**Public member functions in each class**

**StudentWorld class**

void SWchangeSquirtCount(int x);

void SWdecBarrel();

void SWchangeGoldCount(int x);

void SWchangeSonarCount(int x);

* access m\_frackman’s functions that change counts of objects so that when other actor classes needs to change count of objects, can access frackman’s change count functions through student world

void addSquirt(int x, int y, GraphObject::Direction dir);

void addBribe(int x, int y);

* add new Squirt or Bribe (gold nugget dropped by frackman) to private member vector in student world class so that it can doSomething every tick because the move() function in student world has a loop going through the actors vector

void removeDirt(int x, int y);

* remove dirt for when frackman is digging
* placed in student world class because it has access to its private member array of dirt so frackman class can call this function through student world to modify dirt array

int annoyActorsWithinRadius(int x, int y, int imageID, int radius, int hitPoints);

* returns number of actors with imageID within radius around point x,y and changes their health by hitPoints
* placed in student world so that all actor classes can access this to modify other actors health or to check if there are other actors within a certain radius

bool checkForActorCoord(int xmin, int xmax, int xminBound, int xmaxBound, int ymin, int ymax, int yminBound, int ymaxBound, int imageID);

* returns true if there is an actor with imageID within xmin, xmax, ymin, ymax
* placed in student world so that all actor classes can access this to check if other actors are at a certain spot (e.g. boulder class calls this function from student world to check if there is dirt underneath the boulder, dirt is private member of student world class)

void setVisibleWithinRadius(int x, int y, int radius);

* set all objects within radius around point x,y as visible
* placed in student world so that when sonar is used it can call this function to detect other actors that are within the radius stored in the private vector member of actors

GraphObject::Direction directionToFaceFrackman(int x, int y);

* returns direction to face frackman from point x,y
* placed in student world so that protesters can access frackman’s position (stored in private member of student world as m\_frackman) and execute moves accordingly

bool canMoveInDirection(int x, int y, GraphObject::Direction dir);

* return true if position from x,y, in direction dir is not blocked by dirt or within radius 3 of a boulder
* placed in student world so that it can access boulder (in actor vector) and dirt (in dirt array), both of which are private members of student world class

string pathToPoint(int startX, int startY, string exitOrFrackman);

* returns string of directions that startX,startY can go to exit or frackman’s position
* string will end with “…UUULS”, where S is the starting position startX,startY and L will be the next step that the protester should take
* placed in student world because functions needs information of the field (m\_field array private member) and is also accessible to protesters from protester class

**Actor class**

StudentWorld\* getWorld();

* retrieve pointer to student world so that public functions in student world can be accessed

int getHealth();

* return health of actor
* placed in actor class to access private member m\_health

void decHealth(int annoyPoints);

* decrease private member variable m\_health

bool isDead();

* return true if m\_health < = 0
* placed in actor class because m\_health is private member

virtual void doSomething() = 0;

* actor is just a base class, all different actors doSomething different

**Dirt, Boulder, Squirt classes**

**Frackman, Protester, Regular Protester, Hardcore Protester classes**

**Pickups, Powers, Bribe classes**

virtual void doSomething();

* placed and implemented in all these classes because they (can) doSomething in each tick and have different actions
* Protester class has a do something that contains is generally the same for both regular and hardcore, but I created private member functions that differ the doSomething function of each type of protester

**Frackman class**

int getSquirtsLeft();

int getPlayerGoldCount();

int getPlayerSonarChargeCount();

int getNumberOfBarrelsRemainingToBePickedUp();

* return number of private member variables of these objects
* placed in public of frackman class so that student world can access this to update the status text on the screen

void changeSquirtCount(int x);

void decBarrel();

void changeGoldCount(int x);

void changeSonarCount(int x);

* change private member variables of these objects
* placed in public of frackman class so that student world can access this in its own version of the function (e.g. SWchangeSquirtCount etc.) and these SW functions in turn can be accessed by other actor classes when they change object counts

**Protester class**

bool getLeaveField();

void changeLeaveField(bool leaveField);

int getTicksToDoSomething();

void changeTicksToDoSomething(int x);

int getTicksAfterSquirt();

* return or change private member variables of these variables
* placed in public of protester class so that sub classes of protester (regular and hardcore) can access or change private member variables of protester class

int countNMoves(string path);

* returns number of moves that string has (e.g. turn right, turn left, go up – three moves)
* placed in public of protester class so that both regular and hardcore protester can access function
* (could also be placed in protected)

void moveOneInDirection(Direction dir);

* move protester by one step in direction dir
* placed in public of protester class so that both regular and hardcore protester can access function
* (could also be placed in protected)

