1)

1. StudentWorld:
2. Three method functions: init(), move(), and cleanUp(). They are all declared virtual. The init() function initialize the private members of the class and create actors including Earth, the TunnelMan, boulders, oil barrels, and gold nuggets. The move() function displays the line of stats and updates it every tick. It also determines whether to add a protester or a goodie if the time limits are reached. The major purpose of the function is to allow all actors to do something if they can. In this process, the function also checks the status of the player and other actors. If the player is dead, the function immediately returns the information. If other actors are dead, it deletes them. If the level is finished, the function returns the level is finished. Otherwise, the function returns and let the game continue. The cleanUp() function deletes all elements of the game, including the TunnelMan, E arth, and all other actors stored in the vector.

All other public member functions of the class are non-virtual functions because StudentWorld does not have any derived classes.

1. void getX and void getY are simple accessors that return the location of an actor. It is declared in this class because all actors need to have a function to report their locations.
2. void addActor function adds a new actor to the actors vector with the push\_back function. It is declared in this class because StudentWorld manages all actors and store them in a vector.
3. bool digEarth is a boolean function that returns whether the player is digging earth. The function checks if the player overlaps with the Earth elements. If they do, the function delete the Earth elements at the player’s location and returns true. Otherwise, it does nothing to the Earth and returns false. It is declared in this class because StudentWorld manages the Earth elements.
4. void decreaseBarrels decrements the number of oil barrels in the oil field by 1. It is declared in this class because StudentWorld manages the oil barrels.
5. void increaseGold gives the player a gold nugget, so it can be used to bribe the protesters later by the TunnelMan. It is declared in this class because it can be called by function members of the Gold class.
6. void decreaseGold decrements the number of gold nuggets in the oil field by 1. It is declared in this class because StudentWorld manages the gold nuggets.
7. void increaseSonar gives the player a sonar, so it can be later used to detect the gold nuggets and oil barrels around the TunnelMan. It is declared in this class because it can be called by function members of the Sonar class.
8. void increaseWater gives the player 5 water squirts. It is declared in this class because it can be called by function members of the Water class.
9. void dropAGold creates a new Gold actor at the current location of the player. It is declared in this class because StudentWorld manages the gold nuggets.
10. void useSonar reveals all gold and oil barrels in a radius of 12 from the player. It is declared in this class because StudentWorld manages all actors of the game and can therefore change their state of visibility.
11. void annoyPlayer calls the annoyed function in the TunnelMan class and is called by the protesters to decrement the player’s health. It is declared in this class because it can be called the Protester class.
12. void annoyPlayerOrProtester calls the annoyed function in either the TunnelMan class or the Protester class. It is declared in this class because StudentWorld manages all actors of the game and can therefore change the moving actors’ health levels.
13. void decreaseProtester decrements the number of protesters in the oil field by 1. It is declared in this class because StudentWorld manages the protesters.
14. Actor\* getPlayer returns the private member of the class, m\_player. It is declared in this class to allow other classes to get the information of the player, such as location.
15. bool revealObjects determines if an object should be visible in the oil field. It checks whether any object is within the required radius from the actor. If it is within the distance, the function returns true; otherwise, it returns false. It is declared in this class because StudentWorld manages all actors of the game and can therefore change their state of visibility.
16. bool canMove determines if an actor can move to the new location. It is declared in this class because StudentWorld manages all actors of the game and the entire oil field.
17. bool canMoveInDir determines if an actor and move in the picked direction. It is declared in this class because StudentWorld manages all actors of the game and the entire oil field.
18. bool finishLevel checks if all oil barrels of this level are picked up. If yes, the function returns true. It is declared in this class because StudentWorld manages the oil barrels.
19. bool aroundPlayer determines if an object is near the player. It checks whether any actors are within the required radius from the player. If yes, the function returns true. It is declared in this class because StudentWorld manages all actors including oil barrels, gold nuggets, and the TunnelMan.
20. bool bribePortester determines if a gold should be picked up by the protester. It checks whether any gold nuggets dropped by the player are within the required radius from the protesters. If yes, the protester is bribed, and the functions calls the addGold function in the Protester class so that the protester can react accordingly.
