**StudentWorld Class**

virtual ~StudentWorld();

Because this class is derived from GameWorld, the destructor should be virtual. In this function I used the same code I had in the cleanUp() function to destroy the allocated memory in the Earth field, the vector of actors, and the tunnelman pointer.

virtual int init();

In this class, I set the tick counter to 0 so it could reset every level and initialized the number of barrels needed for that level. I then calculated the number of Boulders needed for the level and the number of gold nuggets. I filled the 2D array data member with Earth pointers, generated random positions for the goodies and added them to the actors vector. Then I removed the earth in places boulders were placed, and then added the player pointer.

virtual int move();

I first update the status text bar at the top. I then calculate if I should add a protester and the target amount protestors, and based on those numbers, decide to add/not add a protestor. I then calculated the probability of adding a sonar or a gold nugget and add the necessary actors. I call each actors doSomething function, removing them if they had died during the tick, and check if the player has died. I then either decrement the players lives, start a new level, or continue the game.

virtual void cleanUp();

This function is called everytime there is a new level or the level restarts. I delete the dynamically allocated memory in the actors vector, the array with the Earth pointers, and the pointer to tunnelman.

bool canMove(int x, int y) const;

When called, this function checks whether there are any boulders in the way, or if the actor is about to move out of bounds.

bool checkForEarth(int x, int y) const;

This functions loops through the earth array and returns true if there is any earth at the given location and false otherwise.

bool isTunnelManNearby(int x, int y, int rad);

I used the distance formula and the players coordinates to see if the given x,y coordinates are within the radius (rad) of the player’s location. Returns true if it is, and false if it isn’t.

bool isProtestorNearby(int x, int y, int rad);

Loops through the actors vector and checks if there is a protester within the radius of the given location. Returns true if there is and false if there isn’t.

void killProtester(int x, int y);

This function allows other actors to access the protesters and the given location through getWorld and uses its function to set it dead.

bool canAnnoyTunnelMan(Actor\* p);

This checks if the protester is facing the correct direction to yell at the tunnelman. Returns true if he can, and false if he cannot.

void dropGold(int x, int y);

This function handles the player dropping a gold nugget and makes the nugget visible to the user and also sets it so that it can only be picked up the protesters.

void addSquirt(int x, int y);

Function adds a squirt to the vector for the player and checks for boulders, earth, and the bounds before adding.

void gotGold();

This function updates player’s number of gold nuggets after picking on up.

void gotOil();

This function updates the number of barrels left to be picked up.

void gotWater();

This function increases the number of waters the player should have after picking up a waterpool.

void gotSonar();

This function increases the number of sonars the player should have after getting a sonar.

void useSonar(int x, int y);

This function loops through the actor vector and sets everything within the radius of the sonar to visible.

void annoyTunnelMan(int amount);

This allows actors to decrement the tunnelman’s health through the getWorld function.

bool removeEarth(int x, int y);

This removes any overlapping earth squares at the given position.

***Design Decisions***

struct Coordinates

{

Coordinates(int x, int y);

int x;

int y;

};

I made this private struct so I could easily return a pair of coordinates rather than calling a function twice. I only used it to generate random coordinates so since it was only used in this class, I made it private.

Coordinates getRandomLocation(int xLowerBound, int xUpperBound, int yLowerBound, int

yUpperBound);

I used this function to generate random locations for boulders, barrels and gold in the init() function, and since I only used it within this class, I made it private.

void updateStatusText();

This function was used to format the status display text bar at the top of the screen using stringstream and since I only would call this once every time the move() function was called, I made it private.

double distanceFormula(int x1, int y1, int x2, int y2) const;

This function simply takes in four values and plugs them into the distance formula. I use this with the functions above that check for actors within a certain radius, but no other classes should access it, so I made it private.

***Testing***

In order to test this class, I worked incrementally. I would first ensure that the infrastructure was in place in the Actors classes and then actually place them onto the field. Most of these functions were easy to test because I would be able to see them on the field. I checked whether the status bar was correct and evertime I would act I would check if the score was increasing correctly and the other inventories were also updating. I would escape levels multiple times to check that the locations of certain actors were actually random, and the correct number of them were present. I also had to check that the oil, gold, and boulders were within the earth, while the sonars and protesters should be in one specific place, and the water pools should only be where earth and boulders are not. When I moved the player around I ensured that the earth was being removed and the correct objects were becoming visible when I got near them.

