**FrackMan Report**

*By Nina Chikanov*

**PART ONE: HIGH LEVEL DESCRIPTION OF PUBLIC MEMBER FUNCTIONS**

**Actor**

1. virtual void doSomething()
   1. Made it virtual because every single class defines it’s own doSomething method
2. char getName()
   1. Gave every class derived from Actor a name so I could easily identify them later
3. bool validStraight2Frack()
   1. Checks if the Actor can move in a straight line to FrackMan without running into any boulders or dirt
4. bool isAlive()
   1. The actor is alive if their hit points are greater than 0
   2. This returns the status of whether or not they are alive
5. void setHitPts()
   1. If hitPoints need to be changed at any time, use this function to decrement them
6. int hp() const
   1. Returns how many hit points an actor has at a given moment in time
7. virtual bool annoy()
   1. Made this virtual because all the Agents (FrackMan/ Protesters) can be annoyed and will define their own functionality accordingly
8. virtual bool canPickThingsUp()
   1. Defines if an Actor can pick up other objects/ actors
   2. Virtual because most actors cannot pick up others, however Agents can, so they will change this later.
9. virtual bool canDigThroughDirt()
   1. Defines if an actor can dig through dirt
   2. Virtual because not all actors can dig through dirt - only FrackMan
10. StudentWorld\* getWorld()
    1. Returns a pointer to studentWorld so Actors can access this functionality
11. virtual bool canActorPassThroughMe()
    1. Defines if other actors can pass through this object
    2. Virtual because not all objects can be passed through (ex. Boulder)
12. bool facingTowardFrackMan()
    1. Returns whether or not an actor is facing towards the FrackMan - direct line of sight
13. Direction lineOfSightToFrackMan(Actor\* a)
    1. Determines if FrackMan is in a direct line of sight and returns the direction he is relative to the other actors using the function

**Dirt : public Actor**

1. Has no public functionality - everything it does is taken care of in StudentWorld

**Squirt : public Actor**

1. virtual void doSomething()
   1. If Squirt is close to a Protester, it will annoy it
   2. While it hasn’t travelled it’s full distance, it’ll check to see if it can move the full distance that it’s fired (aka it won’t run into a Boulder or dirt)

**Agent : public Actor**

1. virtual void addGold() = 0;
   1. Didn’t end up doing this, but the idea was to make this virtual because all the Agents have addGold() things they would do
   2. Also I made this pure virtual since Agents can’t add Gold, but the classes derived from it can
2. virtual bool annoy ()
   1. All actors can be annoyed
3. virtual bool canPickThingsUp()
   1. ALL agents can pick things up, so set this accordingly

**FrackMan: public Agent**

1. virtual void doSomething()
   1. Receives input from the user and moves in the direction the player wants it to do
   2. Checks to see if it’s valid to move to that destination
2. virtual bool annoy()
   1. Decrements hit points by 2
   2. Checks if the player dies as a result of being annoyed
3. virtual bool canDigThroughDirt()
   1. True! Can dig through dirt!
4. void addSonar()
   1. update Sonar count
5. void addWater()
   1. update Water count
6. void useSquirt()
   1. Decrement water count
7. void updateGold()
   1. Update gold count
8. int numSquirts()
   1. Return how many waters he has
9. int numSonar()
   1. Returns the number of sonars he has
10. int numGold()
    1. Returns number of gold he has

**Protester: public Agent**

1. bool canProtestMove()
   1. Returns true if the protester can move in the direction passed in
2. bool checkPerp()
   1. Checks if it is possible for protestor to move perpendicularly
3. void decideNumMoves()
   1. Randomly decide how many squares to move before changing direction
4. void move()
   1. If runs out of squares to move, picks a random direction to move, changes direction, decides the number of moves to take during its next move. Otherwise, it’ll continue moving in the chosen direction.
5. virtual void doSomething()
   1. Takes care of all movement detailed in the spec EXCEPT for exiting.
   2. Checks if is in leaving or resting conditions
   3. Shouts when necessary
   4. Checks line of sight
   5. Checks for perpendicular movement
6. virtual bool annoy()
   1. Decrements hit points accordingly
   2. Plays a sound and increases score if it dies
7. void setRestTicks()
   1. sets the number of ticks to wait between movement based on the level
8. Direction pickRandDir()
   1. Picks a random direction to move
9. void moveInChosenDir()
   1. Allows protester to continue movement in a chosen direction when it’s not at the end of it’s “number of squares to move”
10. void setMustLeaveOilField()
    1. sets state to leave the oil field
11. \*\* There are many getters and setters that I created to make it easier to understand what I was doing during every instance, even though I had access to my private member variables in the Protester class. For example, resetActionCount() just sets my private member variable that keeps track of how many ticks have passed equal to 0 instead of me explicitly writing it out \*\*

**Regular Protester: public Protester**

1. void doSomething()
   1. Does the same thing as base class do Something

**Hardcore Protester: public Protester**

1. void doSomething()
   1. Since I didn’t get around to differentiating between protesters, does the same thing as base class’s doSomething

**Boulder : public Actor**

1. void setState()
   1. Controls the different states - waiting, falling, stable
2. virtual bool canActorsPassThroughMe()
   1. Can actors pass through a boulder? No!
3. virtual void doSomething()
   1. If it’s stable, checks the 4 squares below it
   2. It it’s waiting, play a sound when appropriate and checks if a certain number of ticks has elapsed
   3. It it’s falling, fall down!

