Christopher Cross

Project 4 Report

August 23, 2019

*=========================================================*

*ACTOR INHERITANCE HIERARCHY*

*/\* The following indented list represents the inheritance hierarchy for GraphObject*

*the following classes are Abstract Base Classes (ABC): Actor, Inventory, AnimateObject.*

*The RegularProtestor class serves as a base class for the HardCoreProtestor class,\*/*

**class** GraphObject;

**class** Actor; *// ← Actor is an ABC*

**class** TerrainObject; // ← Terrain Object is an ABC

**class** Earth;

**class** Barrel;

**class** Boulder;

**class** WaterRefill;

**class** Inventory; *// ← Inventory is an ABC*

**class** WaterSquirt;

**class** GoldenNuggets;

**class** SonarKit;

**class** AnimateObject; *// ← AnimateObject is an ABC*

**class** TunnelMan;

**class** RegularProtestor;

**class** HardcoreProtestor; *// ← HardCoreProtestor inherits RegularProtestor, which is a regular class*

*=========================================================*

**General Note on Implementation:**

Unless otherwise noted, the reader may assume that the implementation of the doSomething method in each of the derived classes adheres to the design laid out in the (lengthy) specifications). I implemented each of these methods by directly translating the English statements into C++ syntax (to the best of my abilities), so it the specifications serve as an accurate depiction of the logic therein. I may have defined public helper functions in the service of the doSomething method.

**ACTOR**

The actor class contains the public member functions getState( ) and setState ( ), which are defined along with a public enumeration of states for all the derived classes. If I had more time, I would have defined private state enumerations for each of my classes. The actor class is an abstract base class because its public member function doSomething ( ) is pure virtual. I chose to define a pure virtual version of the doSomething( ) function in my base class because all Actors in TunnelMan must have a blah function, and each type of actor defines their own special version of it.

**TerrainObject, Inventory, AnimateObject**

Because TerrainObject, Inventory, and AnimateObject inherit actor’s deSomething( ), they are all abstract base classes as well: this was done to prevent delaraction of those types. There was much post-hoc revision of the member functions of the abstract base classes TerrainObject and Inventory because the initially intuitive distinction I made between them (as indicated by their names) did not align with their implementation, as I had hoped. As a result, both base classes define a public member function activate ( ), whose implementation varies considerably within its derivation subtree. The AnimateObject class is the base class for the TunnelMan class and the Protestor Classes - each of which can be annoyed. Therefore, the AnimateObject class defines the following public methods, which are relatively self-explanatory:

**virtual** **void** annoy(**int** damage);

**virtual** **void** setDead() {m\_Dead = **true**; }

**virtual** **bool** isDead() {**return** m\_Dead; }

**virtual** **int** getHitPoints() { **return** m\_hitPoints; }

However, as it turned out, the setDead( ) and isDead( ) methods were made redundant by the analogous functions defined in the Actor base class. It is likely that these functions could be removed without altering the game, but since this project is due in 2 hours and I am not a betting man…

**TunnelMan**

The TunnelMan class contains the following public member functions:

**virtual** **void** doSomething();

*// Accessor Functions*

**int**& getNSquirt() { **return** m\_nWaterSquirts; }

**int**& getNKit() { **return** m\_nSonarKits; }

**int**& getNnugget() { **return** m\_nGoldenNuggets; }

The doSomething method’s implementation is described in the specifications, and I have implemented it as described. The three accessor functions return a reference value to the StudentWorld Class, where they can be appropriately incremented or decremented. I choose this design to ensure that the StudentWorld class would act as the computational hub of game and to reduce the use of object-variable pointers.

**Regular Protestor**

The Regular Protestor class contains the following public member functions:

virtual void annoy(int damage);

virtual void doSomething();

virtual void shout();

virtual void rest() {ticksToWaitBetweenMoves--; }

virtual void tick();

virtual void resetClock() { m\_shoutClock = 15; }

virtual void followExitPath(char dir);

virtual void attemptShout();

virtual void moveTowardPlayer();

virtual bool sawPlayer();

virtual void pickNewDirection();

virtual bool moveOnce();

virtual void checkForTurn();

These member functions represent a functional decomposition of the Regular Protestor’s doSomething( ) method. The intent of decomposition was to facilitate inheritance when deriving the Hardcore Protestor Class, which I was unable to do. The shout method plays the shout sound and annoys the player. The followExitPath method was intended to guide the protestor along an optimal path to the exit point using a string generated by a public method in the StudentWorld class. The attemptShout method is a wrapper function that checks the direction and shoutClock. The moveTowardPlayer method is self-explanatory. The sawPlayer method is a bool function that returns true if there is a clear path from the protestor to the player. The pickNewDirection method changes the protestor’s direction to one in which it can take at least one step forward. The moveOnce method is self-explanatory. The checkForTurn method attempts to locate a perpendicular clear path.