**Object class**

void makeAppear();

bool isAppear();

int getState();

void changeState(int x);

void incScore(int x);

int getIncScore();

void changeSound(int x);

int getSound();

* return or change private member variables of these variables
* placed in public of object class so that sub classes of object (oil barrel, gold nugget, sonar kit, water pool) can access or change private member variables of object class

**Incomplete functionalities / Known bugs**

**Picking up gold by multiple protesters**

* did not have time to think of and write a function to ensure that only one protester was allowed to pick up and be affected by the bribe
* was thinking of ways to return a pointer to one protester only so that only this protester could be affected by the bribe

**Water squirt appears one block away from frackman, corrected by temp vector**

* when I introduced a new squirt to the field, it would appear one square away from the frackman, probably because of the way my student world class was structured and the timing ticks
* I created a temporary vector so that when a new squirt is introduced it will be added to the temp vector, then transferred into the main actor vector at the end of that move() tick, so that there will be a water squirt on the screen before its doSomething is executed
* not sure if this messes up with the timing or if it is the right solution but it seems to work

**Design decisions / assumptions**

**Using Pythagoras’ Theorem**

* I used this to implement my radius functions
* It made it a lot easier than coming up with the correct coordinates
* I *think* this is what the spec was asking for

**Bribe class**

* I made a new bribe class instead of putting it in the gold nugget class because I felt like the bribe had very different behavior from the gold nugget

**Tests of classes**

For the most part, classes were tested by playing the game to see if the functions worked, or using the debugger to check that variables had the correct values or that the correct if statements were being executed.

**StudentWorld class**

* tested through playing the game overall and making sure the correct screen and status appeared, and that it would go on to the next level when all the oil barrels were collected, and that the correct sounds were played in different situations (e.g. digging dirt sound)
* checked that no obvious errors (e.g. frackman or protesters walking through dirt or boulders, squirts passing through boulders etc.) were present, which means that functions in student world class are in fair working condition
* since actors seem to work well, also another indicator that functions in student world are working because most actors depend on functions in student world class

**Actor class**

* tested through sub classes
* no actor is ever created, only sub classes of actor

**Dirt class**

* checked that dirt appeared correctly on screen which means that it was initialized through its class correctly
* does not have much other functionality apart from being dug and existing (in the way of boulders, frackman and protesters)

**Boulder class**

* played the game and dug the dirt under the boulder to see if it would fall
* checked to see if it made the falling sound as it fell
* checked to see if it killed frackman and made protesters leave the field
* checked to see if it hit dirt or another boulder the falling boulder would disappear

**Squirt class**

* played the game and squirted the water gun
* checked to see if it made protesters freeze and then leave the field if hit enough times
* checked to see if it would disappear after bumping into boulder or dirt
* checked to see that it started in the right position with the right direction
* checked that it moved the right amount at the right rate

**Frackman class**

* used arrow keys to move around to see if it dug dirt
* checked that escape key would lead to player dying – giving up on level and decrease one life
* checked that pressing z would result in sonar charge making nearby objects visible
* checked that pressing tab key would leave behind one bribe (gold nugget)
* checked that pressing spacebar would squirt water only if there was enough space for water to appear

**Protester class**

* basically checked that protesters satisfied spec requirements
* e.g. it would leave field according to the queue function if it was in the leave field state
* e.g. it would shout at frackman if it was close enough, and frackman’s health would decrease
* e.g. it would pick up a bribe and either leave (regular) or freeze temporarily (hardcore)
* e.g. it would turn perpendicularly at some junctions

**Regular Protester, Harcore Protester classes**

* most of these classes were checked in protester class since both are derived from protester class
* checked that the way they were annoyed was correct – it leaves the field more easily than hardcore protester
* checked the queue function for hardcore protester worked, i.e. it will chase the frackman – used cerr to check that path string was correct and changing according to dirt being dug or position of frackman

**Object class**

* tested through sub classes
* no object is ever created, only sub classes of object
* checked when playing game that correct sounds were being played
* checked that objects being picked up would add to the status display and score

**Pickups class – Oil Barrel, Gold Nugget classes**

* tested by playing the game and using frackman to retrieve these objects
* checked that objects would appear when frackman got close enough to objects by randomly digging around the field
* checked that object would disappear after frackman got close enough to retrieve it

**Powers class – Sonar Kit, Water Pool classes**

* tested by playing the game to see if objects would appear randomly and that frackman could retrieve these objects
* checked that objects would disappear after frackman got close enough to retrieve it

**Bribe class**

* tested by getting gold nuggets then using tab key to leave behind nugget for protesters
* checked that regular protesters would leave the field immediately
* checked that hardcore protesters would freeze temporarily then continue to chase frackman
* checked that score was increased the appropriate amount when bribe was picked up