**TunnelMan Report**

The hierarchy of my code looks like:

Actor

TunnelMan

Earth

GoldNuggets

Squirt

Goodies

Sonars

OilBarrels

WaterPool

Protester

RegularProtester

HardcoreProtester

**Actor**

The actor has a normal constructor that passes in value to the super constructor. I have decided to make the depth and direction a default value of 0 and right because that is what most of the actors have in common

Actor(StudentWorld\* studentWorld,int imageID, int startX, int startY, Direction startDirection, float size = 1.0, unsigned int depth = 0)

: GraphObject(imageID, startX, startY, startDirection, size, depth)

The actor also has a necessary virtual default destructor

bool TunnelManNearby(int x, int y);

Some of the classes has something to do when the TunnelMan is nearby (such as Gold nuggets, oil barrel). So I decided to make this function to check whether the tunnel man is nearby and it triggers something

bool isPickupAble(int x, int y, int x2, int y2);

Some of the classes in the game are pick-up able by the TunnelMan, this function checks whether the tunnelman is near enough with the item that it can be picked up

bool inBound(int x, int y);

Checks whether the tunnelman, or other moving objects are in Bound

**Earth**

The Earth has a default constructor:

Earth(StudentWorld\* studentWorld, int loc\_x, int loc\_y)

: Actor(studentWorld, TID\_EARTH, loc\_x, loc\_y, right, 0.25, 3)

Which also enables to set its visibility to true

virtual void doSomething() { return; };

the doSomething() on Earth returns nothing. (Because the earth does not need to do anything). Also made virtual just for good practice.

virtual ~Earth() {};

virtual destructor, made virtual for good practice.

**TunnelMan**

The tunnelman has a constructor:

TunnelMan(StudentWorld\* studentWorld)

: Actor(studentWorld, TID\_PLAYER, 30, 60, right, 1.0, 0)

The constructor also constructs The inventory of the tunnelMan and the current game status, such as score, hit points, oils left, also a private variable bool isAlive which determines whether the tunnelman is alive or not.

virtual ~TunnelMan() {};

The destructor for tunnelMan, made virtual for good practice.

virtual void doSomething();

The do something from tunnelMan, made virtual for good practice.

//add or decrease

void decreaseHitPoints(int num) { m\_hitPoints -= num; }

void addOil() { m\_Oils++; };

void addGoldNuggets() { m\_GoldNuggets++; };

void addSonars() { m\_Sonar++; };

void addWater() { m\_Water += 5; };

void addScore(int num) { m\_Score += num; };

void PickedUpOil() { m\_Oils--; };

void decreaseSonar() { m\_Sonar--; };

These are what I call the decrease/increase functions. Made to decrement or increment all the private variables of the TunnelMan.

//getter functions

int getScore() const { return m\_Score; };

int getWater() const { return m\_Water; };

int getOilLeft() const { return m\_Oils; };

int getSonar() const { return m\_Sonar; };

int getGoldNuggets() const { return m\_GoldNuggets; };

int getHitPoints() const { return m\_hitPoints; };

These are the getter functions which returns info of the tunnelMan’s private variables.

//Set the tunnelman to dead

void setDead() { isAlive = false; };

The only setter function I use in this class is the setDead function

void addSquirt(Direction dir);

This function helps with when the tunnelman shoots a squirt, where it would add a squirt in the world where available.

**Boulders**

The boulders has a constructor:

Boulders(int loc\_x, int loc\_y, StudentWorld\* studentWorld)

: Actor(studentWorld, TID\_BOULDER, loc\_x, loc\_y, down, 1.0, 1)

Virtual destructor:

virtual ~Boulders() {};

And the virtual do something function which is implemented in the cpp.

virtual void doSomething();

Some boolean functions to check the state of the boulder:

bool isDead() { return isAlive == false; };

bool stillAlive() { return isAlive == true; };

And some helper functions that will help with the implementation with the do something method. Especially in the falling state of the boulder

/Helper functions

bool NoEarthBelow();

bool NoBouldersBelow();

bool killTunnelMan();

The NoEarthBelow and NoBouldersBelow function checks for, obviously, if it has reached any earths or boulder where it would state its state to dead.

killTunnelMan checks for any TunnelMan it hits on the way down, if it hits a tunnelMan, therefore tunnelMan is set to dead.