**\*** all of the future classes have empty destructors, virtual for proper polymorphism to occur

**Actor Class : GraphObject**

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

I set visible to true, set active to true, and then initialized the student world pointer within the contructor’s body. The rest were taken care of with the initializer list with GraphObject’s contructor.

virtual ~Actor();

empty destructor

virtual void doSomething() = 0;

I made this function pure virtual so that it forces every subclass to implement it. This is required because in the StudentWorld class, the move function calls for every actor pointer it holds to call it’s doSomething function.

bool isActive();

Returns whether the actor is alive (true) or dead (false).

void setActive(bool status);

Allows you to set active status for any actor.

void newLocation(int& x, int& y, int distance, Direction dir);

Changes the input coordinates based on how far the actor wants to go in what direction.

virtual bool blocksPath();

Returns false on default. I made this function virtual so that other Actor subclasses that do block the path of the player can implement it to return true, and when I loop through the vector of actors, I can pinpoint whether the player will be able to move.

virtual bool isProtester();

Returns false on default. I made this function virtual, so that in the Protestor class I could implement it to return true, so in the future I would be able to loop through the vector of actors and find the protestors.

StudentWorld\* getWorld();

Return the StudentWorld pointer every Actor subclass should have.

***Design Decisions***

I decided to make this class an abstract base class, so that every subclass would have to implement the doSomething function. I also added a pointer to the StudentWorld class so that actors would be able to use it’s functions to have an effect on the other actors. The other private data member was a bool of whether the actor was alive or not, which would let student world know whether to take it off the field or not.

***Testing***

Testing was done in the future with the other derived classes, because if the derived classes worked with these functions, then the base class does too.

**Earth : Actor**

Earth(int startX, int startY, StudentWorld\* world);

Initializes the Earth’s location and its StudentWorld pointer.

virtual ~Earth();

empty destructor

virtual void doSomething();

Earth doesn’t do anything, so it just returns.

***Design Decisions***

None.

***Testing***

Made sure that the field was constructed properly on the field.

**TunnelMan : Actor**

TunnelMan(StudentWorld\* world);

Initializes the player’s StudentWorld pointer.

virtual ~TunnelMan();

empty destructor

virtual void doSomething();

If dead, return. Then check for user input and act accordingly. If the key pressed is escape, set the player dead and return. If it is a directional key, then check if the pathway is open and move in the direction presses. If the key pressed is space, decrement the amount of water, play the squirt sound and pass the coordinates for a new squirt to StudentWorld to handle. If the key was tab, then decrement the number of gold nuggets and call the drop gold function in student world. If the key was z, then decrement the number of sonars left, play the sound, and call the student world function to use the sonar. If the players location overlaps with any Earth object, make the digging sound and remove the earth.

int getPoints() const;

Returns the number of hit points the player has left.

void setPoints(int points);

Sets the number of hit points to the parameter.

int getSonar() const;

Returns the number of sonars the player has left.

void setSonar(int sonar);

Adds the parameter to the current number of sonars.

int getWater() const;

Returns the number of waters the player has left.

void setWater(int water);

Adds the parameter to the current number of waters.

int getGold() const;

Returns the number of gold nuggets the player has left.

void setGold(int golds);

Adds the parameter to the current number of gold.

bool annoy(int decPoints);

Returns true if the player lost their life, false otherwise. It decrements from the player’s health points, and if it has gone below 0, it sets the player dead and plays the sound to give up.

***Design Decisions***

I decided to make private data members to track the player’s inventories and health, so I had them for hit points, water, sonars, and gold.

***Testing***

I tested the tunnelman by implementing the user input keys incrementally. I first implemented the directional keys and made sure that the tunnelman moved as I said and did not run into boulders or go out of bounds while also removing overlapping earth. I then implemented the squirt gun and made sure it travelled in the correct direction. For the escape button, I simply pressed escape to check that it ended the life. And then finally I checked that I could drop the gold nugget and not pick it up and also use the sonar to set things visible.

