CS32 Project 3 Report

1. ***A high-level description of each of your public member functions in each of your classes, and why you chose to define each member function in its host class; also explain why (or why not) you decided to make each function virtual or pure virtual.***

StudentWorld class:

StudentWorld(std::string assetDir):

* Constructor, called when the StudentWorld (i.e., oil field) is constructed
* Initialize all the member variables

virtual ~StudentWorld():

* Destructor, called after the game is finished
* Release all the dynamically allocated spaces by deleting all the pointers, clear all the vectors
* Made virtual in case other future world will inherit it

virtual int init():

* Initialize a new game, create a new oil field
* Initialize all the member variables, including:
* int type: Prtotester number and adding time; number of Oil left; initial tick number
* vectors: available spots to put boulder/gold/oil; available spots to put waterpool
* int array: arrays indicating the distance to fracman and (60, 60)
* actor objects: fracman, dirt array, all other actors except water squirt, push other actors into the vector
* Made virtual in case other future world will inherit it

virtual int move():

* Display game status text
* Ask all the actors to do something (call doSomething())
* If an actor died during this move(), delete the object and set pointer to proper value
* Based on whether fracman is dead or fracman collected all oil, decide return value
* Made virtual in case other future world will inherit it

virtual void cleanUp():

* Release all the dynamically allocated spaces by deleting all the pointers, clear all the vectors
* Made virtual in case other future world will inherit it

void setDisplayText():

* Display the status of the game including score, level, lives, health, squirts, gold, sonar, oil
* Non-virtual no difference implementation, only used in this class

int getTick() const:

* return current tick at this level
* Non-virtual no difference implementation, only used in this class

double fracRadius(int d\_X, int d\_Y) const:

* return the radius distance from (d\_X, d\_Y) to fracman current position
* Non-virtual no difference implementation, only used in this class

bool protBribe(int d\_X, int d\_Y) const:

* bribe any protester within radius of 3 from (d\_X, d\_Y)
* if there is more than one protesters that are within radius of 3 from (d\_X, d\_Y) , return true, call the bribe() function of any one protester
* if not, return false
* Non-virtual no difference implementation, only used in this class

bool protHit(int d\_X, int d\_Y, int annoy) const:

* hit any protesters within radius of 3 from (d\_X, d\_Y), with "annoy" hitpoints
* if there is more than one protesters that are within radius of 3 from (d\_X, d\_Y) , return true, call the annoy() function with “annoy” hit points on all protesters
* if not, return false
* Non-virtual no difference implementation, only used in this class

bool checkDirt(int d\_X, int d\_Y) const:

* check if there is dirt at 4x4 square at (d\_X, d\_Y) from the dirt array
* return true if there is dirt
* Non-virtual no difference implementation, only used in this class

bool checkBoulder(int d\_X, int d\_Y)const:

* check if there is boulder at 4x4 square at (d\_X, d\_Y) from the otherObj vector
* return true if there is boulder
* Non-virtual no difference implementation, only used in this class

bool checkDirtBoulder(int d\_X, int d\_Y) const:

* check if there is dirt or Boulder at 4x4 square at (d\_X, d\_Y)
* return true if there is dirt or boulder
* Non-virtual no difference implementation, only used in this class

bool faceFrac(int d\_X, int d\_Y, GraphObject::Direction dir) const;

* check if protester at (d\_X, d\_Y) with direction dir is facing fracman
* return true if facing
* Non-virtual no difference implementation, only used in this class

bool ableToMove(GraphObject::Direction dir, int d\_X, int d\_Y) const:

* base on the direction, check if there is dirt or boulder on next position
* if yes, return false, if no, return true
* Non-virtual no difference implementation, only used in this class

int moveToFrac(int d\_X, int d\_Y) const;

* based on the array indicating the distance from fracman, return distance from (d\_X, d\_Y) to fracma`s current position
* Non-virtual no difference implementation, only used in this class

void removeDirt(int d\_X, int d\_Y);

* clear a 4x4 region of dirt at (d\_X, d\_Y) by setting dirt object in that region to be “dead”
* Non-virtual no difference implementation, only used in this class

void pushOther(Actor\* a):

* push actor a into the otherObj vector
* Non-virtual no difference implementation, only used in this class

void pushDirtFree(int d\_X, int d\_Y):

* push (d\_X, d\_Y) into dirt free vector indicating (d\_X, d\_Y) 4x4 square is not dirt free
* Non-virtual no difference implementation, only used in this class

void makeVisible(int d\_X, int d\_Y, int radius) const;

* set all invisible objects within the distance of “radius” to (d\_X, d\_Y) to be visible
* Non-virtual no difference implementation, only used in this class

void changeOil(int o);