21. bool noEarth checks the 4\*4 square of the current location. If all pointers in the m\_earth member are null, the function returns true; otherwise, it returns false. It is declared in this class because StudentWorld manages all Earth elements.
22. bool addGoodie determines if a goodie needs to be added in this tick. The function uses the required probability and random integers to decide. It is declared in this class because StudentWorld manages the goodies.
23. bool isValidForPool checks if the location is good for creating a water pool actor. A water pool can only appear at a location with no earth in the 4\*4 square. It is declared in this class because StudentWorld manages all Earth elements and can thus check their locations.
24. int addSonarOrPool determines if a sonar or a water pool should be added. The function uses random integers to decide. There is a 1/5 chance a sonar is to be added, so the function returns 0. In all other cases, the function returns 1, indicating that a water pool should be added instead. It is declared in this class because StudentWorld manages the goodies.
25. int getGoodieTime returns how long a goodie should last. It is declared in this class because StudentWorld manages the visibility and state of all goodies.
26. double distance calculates the distance between two passed in locations. It is declared in this class because all actor classes can use the function to determine the distance between them and other actos.
27. bool facingPlayer helps the protesters determine if they are facing the player. The function is declared in the StudentWorld class because StudentWorld has access to the TunnelMan class and can thus get the player’s location.
28. void markMaze updates the maze with integers, which indicate the steps away from the destination. It is important for the protesters exiting the oil field and harcore protesters hunting the player. It is declared in this class because StudentWorld manages the maze.
29. Direction findDirForward returns the optimal direction for the protesters moving one step closer to the destination. It is declared in this class because StudentWorld manages the maze.
30. Direction findExitDirection returns the optimal direction for the protesters moving one step closer to the destination(60,60) to exit the oil field. It is declared in this class because StudentWorld manages the maze.
31. Direction findDirectionToPlayer returns the optimal direction for the protesters moving one step closer to the player, if the protester is with 16+level\*2 legal steps from the player. . It is declared in this class because StudentWorld manages the maze, the player, and the protesters.
32. Actor:
33. StudentWorld\* getWorld is an accessor that returns the m\_world. It is declared in the Actor class because the actors will need to use the public functions in the StudentWorld class.
34. void dead sets an actor’s state to dead, so it will be later deleted from the game. It is declared in the Actor class because all actors can die.
35. bool isAlive returns if an actor is alive or dead. It is declared in the Actor class because it’s a state of all actors.
36. virtual bool isTunnelMan is a virtual function that returns if an actor is TunnelMan. It is virtual because all actors should tell the game if they are TunnelMan or no.
37. virtual bool isProtester is a virtual function that returns if an actor is a protester. It is virtual because all actors should tell the game if they are protesters or no.
38. virtual bool block is a virtual function that returns if an actor blocks other actors. It is virtual because all actors should tell the game if they block other actors or no.
39. virtual void doSomething is a pure virtual function because all actors should have their own special version of doSomething.
40. MovingActor:
41. int getHealth is an accessor that returns the health of an actor that can move because all moving actors can be annoyed, and their health levels can be decremented.
42. virtual void addGold is a pure virtual function because all actors should have their own special version of addGold.
43. virtual bool annoyed is a virtual function that implements the basics behaviors of all moving actors when their health is lower than or equal to 0. It is declared here because all moving actors have these common behaviors.
44. TunnelMan:
45. virtual void doSomething defines the TunnalMan’s special version of doSomething. The function checks if the player is still alive. If it’s not, the player doesn’t do anything. Otherwise, the player can move, dig earth, squirt water, drop a gold, use sonar, or escape.
46. virtual bool isTunnelMan returns true because the actor is of type TunnelMan. This is a special version of the function because all other actors use the base function, which returns false.
47. virtual void addGold is a special version of addGold that only belongs to the player. When the player picks up a gold, it simply gets a gold nugget.
48. void loseGold decrements the number of golds the player has by 1. It is declared here because only the TunnelMan can get and use a gold.
49. void addSonar increments the number of sonars the player has by 1. It is declared here because only the TunnelMan can get and use a sonar.
50. void addWater addSonar increments the number of squirts the player has by 1. It is declared here because only the TunnelMan can get and use a water squirt.