**GoldNuggets**

The GoldNuggets has a constructor:

GoldNuggets(int loc\_x, int loc\_y, StudentWorld\* studentWorld, bool visible, bool canTunnelManPickUp, bool isPermanentState)

: Actor(studentWorld, TID\_GOLD, loc\_x, loc\_y, right, 1.0, 2)

The constructor is a bit different from any other game classes, I added the state whether the gold nuggets can be picked up by the Tunnelman or by the protester. Depending on the situation, I can construct this to be pickup able by tunnelMan only or by protester.

The gold Nuggets has a virtual destructor:

virtual ~GoldNuggets() {};

It also has a virtual do something, which has a very long implementation according to the spec

virtual void doSomething();

**Squirt**

The squirt class has a constructor :

Squirt(StudentWorld\* studentWorld, TunnelMan\* tunnelMan, Direction dir, int loc\_x, int loc\_y)

: Actor(studentWorld, TID\_WATER\_SPURT, loc\_x, loc\_y, dir, 1.0, 1)

Has a virtual destructor and do something:

virtual ~Squirt() {};

virtual void doSomething();

And has other functions that helps with the implementation of do something, such as :

bool isDead() { return isAlive == false; };

void setDead() { isAlive = false; };

void decrementDistanceTravelled() { distanceTravelled--; };

**Goodies**

The goodies class will define : **Sonars**, **WaterPool**, **Oils**, and the goodies has some functions :

The copy constructor

Goodies(int imageID, int loc\_x, int loc\_y, Direction dir, float size, unsigned int depth, StudentWorld\* studentWorld, TunnelMan\* tunnelMan)

: Actor(studentWorld, imageID, loc\_x, loc\_y, dir, size, depth)

{

Virtual destructor;

virtual ~Goodies() {};

It has a pure virtual function, do something (as Sonars, WaterPools and the Oils will have its own doSomething method by itself)

virtual void doSomething() = 0;

It can check if it’s still alive, or set itself to be dead.

void setDead() { isAlive = false; }

bool isDead() { return isAlive == false; };

And it has two getter functions that retrieves the Student World and Tunnel Man

StudentWorld\* getWorld() { return m\_StudentWorld; };

TunnelMan\* getTunnelMan() { return m\_TunnelMan; };

**Sonars, Water Pools, and Oils**

All of these classes has member functions that are very similar to each other. The only difference is the constructor (how it’s constructed).

They all have virtual destructors for good practice

They have their own doSomething() implementation, which are set to virtual (for good practice). Most of the functions are used from either the studentWorld, actor or tunnelMan

**Protester**

The protester class will be a super class for both the Regular Protester and HardCore protester.

It has a constructor:

Protester(StudentWorld\* studentWorld, TunnelMan\* tunnelMan, int imageID, int startX, int startY, Direction dir, float size, unsigned int depth)

: Actor(studentWorld, imageID, startX, startY, dir, size, depth)

It will have a reference to tunnelman and the studentworld as it would be very beneficial for the do something implementation.

It has both a virtual destructor and virtual doSomething method (for good practice)

virtual ~Protester() {}

virtual void doSomething();

It has some setter functions to to set its private variables:

void setTickWaited(int num\_ticks) { ticksWaited = num\_ticks; };

void setShoutingInterval(int num\_ticks) { ShoutingInterval = num\_ticks; };

void setnumSquaresToMoveInCurrentDirection(int num) { numSquaresToMoveInCurrentDirection = num; };

void setLeaveOilState() { leaveOilState = true; };

void setTicksWaitedAfterShot(int num) { ticksWaitedAfterShot = num; };

void setDead() { isAlive = false; };