* change the number of oil by o
* Non-virtual no difference implementation, only used in this class

void changeFracGold(int g) const;

* change the gold held by fracman by g
* Non-virtual no difference implementation, only used in this class

void annoyFrac(int p) const;

* call the annoy() function of frac man to reduce p hit points
* Non-virtual no difference implementation, only used in this class

bool alignFrac(int d\_X, int d\_Y, GraphObject::Direction &dir) const;

* check if (d\_X, d\_Y) has the same X or Y with fracman AND if there is any dirt or boulder in between them
* if yes, change dir to direction that facing fracman and return yes
* if no, return false without changing dir
* Non-virtual no difference implementation, only used in this class

void changeSonar(int num) const;

* change the number of sonar fracman has by num
* Non-virtual no difference implementation, only used in this class

void changeWater(int num) const;

* change the number of water fracman has by num
* Non-virtual no difference implementation, only used in this class

void breadthSearch(bool frac):

* if true, set the distance array from fracman; if false, set the distance array from (60, 60)
* pseudo code:

initialize the distance array to be -1.

push the destination coordinates and distance (0) into a queue.

while the queue is not empty:

for the four neighboring position:

if can move to the position AND the position has not been visited

set the distance to be current distance + 1.

save the distance data in the array.

push this position to the queue.

* Non-virtual no difference implementation, only used in this class

GraphObject::Direction optimalDirection(int d\_X, int d\_Y, bool frac);

* return the optimal direction to move to fracman or (60, 60) based on the distance array
* for four neighbor position:

if can move to that position and the distance is smaller

return that direction

* Non-virtual no difference implementation, only used in this class

bool turn(int d\_X, int d\_Y, GraphObject::Direction current\_dir, GraphObject::Direction target\_dir) const;

* if (d\_X, d\_Y) is at interset, return true;
* set the target\_dir to be the direction protester is turning to
* Non-virtual no difference implementation, only used in this class

int randInt(int min, int max) const;

* create random number between min and max inclusive
* Non-virtual no difference implementation, only used in this class

Actor class:

Actor(int imageID, int startX, int startY, Direction dir, double size, unsigned int depth, StudentWorld\* world);

* constructor: set the image ID, starting position, direction, size, image depth and studentWorld object
* set the all the actor to be alive, current world, state to be zero, ID to be corresponding image\_ID, all actor visible (gold and oil will be changed to invisible in their constructor)

virtual ~Actor() ;

* set the actor to be invisible
* has to be virtual for polymorphism

int getState() const;

* return the current state of the actor

bool getAlive() const;

* return true if the actor is alive; false otherwise

StudentWorld\* getWorld() const;

* return the current world the actor is in

bool getVisible() const;

* return true if the actor is currently visible; return false otherwise

int getID() const;

* return the ID of the actor

void setState(int state);

* set the actors state to be “state”

void killActor();

* kill the actor (set alive to be false) so it can be removed at the end of tick

void changeVisible(bool vi);

* change the visibility to be visible (true) or invisible (false)

virtual void doSomething() = 0;

* all the actor will do something during move (for dirt, something is nothing)
* I made it pure virtual so the actor is ABC, no instance of Actor class

virtual void annoyed(int a);

* When the actor is annoyed, it lose a hit points
* Virtual because fracman and protester have their own implementation after being annoyed but other actors cannot be annoyed. So this function is only redefined for fracman and protesters and not pure virtual

void move(Direction dir, int X, int Y);

* The actor will move from (X, Y) toward dir one square
* No need to be virtual since all actors move the same way

virtual void bribe();

* The function is called with the actor is bribed by gold
* Virtual because protester has its own implementation. But other actors cannot be bribed so it is not pure virtual.

Agent class:

Agent(int imageID, int startX, int startY, Direction dir, double size, unsigned int depth, StudentWorld\* world, int hit);

* Constructor. Agent include fracman and protester. Other than variable need to be constructed for all actors. It initialize the hit points which is for both fracman and protesters

virtual ~Agent();

* Nothing to free. Base class destructor needs to be virtual

virtual void annoyed(int hpoint);

* Pure virtual since both fracman and protester have their own implementation

virtual void bribe();

* Virtual since only protester can be bribed by gold

int getHitPoint() const;

* Return current hit points

void setHitPoint(int change);

* Reduce hit points by “change”

FrackMan class:

FrackMan(StudentWorld\* world);

* Constructor, initialize all the parameters needed for actor and agent
* Initialize the water squirt/sonar/health/gold the frackman having now

virtual ~FrackMan();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

virtual void doSomething();

