My Actor.h file contains 10 useful and functioning classes: Actor, Dirt, Agent, FrackMan, Boulder, Squirt, ActivatingObject, Oil Barrel, GoldNugget, SonarKit.

My Actor class has 6 functions. The constructor’s inputs include the image id, coordinate location, size, depth, world, and visibility booleon; some of these inputs supplement the GraphObject constructor in the member initialization list. The body of the constructor sets the visibility of the object to true or false depending on the input. The destructor sets the visibility to false. Every derived class (except for the ABC) has a graphical object linked to it, so the visibility controls the appearance of the object while in game, and thus the actor class must control it. The doSomething function returns a void and is pure virtual as every derived class must “do something” unless it is dead which leads to the isAlive function. The isAlive function returns a flag is crucial in the doSomething functions because it decides whether or not the game object is in play and the doSomething function should return immediately. The setDead function returns a void sets an actor to “dead” for whatever reason and thus it can be removed/destroyed at the end of the tick. The getWorld function returns the pointer to StudentWorld so that the base class and derived classes can control the functions in the “Game World.” The above three functions are non-virtual because they carry out only one specific function in all the classes. The next virtual function is “annoy.” I set this virtual because different objects may “annoy” another object differently. However, I didn’t finish full functionality of the game, so I don’t use the benefit of it being virtual right now.

I tested this class by simply deriving other classes and adding object to the “world.” I was able to insert visible and non-visible objects with varying position, image, direction, size, and depth. Thus, I know this class is functions properly.

My Dirt class is the simplest. It has 3 functions and is derived from Actor. The constructor takes a pointer to the world and coordinates for location. The destructor doesn’t do anything special. The doSomething function doesn’t do anything either. I tested this class just by adding it to the world and it works fine.

My Agent class contains 4 functions and is derived from Actor. It is also a base class to FrackMan and the protestors. The constructor is similar to that of Actor, whose constructor is mentioned in the member initialization list of this class. It has an integer input for hit points since each agent has measured health. The addGold function returns a void and is pure virtual, since each derived class must have a unique implementation. This function is called when an actor picks up a gold nugget. The getHitPoints function returns an integer which measured the current hitpoints of the agent. This is a constant function since no values are manipulated. The annoy function is virtual and returns a flag (which could be used to detect if hp is equal to or less than 0). The annoy function can further specialized with derived classes for interacting with different game objects and agents.

I tested this class by implemented FrackMan and printing out the health on the stat bar at the top. Further, I managed to pickup gold nuggets and also print the count in the stat bar.

The FrackMan class is derived from Agent and contains 9 functions. The constructor takes just a StudentWorld pointer because its starting location and direction is fixed every time it is initialized. The doSomething function returns a void and manages the output for each key that is pressed. The getWaterUnits functions returns a integer of the number of “Squirts” the player has remaining. The getGoldNuggets and getSonarCharge function similarly. The addGold and addSonar functions return a void and simply increment the gold nuggets and sonar charges the player contains. The canFireSquirt returns a flag and and takes an input of direction. It’s purpose to detect if the surroundings of the FrackMan permit it to fire a squirt. It is non virtual because only the FrackMan can fire a Squirt and no other classes is derived from FrackMan.

I tested this class by generating a FrackMan onto the map and moving it, firing a Squirt, and collecting objects.

The Boulder class is derived from Actor and contains 2 functions. The constructor takes a StudentWorld pointer and coordinate location. The doSomething function simply checks if there is dirt underneath the boulder. If there isn’t any it would enter a “falling state.”

I tested this class by adding boulders to the game and removing the dirt underneath them and watching them transition into a falling state and eventually get deleted from the game.

The Squirt class is derived from Actor and contains 2 functions. The constructor takes a StudentWorld pointer, coordinate location, and direction. It creates a Squirt in the game world when the player presses space bar. The doSomething function scans if the squirt hit a protestor to inflict annoyance. It also sets the squirt dead after it moves 4 units from its initial locations.

I tested this class by firing a squirt in an open direction and against a boulder. However, it seems like the game crashes when a squirt reaches a boulder. I’m assuming at this point that the error is related to getting the object pointer of objects around the squirt and the boulder specifically crashes it.

The ActivatingObject is derived from Actor and is an Abstract Base Class because it doesn’t define doSomething. In fact, it only has one constructor! The constructor includes the items mentioned in other classes and some additional flags for interacting with player or protestor.

I tested this class by testing classes which are derived from it.

The OilBarrel class is derived from ActivatingObject and contains 2 functions. The constructor’s inputs include a pointer to StudentWorld and starting coordinates. The doSomething function checks to see if FrackMan is nearby and if he is then set itself visible. If the FrackMan runs it over, then increase the score, play a sound, and set itself dead.

I tested this class by using FrackMan to move around and find the OilBarrel. The score, sound, and graphics work flawlessly.

The GoldNugget class is derived from ActivatingObject and contains 3 functions. The constructor takes in a pointer to StudentWorld, starting coordinates, and a flag for activateOnPlayer and temporary. The temporary flag indicates whether the Gold Nugget is permanently or temporarily going to remain in the game world. The doSomething function is similar to the function defined for OilBarrel as explained earlier except the sound and number of points added is different. The isTemporary function simply returns a flag which indicates whether or not the nugget is temporary. Unfortunately, I didn’t implement the game far enough to make this function useful.

I tested this class the same way I tested OilBarrel as explained earlier.

The SonarKit class is derived from ActivatingObject and contains 2 functions. The constructor takes in a pointer to StudentWorld, starting coordinates, and the number of ticks for which the object is to remain in world after being added. The doSomething is similar to that of GoldNugget’s and OilBarrel’s except that the SonarKit will be set dead if the number of ticks reaches zero after being decremented each time the function is called.

I tested this class by running the game and waiting for the icon to disappear after the ticks integer reaches 0. I also tested its other functionality the same way I tested GoldNugget and OilBarrel as explained above.

The StudentWorld.h file contains one class name StudentWorld which is derived from GameWorld which was give in the skeleton code. This class contains 17 useful functions. One getObject function returns a pointer to an actor at a specific coordinate. The other function takes in an ID for cases where there are two different objects with the same coordinates. The getFrackMan function returns a pointer to the FrackMan. This function allows for easy access to a popular pointer which also improves performance by avoiding long while loops. The numberOfBarrels returns an integer of the number of barrels. This is used for the stat bar at the top as well for checking whether the level is completed for not. The getEmptyValidCoordinates function returns a flag (which is always true) and generates coordinates when initializing objects. This way objects can maintain a 6 unit radius between each other when being initialized. The removeDeadGameObjects simply removes dead/useless objects after each tick. The updateDisplayText returns a void and is responsible for printing out the statistics at the top of the game world. The addActor function returns a void and simply adds an actor pointer to the list of all the game objects (except FrackMan). The clearDirt functions returns a void and takes a set of coordinates and clears 4 by 4 square. One example of this function being is used is for clearing dirt for boulders when they are being initialized. The canActorMoveTo functions returns a flag indicating whether or not an Actor can move to a ceratin coordinate. The revealAllNearbyObjects returns a void and takes in a set of coordinates and radius. This is utilized when a sonar charge has been used and nearby hidden objects become visible. The isNearFrackMan functions takes in an actor pointer and radius. This function returns a booleon simply when the FrackMan is within the radius specified. This is useful for implementing activating objects. None of the functions in this class are virtual because no other class is derived from this.

I tested this class by playing the game and making sure everything in the world does what it’s supposed to do.

I wasn’t able to implement the protestors; however, if I did, the highlight of the class would be implementing stacks or queues to navigate the map.