It also has some decrementing functions:

void decrementNumSquaresToMoveInCurrentDirection() { numSquaresToMoveInCurrentDirection--; }

void decrementHitPoints(int num) { m\_HitPoints -= num; };

void decrementTicksWaitedAfterShot() { ticksWaitedAfterShot--; };

It has some getter functions that can get reference to the tunnel man and the student world

StudentWorld\* getWorld() { return m\_StudentWorld; };

TunnelMan\* getTunnelMan() { return m\_TunnelMan; };

It also has some helper functions that helps with the do something implementation

//Helper Functions

bool isFacingTunnelMan(int TunnelMan\_x, int TunnelMan\_y);

bool inLineWithTunnelMan(int TunnelMan\_x, int TunnelMan\_y);

void chaseTunnelMan(int TunnelMan\_x, int TunnelMan\_y);

bool canPickUpGoldNuggets();

bool isShotBySquirt();

bool canMove(int x, int y);

void pickDirection();

void GoToCurrDirection();

the first 3 helper functions helps with the conditions where it sees the tunnelman and chase it.

The next 2 are triggered if a protester picks up a gold nugget or is shot by a squirt

The last 3 is for the random movements the protesters make.

Since this both the implementation of Hardcore protesters and Regular Protesters are almost the same, The regular protester and Hardcore protester all only has a constructor and destructor. The doSomething() method is implemented in its super class.

**Student World**

StudentWorld(std::string assetDir)

: GameWorld(assetDir)

{

}

virtual ~StudentWorld() { cleanUp(); };

virtual int init();

virtual int move();

virtual void cleanUp();

These are the functions and constructors given by the spec. They serve as the skeleton of the TunnelMan game.

Other functions I have implemented in the StudentWorld class is, functions that generates the items needed for the oil field in every level. Getting the functions done this way makes it easy to organize the code in the init function of the student world.

void generateEarth();

void generateBoulder();

void generateTunnelMan();

void generateGoldNuggets();

void generateOilBarrels();

//With the exception of Water and Sonar, they are generated in the move() function (Protesters too)

void generateWaterPool();

void generateSonar();

void generateProtesters();

I also implement functions that cleans dead actors as the game goes by:

void cleanBoulder();

void cleanGoldNuggets();

void cleanOils();

void cleanSonars();

void cleanWaterPool();

void cleanRegularProtester();

void cleanSquirts();

Some getter functions to get reference of objects in the game, so therefore can be used by actors.

TunnelMan\* getTunnelMan() { return m\_TunnelMan; };

std::vector<Protester\*>& getProtester() { return m\_Protester; };

std::vector<Squirt\*>& getSquirts() { return m\_Squirt; };

std::vector<GoldNuggets\*>& getGoldNuggets() { return m\_GoldNuggets; };

std::vector<Boulders\*>& getBoulders() { return m\_Boulders; };

And some helper functions that helps with the doSomething implementations of the actors, and others:

//Check if there's any earth in (x,y)

bool EarthPresent(int x, int y);

//Check if there's any boulder in (x,y)

bool BoulderPresent(int x, int y);

//Dig Earth if player overlaps it

void digEarth(int x, int y);

//remove Earth if boulder is overlapping it

void removeEarth(int x, int y);

//check for valid water pool position

bool validWaterPoolPosition(int x, int y);

//check if the destination is a valid spot to move.

bool isValidSpotToMove(int pos\_x, int pos\_y, int dest\_x, int dest\_y);

//Helper functions for tunnelman

void activateSonar();

void TunnelMandropGold();

//Helper Functions for protesters

bool canMoveTowardsTunnelMan(int Protester\_x, int Protester\_y, int TunnelMan\_x, int TunnelMan\_y);

And some other functions basic functions:

void setAddProtesterTicksWaited(int num) { AddProtesterTicksWaited = num; };

void decrementAddProtesterTicksWaited() { AddProtesterTicksWaited--; };

**How I tested my classes:**

**TunnelMan**

* Make sure the tunnelMan cannot go out of bounds
* Make sure that the tunnelMan can only move to the facing direction if a corresponding directional key is hit. Else, the tunnelMan should only change direction, not move.
* Running the tunnelman through a boulder and see if it can pass through a boulder or not
* Check to see if every of the non-directional keys (z, tab, space, escape) works perfectly fine.
* Check if the tunnelMan can be annoyed by the protester and decrease its hit points. Die if hitpoints are 0

**Earth**:

Since the earth does nothing at a tick, it isn’t really tested; The only thing that’s tested is the initiation of the earth on the oil field (Make sure it’s 64 by 60), also the initiation for the trim for the well.