* Things frackman does in each tick, same as the specification:
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

virtual void annoyed(int hpoint);

* Things frackman does after getting annoyed by protester or boulder, same as spec
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

int getHealth() const;

* Return the health of frackman for display
* Non-virtual since there is no other implementation

int getSquirt() const;

* Return the squirt the frackman has now
* Non-virtual since there is no other implementation

int getSonar() const;

* Return the sonar the frackman has now
* Non-virtual since there is no other implementation

int getGold() const;

* Return the gold the frackman has now
* Non-virtual since there is no other implementation

void changeGold(int g);

* change the gold the frackman has now by g
* Non-virtual since there is no other implementation

void changeSonar(int num);

* change the sonar the frackman has now by num
* Non-virtual since there is no other implementation

void changeWater(int num);

* change the water squirt the frackman has now by num
* Non-virtual **since** there is no other implementation

Protester class

Protester(StudentWorld\* world, int imageID, int hit);

* Constructor, set the variables needed for actor and agent
* Initialize variables indicating: rest tick between non-resting move, ticks to wait after stunned by water squirt, ticks needed to turn at intersect, ticks needed to make next shout at frackman, minimum numbers of steps to move before turn

virtual ~Protester();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

virtual void doSomething();

* Things both regular protester and hardcore protester does in each tick, same as spec. No such function in derived classes
* For different things they do in each tick, define pure virtual functions as private and these two types protesters can have their own implementations
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

virtual void annoyed(int hpoint);

* Things both regular protester and hardcore protester does when annoyed, same as spec. No such function in derived class
* For different things they do in each tick, define pure virtual functions as private and these two types protesters can have their own implementations
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

virtual void bribe();

* Things both regular protester and hardcore protester does after bribed, same as spec. No such function in derived class
* For different things they do in each tick, define pure virtual functions as private and these two types protesters can have their own implementations
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

void setRestTick(int n);

* Set the rest stick to be n
* Non-virtual since there is no other implementation

Regular Protester class

RegularProtester(StudentWorld\* world);

Constructor, set the variables needed for actor, agent and protester

virtual ~RegularProtester();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

Hardcore Protester class

HardcoreProtester(StudentWorld\* world);

* Constructor, set the variables needed for actor, agent and protester

virtual ~HardcoreProtester();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

Treasure class

Treasure(int imageID, int X, int Y, StudentWorld\* world, int state = 0);

* Constructor, set the variables needed for actor
* State 0 is permanent state and state 1 is temporary state. If starting in temporary state (water pool, sonar, gold for bribery), set the lifetime and start counting

virtual ~Treasure();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

virtual void doSomething();

* Things gold nuggets, oil, sonar, and water pool will do in common. No such function in derived classes
* For different things they do in each tick, define pure virtual functions as private and these four objects can have their own implementations.
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

Oil class

Oil(int o\_X, int o\_Y, StudentWorld\* world);

* Constructor, set the variables needed for actor, treasure
* Change to invisible

virtual ~Oil();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

Gold class

Gold(int g\_X, int g\_Y, StudentWorld\* world, int state);

* Constructor, set the variables needed for actor, treasure
* If in permanent state, change to invisible

virtual ~Gold();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

Sonar class

Sonar(StudentWorld\* world);

* Constructor, set the variables needed for actor, treasure

virtual ~Sonar();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

WaterPool class

WaterPool(int w\_x, int w\_y, StudentWorld\* world);

* Constructor, set the variables needed for actor, treasure

virtual ~ WaterPool ();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

WaterSquirt class

WaterSquirt(int w\_X, int w\_Y, Direction dir, StudentWorld\* world);

* Constructor, set the variables needed for actor
* Set the variable indicating how many steps it has traveled to be zero

virtual ~WaterSquirt();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

virtual void doSomething();

* Things water squirt does in each tick, same as spec
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

Dirt class

Dirt(int d\_X, int d\_Y, StudentWorld\* world) :Actor(IID\_DIRT, d\_X, d\_Y, right, 0.25, 3, world) {}

* Constructor, set the variables needed for actor

Virtual ~Dirt() {}

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

Virual Void doSomething(){}

* Do nothing
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

Boulder class

Boulder(int b\_X, int b\_Y, StudentWorld\* world);

* Constructor, set the variables needed for actor

virtual ~Boulder();

* Destructor, free nothing. Made virtual since destructor is virtual in base class. But this is optional

virtual void doSomething();

* Things boulder does in each tick, same as spec
* Made virtual since this function is virtual in base class for polymorphism. But this is optional

1. ***A list of all functionality that you failed to finish as well as known bugs in your classes***

As far as I know, I have implemented all the functionalities in the spec.