**Goodie : public Actor**

1. virtual void updateFrackScore()
   1. Updates the score of the game
2. bool canBePickedUpByFrack()
   1. All goodies can be picked up by the FrackMan

**Water: public Goodie**

1. virtual void doSomething()
   1. Checks if alive and activates when close to FrackMan / is able to be picked up

**SonarKit: public Goodie**

1. virtual void doSomething()
   1. Checks if alive and activates when close to FrackMan / is able to be picked up

**GoldNugget: public Goodie**

1. virtual void doSomething()
   1. Does what is detailed in the spec in terms of being picked up by the FrackMan

**Barrel: public Goodie**

1. virtual void doSomething()
   1. Does what is detailed in the spec in terms of being picked up by the FrackMan

**StudentWorld**

1. void setDisplayText()
   1. Sets the display text at the top of the game screen
2. virtual int init()
   1. Initializes all game players and the game field
3. virtual void cleanUp()
   1. When a level ends, it cleans up all the actors
4. virtual int move()
   1. Does what is detailed in the spec
5. bool isDirt()
   1. Checks if there is dirt in a certain x, y position
6. void cleanDirtInTheWay()
   1. Clears the dirt that is in FrackMan’s way
7. bool touchedByFrack()
   1. Checks if items are within a certain radius of FrackMan
8. bool willMoveIntoBoulder()
   1. Returns if the proposed move will result in interacting with a boulder
9. bool isBoulder()
   1. Checks if there is a boulder in the position
10. FrackMan\* returnFrackMan()
    1. Returns a pointer to FrackMan
11. void setNumOil()
    1. Updates the number of oil barrels in the field
12. int getNumOil()
    1. Return how many oil barrels are left in the grid
13. bool canActorMoveTo()
    1. Checks if an actor can move into that position (aka there are no boulders or dirt)
14. void addActor()
    1. Adds an actor into the playing field
15. void annoyActor(Actor\*a)
    1. Annoys the appropriate actor
16. bool isRadLessThanNum()
    1. Checks if the radius / distance between actors is less than a certain number
17. bool hisProtester(Squirt\* s)
    1. Hits and annoys the protester with a squirt

**PART TWO: FUNCTIONALITY FAILED TO FINISH**

1. Boulder doesn’t do what it needs to when it hits a protester
2. Oil / Gold are not detected by Sonar Kit
3. FrackMan cannot drop gold or use sonar kits
4. Protesters do not exit the grid - they just die on the spot
5. When a protester is hit with a squirt, it is not “stunned”
6. Sonar Kits and Water Pools will not be generated
7. The code for the above objects ^^ is not fully implemented. They were just created but sonar kits do not reveal objects and water pools do not randomly appear
8. Protesters cannot be bribed
9. Squirts appear once even when in dirt - even though they don’t move through fully, they still appear
10. Hardcore Protesters are not at all differentiated from Regular Protesters
11. The display text at the top of the screen does not follow exactly as it should

**PART THREE: ASSUMPTIONS**

The spec was unambiguous, therefore I did not feel the need to make any assumptions

**PART FOUR: HOW I TESTED EACH CLASS**

**FrackMan**

1. Movement - Moved to boulders to see if can pass through them, same with clearing dirt
2. Picking up items - moving to the items and seeing if they disappear when they overlap
3. Testing all the keys to see if they do the right thing (set Direction)
4. Moving under Boulders to see if they’ll kill me intentionally

**Dirt**

1. Moved FrackMan through dirt to see if the dirt cleared correctly
2. Generation - Checked to see if the mineshaft was properly generated

**Boulder**

1. Generation - generated more than 2 boulders at once to see if they would overlap or generate in places they shouldn’t
2. Protester Interaction - Protester will not move into boulder b/c I intentionally generated one in the top row so I could see how he reacted
3. FrackMan Interaction - FrackMan will not move into boulder and it will be killed by one, tested this by directly interacting with the objects while playing the game

**Sonar Kit**

1. Didn’t finish a lot of the functionality for Sonar Kit - and since I wasn’t able to get them to generate correctly, I didn’t have much to test for here.

**Gold**

1. Generated many golds at once - set their default to visible to I could see if the generation worked properly and if they were spaced out correctly
2. Moved through the gold to see if I could pick it up and if the score increased as a result

**Squirt**

1. Used squirts on protesters in my grid to see if they would eventually die
2. Turned towards dirt patches and tried to shoot
3. Turned towards boulders and tried to shoot
4. Shot in random directions
5. Shot when it would go out of bounds

**Water**

1. Wasn’t able to generate water pools - not much to test

**Regular Protester**

1. After every step of the movement, checked to see if it did was it was supposed to
2. Moved FrackMan close to him to see if he would shout
3. Moved FrackMan close to him to see if the score / health of FrackMan would be decremented
4. Squirts - checked to see if he died after being blasted 3 times
5. Boulders - set a boulder in front of him and checked to see if he would move through it
6. Intersections - intentionally created intersections for him to move int
7. Line of Sight - moved FrackMan in and out of the direct line of sight. Checked what would happen when their x, y coordinates lined up
8. Set a breakpoint where I generated random directions to see if it would generate correctly or check all possible directions
9. Basically lots of digging different paths and seeing if he can move autonomously

**Hardcore Protester**

1. Since his functionality is the same as Regular Protesters right now, checked the same things I did for the regular protester to see if they behave the same way for now.

**Gold**

1. Generation - set it to visible and generated 20+ golds at once
2. Moved FrackMan through them to see if he would pick them up
3. Gold cannot be dropped right now, so didn’t test for that
4. Moved through the grid to see if it made itself visible when we’re close to it

**Oil**

**1.** Tested it the same way as I would for Gold, since they behave very very very similarly at this point in my project.