**Boulders**:

I tested to see if the boulder is changed to a falling state if there’s no earth below it. Also tested the boulder’s falling state by digging below the Boulders and check to see if the boulder can fall and set itself to dead until it hits Earth or another boulder. I also tested whether it can kill Protesters (and it will continue moving).

**Squirts:**

I tested Squirts to see if it can travel full through the 4 distances, checked whether it states itself to dead if it hits a boulder, earth, or after annoying a protester.

**Barrel of Oils**:

I tested whether the Barrel of oils is set to visible if the tunnelman is nearby the oil, also tested to set itself to visible if a sonar is used, checked whether it is removed from the game when the tunnelman picks up the oil (and play the sound) and can decrement the oils left in the game status. Also checked if the tunnelman can advance to the next level once all oils have been collected.

**Gold Nuggets**:

I tested whether the gold nuggets can be set to visible if a sonar is used within the specific distance, or when the tunnelman is enarby. I also tested the behavior of the gold Nuggets when it is dropped by the TunnelMan (the tunnelMan should not be able to pick the gold nuggets, only the protester). Also tested if the GoldNuggets can disappear after certain intervals if it is dropped by the tunnelMan, but it does not disappear when it has not been picked up by the TunnelMan

**For boulders, barrel of oils and Gold Nuggets**, I also tested and made sure they are a certain distance apart from each other. I tested this by making all three visible and see if they can overlap. Also checked if they do not overlap on vertical well.

**Water Pool:**

Tetsed the location of waterpools, whether they spawn in places where there is no Earth. Also checked if it can be picked up by the tunnelman and can be added as water counts. Also tested if it can disappear after certain amount of ticks if it is not picked up by the TunnelMan.

**Sonar Kit**:

I tested whether the Sonar kit can appear in the game after certain chances every tick. Tested if it can be picked up by the tunnelMan, and can disappear if it is not picked up after a certain period of time.

**Protesters:**

Tested the Protesters behavior when it is in line with tunnelMan; If nothing is blocking, it should chase the tunnelman, else, it should not be chasing the tunnelMan. Also check if the Protester can randomly move on its own (perpendicularly, or straight). Tested whether the protesters can pick up gold nuggets only dropped by the tunnelman, get hit by squirts, etc.

**Design Decisions and Assumptions**

In the code, I did not make most of my functions virtual, because I believe that each classes has its own unique behavior on reacting to some way, and therefore I made most of my member functions in the class just as I need to without thinking of the virtual.

I used vectors to contain most of my classes, specifically vectors that contains pointers. The reason I’m doing this because it would be easier to delete and add rather than using arrays. Each class has its own vector container, I did not make every class contained in one vector container because I found it more organized and easier as most classes would need to depend on the location, status and other classes.

I tried to keep duplicate code as few as possible. Some codes might look like duplicates but I implemented them just so that it can be specific to the class itself where its behavior may be different from other classes that are using the original function. This would save time for debugging in case my code crashes in a certain way.

The isAlive variable is almost there in every functions. This is to keep track whether that object still exists in the game; makes it easier for the move() function on the StudentWorld class to be able to delete dead objects.

Also, I assumed that anything y >= 60 should be able to move freely since there is no Earth or Boulders blocking in that range.

**All functionality I failed to finish**

I did not finish the Hardcore protester specifically, and since it is very similar to the regular protester, I just treat it as a regular protester for now. I also did not find a way that the protesters can find its way back to the exit when it is on the leave the oil state. If there are some additional bugs and missing functionalities, that would be because I have not been able to play it 100% to debug the game completely because of the shortness in time.