnelman picked it up, it would disappear from the board and increase the score. I also tested dropping it in specific places, and made sure it sank into the earth after the given time.
      3. After that, I came to the protesters. I made sure to drop it in front of the RegularProtesters and saw that their status changed to “falling”, or returning to the spawn point to leave the field, and the score was properly incremented. To skip the pick up stage of gold, I set m\_gold to 100 for this testing.
      4. Finally that, I came to the hardcore protesters. I made sure to drop it in front of the HardcoreProtesters and saw the score increment, and that they paused, but did not return to the spawn point. To skip the pick up stage of gold, I set m\_gold to 100 for this testing.
9. SonarKit
   1. There were 2 pieces to testing the sonar kits.
      1. First, the spawning of them on the board. I made sure that the spawning of them took place in their specified location, and watched as it appeared, collected, and used it several times in 1 game.
      2. Next, I tested the tunnelman’s use of a sonarkit. I went around the board digging, and repeatedly used the sonar to reveal nearby items. To skip the pick up stage of sonar, I set m\_sonar to 100 for this testing.
10. Waterpool
    1. To test the waterpool, I dug random parts of the board and watched the waterpools spawn. I made sure to fix bugs of when the waterpool spawned on top of squares that had earth in them, and made sure to watch that waterpools sank back into the earth. Finally, to test it, I had to ensure that the waterpool would increase the number of squirts I had by collecting a waterpool and watching it disappear from the board.
11. StudentWorld
    1. There are several parts to testing the code for the game. Through the process of testing the above functions, a majority of it was tested. Here are the specific steps I took to ensure that StudentWorld was functioning properly.
       1. Init: I would run the code to make sure everything spawned properly, and that no crashes occurred. Initially, prior to part 1’s submission, I had several issues with this, and thus, required to comment parts of it out prior to submission.
       2. Move: I would play through games to make sure every relevant actor would initiate their doSomething, breakpointing on the line “for (auto p : actorsList) { p->doSomething(); }. Additionally, I made sure to breakpoint on the removals to ensure that the memory was returned to the computer
       3. Set Display Text + cleanup: I played through several levels 5 times. This confirmed that no memory issues or bad execs occurred, while I watched that the score and item counts were updated properly

Throughout my testing, I used a significant number of cout statements and breakpoints to ensure the correct outputs and actions, in particular in the Actor::check\_\_\_\_\_ functions, TunnelMan::moveAndDig(), and protester::initialBevhaviour and protester::actionBehaviour functions.