1. ***A list of other design decisions and assumptions you made***
2. The numSquaresToMoveInCurrentDirection for protesters are set to be 43 (spec says [8, 60])
3. Assume the boulder cannot be created in the mine shaft. Even partial overlap is not allowed. In reality, the boulders overlapping with shaft would be destroyed when creating the shaft.
4. Assume multiple waterpool or sonar can be created at the same place.
5. Unless explicitly said in the spec that “within a radius of N from the center”, I assume all “within a radius of N” means from the left bottom corner of the object
6. ***A description of how you tested each of your classes***

Actor 🡪 Agent 🡪 FrackMan:

First, I checked the initial position, direction, sonar, gold, water, health of the frackman to validate the constructors. Then I dig thru the sand all the way to the four boundaries to check if dirt is removed, has sound, stays in oil field and moving in the right manner. Thirdly, I tested all the keys. For the escape key, press will decrease the lives by one under game over; for spacebar key, water squirt will be fired starts and ends at right position, reducing the number by one. If facing dirt or boulder, still make noise and reducing number but no water showing up; for Z or z, sonar reduced by one and hidden items show up; for the TAB key, it generates a gold which will disappear if not picked up by Protester.

When the frackman is hit by boulder, the level is immediately over; if annoyed by protester, the health reduced by 20% each time until die.

Actor 🡪 Dirt:

Check if the dirts are created at right place of the oil field with a mine shaft.

Actor 🡪 Boulder:

First, open new games or press ESCAPE key to see if boulder is randomly distributed in the oil field except the shaft at least 6 away from each other. Then, use frackman to dig away dirts beneath the boulder. After about 30 ticks, it falls down all the way to another boulder or dirt or bottom boundary of the oil field.

If frackman is hit, game over. If protester is hit, he returns to (60, 60) and disappear.

Actor 🡪 Squirt:

First, fire squirt to see if it show up in right direction and with right beginning and ending position. Then fire against dirt, boulder or boundary, no image out but there is still sound and number reducing. Fire at protester, it takes 3 squirt for regular one to give up and 10 for hardcore one.

Actor 🡪 Treasure 🡪 Oil:

First, make the oil visible when created and check if randomly distributed in different games and far away from each other. Then turn it into invisible state. Move the frackman close to the oil to check if it shows up. Move closer to see if it can be picked up with sound and score increased. If all oil picked up, next level.

Actor 🡪 Treasure 🡪 Gold:

First, check if the gold is created at random places at right position either by setting them to be visible or use sonar. If frackman is within 4, it shows up. If getting closer, it can be picked up with sound and number/score increase.

Pressing TAB to drop a gold piece. If protester doesn`t pick up, it should disappear. If regular protester picked up, it should return with sound; if hardcore protester picked up, it should stop for a while but then continue. Picking up will increase points. Test to see it can only bribe one protester.

Actor 🡪 Treasure 🡪 Sonar:

First, check if sonar is created at (0, 60) and will disappear after a while if not picked up by frackman. If picked up by frackman, it should disappear and increase the number/score.

Actor 🡪 Treasure 🡪 WaterPool:

First, check if waterpool is created randomly in dirt-free position and will disappear after a while if not picked up by frackman. If picked up by frackman, it should disappear and increase the number/score.

Actor 🡪 Agent 🡪 Regular Protester:

First, check if the protester is created at (60, 60) with the right facing direction. Check if it moves towards the frackman with the similar speed as the sample game. Check if it automatically change its direction if hit dirt, boulder, boundary or simply walking in one direction for two far. Use water squirt or boulder to fully annoy the protester, see if it starts to go back to (60, 60). While it is going back, squirt or boulder will not hurt him anymore. It will not shout at frackman anymore. While it is going back, dig new routes see if it change routs based on shortest path. After it arrive at (60, 60), it should disappear. Then make the fracman within 4 units away from the protester and protester facing it. It should produce sound and reducing health by 20%. It should not shout again for another 15 non-resting ticks. After that, check if the protester is can see the frackman. If yes, it will turn to frackman and move towards it. If the protester walks in a straight line for a while, it should be able to turn a intersect.

When it is squirted, it should make sound and rest for a while. After it is fully annoyed, it should go back.

Actor 🡪 Agent 🡪 Hardcore Protester:

Other than what is tested for regular protester:

Move the frackman close to the protester but the protester cannot see the frack man. See if the protester can automatically detect the fracman and find the best way to follow it. When it is annoyed, it takes more quirts to quit but when it is fully annoyed, it should behave the same as regular protester.

When it is bribed, it stops for a while and start to move again.

StudentWorld:

Tested while tested above classes