51. int getGold is an accessor that returns the number of golds the player has.
52. int getSonar is an accessor that returns the number of sonars the player has.
53. int getWater is an accessor that returns the number of squirts the player has.
54. void loseSonar decrements the number of sonars the player has by 1. It is declared here because only the TunnelMan can get and use a sonar.
55. void shootWater allows the player to use water squirt in the direction it’s facing. The function plays the sound of using water squirt and decrement the number of squirts the player has by 1. It is declared in the TunnelMan class because only the player can use a squirt.
56. Protester:
57. virtual void doSomething is a special version of doSomething for protesters. It checks if the protester is still alive. If no, the protester does nothing. It also checks if the protester is in the leaving state. If it is, do nothing but move one step closer to 60, 60 every tick. Otherwise, the function checks if it is the tick for the protester to move. If it is not, the protester waits. If yes, it checks if the protester is near and facing the player. If it has not shouted at the player in the last 15 ticks, it annoys the player. Otherwise, the function checks if the player is in the line of sight of the protester. If yes, the protester changes direction to face the player and move towards it. Else if the protester is a hardcore protester and is within 16+2\*level steps to the player. If it is, it moves towards the player. If all above are not correct, the protester may continue moving in the current direction if required to. Otherwise, choose a new direction to move. If the protester has not move perpendicularly in the last non-resting 200 ticks, the function let it choose a viable or random perpendicular direction to move.
58. int numSquaresToMoveInCurrentDirection returns a random number between 8 and 60 inclusive. This is the function that helps the protesters determine how many steps they need to move in the current direction.
59. Direction chooseRandDir allows the protester to choose a random direction to move in the next non-resting tick.
60. Direction chooseRandPerp allows the protester to choose a viable or random perpendicular direction to move in the next non-resting tick.
61. virtual bool annoyed is a special version of annoyed. When annoyed, the protesters not only decrement their health levels but are also stunned. If they are annoyed to death, they also increase the player’s score.
62. virtual void addGold() is a special version of addGold for regular protesters. If the regular protesters get a gold, the function plays the sound of protesters getting a gold and they immediate leave the oil field.
63. virtual bool isProtester is a special version of isProtester that returns true. All other actors return false.
64. void resetRestTicks reset the rest ticks for the protester when either the rest ticks are depleted, or the protester is stunned.
65. virtual bool isHardcore is a virtual function that returns if an actor is a hardcore protester. It is virtual because all protesters should tell the game if they are hardcore or no.
66. void sensePlayer is a function that determines if a protester is hardcore and if it needs to hunt the player. The function calculates the number of steps of the protester away from the player. If the hardcore protester is within the required number of steps, the function allows the hardcore player to chase and hunt the player.
67. Direction playerInSight determines if the player is in the line of sight of the protester. If it is, the function returns the direction of the player relative to the protester. If it is not, the function returns none.
68. HardcoreProtester
69. virtual void addGold is a special version of addGold for only hardcore protesters. When a hardcore protester gets a gold, it merely stops to stare at the gold and does not leave the oil field like regular protesters do.
70. virtual bool isHardcore is a special version of isHardcore for only hardcore protesters. The function returns true, while all other versions return false.
71. Earth:
72. virtual void doSomething is a special version of doSomething for only earth. The function is empty because earth does not do anything.
73. Water:
74. virtual void doSomething is a special version of doSomething for only water. It checks if it has travelled 4 square. If yes, the function sets it to dead. Otherwise, it continues travelling in the direction.
75. Boulders:
76. virtual void doSomething is a special version of doSomething for only boulders. The function checks if the boulder is in a stable state. If it is, it does not do anything. If it is not, it waits for 30 ticks and starts falling. If it encounters the player or any protesters, it annoys them. If it encounters earth or other boulders, it disappears and die immediately.
77. bool isStable checks if there is any earth below the boulder. If there is no earth below it, the function returns false.
78. virtual bool block is a special version of block for only boulders. Boulders are the only objects that block the other actors, so this function returns true.
79. Goodies:
80. virtual void doSomething is a special version of doSomething for only goodies. The function checks if the goodie is still alive. If it is not, it does nothing. If it is, the function checks if the goodie is to be revealed or picked up. If both are not true, it stays still and does nothing.
81. OilBarrels:

no special functions.

1. Gold:
2. virtual void doSomething is a special version of doSomething for only golds. The function checks if the goodie is still alive. If it is not, it does nothing. If it is, the function checks if the goodie is in a permanent state. If yes, the function checks if the gold should be revealed or picked up. If both are not true, it stays still and does nothing. if the gold is in a temporary state, the function checks if there are any protesters nearby and can be bribed. If yes, the gold bribes the protester.
3. Sonar:
4. virtual void doSomething is a special version of doSomething for only sonars. The function checks if the goodie is still alive. If it is not, it does nothing. If its lifetime limit is reached, the function makes it dead. Otherwise, the function checks if the goodie is picked up by the player.
5. WaterPool:
6. virtual void doSomething is a special version of doSomething for only water pools. The function checks if the goodie is still alive. If it is not, it does nothing. If its lifetime limit is reached, the function makes it dead. Otherwise, the function checks if the goodie is picked up by the player.

2) I implemented all functionalities.

3) a. It wasn’t clear in the spec that if the goodies can overlap (water pools and sonars) , so I decided to wirte a function to prevent them from overlapping as I observed in the sample executable given.

4)

1. StudentWorld: To test the class, I ran the entire program to see if all objects are created correctly and the moving actors could move. I also checked if the player could dig earth. I tried to press escape and see if the level restarts and lives are decremented. I tried to kill the player with protesters and boulders. The level correctly restarted, and the lives are decremented by one. I also tried to finish the level by picking up all oil barrels to check if the game could advance to the next level. I played the game until level 8 to see if the numbers of oil barrels, gold nuggets, and protesters are correct. I made sure if the player could pick objects and if the protesters could pick up gold nuggets. I also paid attention to if an object reveals itself when the player gets close to it but not close enough to pick it up. I made sure when the player picks up an goodie or is annoyed, the stats are changed correctly.

2. Actor: To test this class, I made sure to see if the members could use the public member functions of the StudentWorld class. Since this class does not have many special versions of functions, I made more tests on its derived classes.

3. MovingActor: Since this class does not have many special versions of functions, I made more tests on its derived classes.

4. TunnelMan: To test the TunnelMan class, I first checked if the TunnelMan can move as I press the direction keys. I also checked if it follows my other instructions, such as using sonar, using squirt, and escape. I made sure that the stats changed correctly when I use sonars and water squirts. I checked if the TunnelMan can dig earth and pickup goodies, and when they pick up goodies, the numbers of the goodies the player had increased. I chekced when the player was annoyed by the protester or bonked by a boulder, its health levels were decreased. When its health reached 0, I made sure it was set to dead.

5. Protester: To test the Protester class, I paid attention to if they are created correctly since there are two types of protesters. I tried with each type to see how many water squirts should be used to give them up. The hardcore protesters needed 10 and the regular protesters needed 3. This means that they have the correct initial health. I also checked if they did not move in every tick but in every three ticks. I made sure they randomly changed their directions after moving in the same direction for 8 to 60 steps, and sometimes they move perpendicularly if they had not done so in the last 200 non-resting ticks. I checked if their health reached 0, they immediately left the oil field in the optimal path with no resting ticks. I tried to drop a gold and see if the protesters pick up the gold: the regular protesters should immediately leave the oil field, while the hardcore protesters should stop and be stunned. To see if they change their directions to the player when it is in their sights, I moved the player into their sights. They immediately changed the directions to face the player and moved towards the TunnelMan.

6. HardcoreProtester: There are only two differences between hardcore protesters and regular protesters. First, when a hardcore protester picks up a gold, it merely stares at it and then starts to move again after some time. Second, it can sense the player’s cell phone signal and move towards it when it is within 16+level\*2 steps from the player. I tried to move the player close to a hardcore protester with earth between them (the sight is blocked). The hardcore protester immediately started to move towards the player.

7. Earth: To test this class, I made sure that the TunnelMan dig the earth. After digging, the earth disappeared. I also made sure that the earth blocks the movement of boulders and protesters.

8. Water: To test the water class, I instructed the TunnelMan to squirt water, and sound was played. I made sure that water appears and travels 4 squares in the direction that the player is facing before it was dead. I also checked if water disappears when it hits earth or a protester. I made sure that water annoys the protesters by 2 when it hits them.

9. Boulders: When the game just started, I made sure that the boulders did not overlap with any earth and were in a stable state. Then, I played the TunnelMan to dig all earth below the boulders. I observed that the boulders waited 30 ticks and started to fall with the sound of falling. When they hit earth or other boulders, they immediately disappeared. To check if they could kill the protester or the player, I let them fall onto both types of protesters and the player. The protesters left the oil field, while the player’s lives were decremented and the level restarted. I also made sure that when the boulders kill the protesters, the player’s score is incremented by 500.

10. Goodies: I made sure when the player gets very close, they are picked up by the player. If they are close to the player but not close enough to be picked up, they became visible. Otherwise, they stay still and invisible.

11. OilBarrels: I made sure when the player gets very close, they are picked up by the player. When they are picked up, the game plays the sound of player getting an oil barrel. I also checked if the player’s score is incremented by 1000. If they are close to the player but not close enough to be picked up, they became visible. Otherwise, they stay still and invisible.

12. Gold: If a gold is in permanent state, I made sure when the player gets very close, it is picked up by the player. If it is close to the player but not close enough to be picked up, it becomes visible. Otherwise, it stays still and invisible. If the gold is in temporary state, I checked if it can be picked up by the protester and the sound of protesters getting a gold is played. If it is not picked up after a certain time, it disappears and can no longer be picked up by the protesters.

13. Sonar: I made sure they are only created at 0, 60. I checked if a sonar can be picked up and used by the player. If it is not picked up after a certain time, it disappears and can no longer be picked up by the player.

14. WaterPool: I made sure they are only created at locations without any earth. I checked if a water pool can be picked up and used by the player. If it is not picked up after a certain time, it disappears and can no longer be picked up by the player.