**Protester : Actor**

Protester(StudentWorld\* world, int imageID);

Initialize counters to check when to shout, when to turn, when to do something, how many squares to move in a direction, and initialize the bool that indicates if the protestor is leaving the field.

virtual ~Protester();

empty destructor

virtual void doSomething();

Check if alive. If the current tick has not decremented to 0 then return. Once the current tick is at 0, reset it to the number of ticks to wait. If the protester if trying to leave the oil field, and is at (60,60), set dead. Decrement the number of ticks to wait until the protester can shout at the player. If it can then check if the tunnelman is nearby and if it is facing in the correct direction, then shout and decrement the number of hit points for tunnelman, while also resetting the number of ticks to wait until it can shout again then return. Otherwise, move randomly throughout the oil field. If the number of squares to move and the number of ticks until a turn are both greater than 0, decrement both counters, check if the protester can move in one more square in its current direction and move. If it cannot then reset the number of squares to move in current direction to 0. If either of the above values are 0 then pick a random direction and attempt to move there.

bool leavingField();

Returns whether the protester is leaving the oil field

void setLeavingField(bool leave);

Set whether the protester is leaving the oil field

int getSquares();

Returns the number of gold squares the protester has left to travel.

void setSquares(int squares);

Sets the number of gold squares the protester has left to travel.

int getCurTick();

Returns the current tick the protester is on.

void setCurTick(int tick);

Sets the current tick the protester is on.

int getTicksToWait();

Returns the number of ticks the protester has to wait to do something.

void setTicksToWait(int amount);

Sets the number of ticks the protester has to wait to do something.

int getPoints();

Returns the number of hit points the protester has.

void setPoints(int amount);

Sets the number of hit points the protester has.

virtual void annoy(int amount);

If the protester is leaving the field then return. Otherwise, decrement the amount from its hit points and if its hit points decrease to below 0 then play the give up sound and set leaving the field to true and also set dead. If it is still alive then play annoyed sound and reset the number of ticks to wait to do something. I made this function virtual so that my Hardcore protester could do something different.

virtual bool isProtester();

Returns true to identify protesters in the actors vector

***Design Decisions***

With the hopes that I would get to my derived subclasses I created getters and setters for each one of the variables that the two types of protesters differed over. My private data members simply kept track of the different constraints each move needed.

***Testing***

I just checked if the protesters were in the right spots when they were added and that they did not run into earth or boulders, while also turning randomly. I also ran my tunnelman near them to make sure that they shouted appropriately at different intervals of ticks and in the correct direction and radius.

**Regular : Actor**

Regular(StudentWorld\* world);

Initializes ID Image and number of hit points.

***Design Decisions***

None

**Hardcore : Actor**

Hardcore(StudentWorld\* world);

Initializes ID Image and number of hit points.

***Design Decisions***

None

**Goodies : Actor**

Goodies(int imageID, int startX, int startY, Direction dir, double size, unsigned int

depth, StudentWorld\* world);

Initializes tick counter and the Boolean that determines whether the object has been revealed or not.

virtual ~Goodies()

empty destructor

virtual int getTick();

Returns the current tick.

virtual void incTick();

Increments the tick counter.

virtual bool getVisibility();

Returns true if the object has been revealed, and false if it has not.

virtual void revealGoodie(Goodies\* obj, int rad);

Check if the object is active and if the tunnelman is nearby. If both of these conditions are true, then set visible to true, and set the data member revealed to true.

***Design Decisions***

Because many of the sitting objects have constraints on whether they are temporary or permanent or visible or not visible, I made a private data member to count the number of ticks when they need it, and a data member that keeps track of whether they were revealed when the tunnelman came near or not.

***Testing***

Since this is an abstract base class, I tested it by testing all of my subclasses.

**Boulders : Goodies**

Boulders(int startX, int startY, StudentWorld\* world);

Initialize to stable status.

virtual void doSomething();

Check if active. If it is in a stable status, then check for earth underneath. If there is no earth, change status to waiting, and increment the tick. Once the tick count is 30, change the status to falling. If the tunnelman is in the way, the decrement hit points. If a protester is in the way, the decrement its hit points, and increment the player’s score. Otherwise move one square down if it can move. If it can’t then set dead.

virtual ~Boulders()

empty destructor

virtual bool blocksPath();

Returns true to differentiate from other actors when checking for open pathways.

***Design Decisions***

I decided to make a private int data member that would tell me the status of the boulder, so 1 would mean stable, 2 would mean waiting, and 3 would mean falling.

***Testing***

I tested this class by first ensuring the location of the boulders and that the player could not run into them. I also took out a couple units of earth underneath the boulder and made sure it stayed put. I then took out all of the squares under the boulder and checked that it either annoyed the player, annoyed the protester, fell until it hit earth, or fell until it got to the border.

**Squirt : Goodies**

Squirt(int startX, int startY, Direction dir, StudentWorld\* world);

Initialize travel distance for the squirt.

virtual void doSomething();

Check if active. Then check if it stil has more distance to travel. If it doesn’t then set it dead. If it does then, if it can move one unit in its given direction, move one unit. If a protester is hit, set the squirt dead, increase the player’s score and annoy the protester. If it can’t move one unit, then set it dead.

virtual ~Squirt();

empty destructor

***Design Decisions***

I implemented a private int data member that kept track of how far it had travelled.

***Testing***

I first ensured that the squirt only traveled four units. I then made sure that it would die when it hit a boulder, earth, or a protestor. Then I check to see if the water amount decremented.

**Nugget : Goodies**

Nugget(int startX, int startY, StudentWorld\* world, bool hasBeenPicked);

If it was not dropped by the player, then set visible to false.

virtual ~Nugget()

empty destructor

virtual void doSomething();

Check if active. If it was not dropped by the tunnelman and it is not visible and the tunnelman is nearby then reveal itself and return. If it has been revealed and it was not dropped by the tunnelman and the tunnelman overlapped it, play the sound, increment the player’s amount of gold and its score, and set it dead. If it was dropped by the player and it is still visible increment the tick and check if any protesters have picked it up, at which point it would be set dead and would play the sound and increase the player’s score. If the ticks reach 100, then set dead.

***Design Decisions***

I implemented a private data member that indicated whether this was a nugget that was dropped by the player or one in the earth.

***Testing***

I reversed my code where the gold nuggets started out visible in the earth and as I moved in it’s radius it disappeared, which told me that if I switched my code, it would work in the reverse way. I also checked if the gold was picked up by the player when they overlap, and that the player could not pick up a gold nugget it had dropped. If it had been dropped, I also checked that it disappeared after a certain amount of ticks.

**Barrel : Goodies**

Barrel(int startX, int startY, StudentWorld\* world);

Set visible to false

virtual ~Barrel()

empty destructor

virtual void doSomething();

Check if active. If it is not visible the reveal goodie if player is nearby. Otherwise if it is visible and the player is overlapping it, play the sound, increment the oil count and score, and set dead.

***Design Decisions***

None.

***Testing***

I first set these to visible so that I could make sure that its locations were randomized and buried within earth. I then would have it disappear when my tunnelman moved closer so I knew that the reverse code would work as well. I finally overlapped my player with the barrel and ensured that my inventory was update and my score was incremented.

**Sonar : Goodies**

Sonar(StudentWorld\* world);

Initialize the number of ticks left before it disappears.

virtual ~Sonar()

empty destructor

virtual void doSomething();

Check if active. Increment the tick count, and if the tick count reaches the number of ticks left, then set dead and return. Otherwise, check if the player picks it up and play the sound, increase the score and the number of sonars, and then set dead.

***Design Decisions***

I implemented an int private data member to calculate how many ticks the sonar should remain for.

***Testing***

To test this, I directly added a sonar in the student world’s init function and checked to see if it would disappear if I didn’t pick it up. And then I would check if the player could pick it up and if my inventory and score was updated accordingly.

**WaterPool : Goodies**

WaterPool(int startX, int startY, StudentWorld\* world);

Initialize the number of ticks left before it disappears.

virtual ~WaterPool()

empty destructor

virtual void doSomething();

Check if active. Increment the tick count, and if the tick count reaches the number of ticks left, then set dead and return. Otherwise, check if the player picks it up and play the sound, increase the score and the number of sonars, and then set dead.

***Design Decisions***

I also implemented a private int data member to track how long the water pool should remain.

***Testing***

To test this, I first would leave the water pool alone to make sure that it would disappear after a certain amount of ticks. After checking this, I would overlap my player with a water pool and check to see that my water count and score was incremented the appropriate amount.

**Failed Functionality**

Due to the power outages, I did not have enough time to finish the Regular and Hardcore Protesters. As of right now, there are no differences between the Hardcore and Regular Protestor. The regular protester’s doSomething is lacking the ability to leave the oil field, and move if tunnelman is within sight. Also, I implemented an annoy function for the protestors, but did not develop a mechanism from which to call it, so for now, when a Boulder falls on it, it simply disappears, and being squirted and picking up gold only takes care of the goodies, incrementing the player’s score, and playing the sounds required. I also planned on making a pure virtual function for picking up gold in the Protester class and then implementing it in the subclasses so that objects could not be made with the Protestor class, but I did not get to that.