**Hardcore Protestor**

The Hardcore Protestor class was unimplemented, but would have inherited much of hte functionality from the Regular Protestor class.

**WaterSquirt**

The WaterSquirt class it quite simple. Its doSomething method is based around a switch that determines its direction and checks to see that it is not over the boundary of the map. It also checks for boulders and earth. The intent was that the StudentWorld object would handle collision between water squirts and protestors; such a function was implemented by never deployed and never tested.

**Barrel**

The Barrel class is also simple. Its doSomething method checks to see if the player is within range and, if so, makes itself visible or, for a closer range, calls its activate method. The activate method plays the appropriate sound, updates the values feed into the stat line, and sets itself to a dead state for subsequent removal.

**GoldenNugget**

The GoldenNugget’s doSomething method checks to see if the player is within range and, if so, updates the pertinent values and sets its state to dead. I was unable to implement the *use-as-bribe* functionality of the GoldenNugget class, so while it can increase the player’s score, it cannot save the player’s ass.

**Sonar Kit**

The Sonar Kit class does everything described in the specifications; however, I was unable to implement a method for *using* the sonar kit -- that is, the Sonar Kit increases the player’s score and such, but it cannot be used to uncover hidden goodies within range.

**WaterRefill**

The WaterRefill’s doSomething method checks to see if the player is within range and, if so, updates the pertinent values and sets its state to dead. Otherwise it decrements its waitClock since it is in a temporary state.

**Earth**

The Earth class’s doSomething method does nothing; it is an empty function. The Earth class exists to be dug and/or block squirts and protestors.

**StudentWorld**

The StudentWorld class is the computational hub of the game. It is responsible for controlling the interactions between the actor objects and it is the interface between the actors and the game and graphics frameworks. Consequently, the StudentWorld class contains may self-explanatory accessor and mutator functions that will not be discussed here. I was pretty loose about virtualizing functions; in hindsight, I should not have virtualized anything, since I did not plan to derive any StudentWorld classes! The public member functions of the StudentWorld class, along with brief descriptions, is as follows:

**virtual** **void** setDisplayText( ) - is a wrapper function that is used to get values for the stat line and call GameWorld’s method

**virtual** std::string formatDisplay(**...**) - uses and stringstream variable to format the game stat line; is called from within the scope of setDisplayText( )

**virtual** **void** digEarth(**...**) - checks the Earth-object pointer matrix to see if any earth objects overlap with the player; if so, takes appropriate steps and deletes said objects.

**virtual** **bool** checkEarth(**...**) - returns true if there is an earth object in the earth pointer field at the indicated located

**virtual** **bool** blocked(**..**) - returns true if the indicated location is within a certain distance of a boulder

**virtual** **void** crushedByRock(**...**) - kills any AnimateObject-derived variables within a certain distance of a boulder in a falling state

**virtual** **bool** hitBySquirt(.**..**) - annoys any protestor variable within a certain distance of an extant WaterSquirt object

**virtual** **double** getDistance(**...**) - calculates Euclidean distance between to coordinate pairs

**virtual** **void** decBarrelCount( ) - self-explanatory, the level is passed if the barrelCount reaches zero

**virtual** **void** addGoodies() - adds goodies according to the project specifications, can be a real time-hog

The following three functions were intended to be used to generate an exit path for a protestor that has given up:

**virtual** **void** create\_path(**...**) - it would have taken the current location of a protestor as well as a reference to that protestor’s exitPath string and called find\_path

**virtual** **void** find\_path(**...**) - implements a depth-first search and stores a string representing the path in a reference variable, which a protestor would have then used to navigate to the exit point. Taken from Kung-Hua Chang's 2015 CS 32 Final Study Guide

**virtual** **void** update\_protestorMaze( ) - updates information relating to the location of boulder and earth objects

**virtual** **bool** clearPath(**...**) - determines if there is a clear path from a protestor to the player within the Earth pointer matrix

**List of unimplemented functionality:**

1. Breadth-first search
2. GoldenNugget’s bribe-function
3. Sonar Kits’s locate-function
4. WaterSquirts’ annoy-protestor-function

\*Some of the functionality exists within the source code but could not be deployed or tested due to time constraints. These include a depth-first search function that would have accomplished the breadth-first search’s functionality.

**Testing**

**Overview:** Because the game is in a partially-completed state, I was unable to thoroughly test along the classes and methods. Furthermore, due to the dynamic nature of the program, I was unable to design a systematic assertion-based checking system. Instead, testing was primarily based on the stability of gameplay and the consistent functionality of the mechanics. The following is a set of observations based on test plays

**TunnelMan**: To test the TunnelMan object, I moved the tunnelman around the boundary of the map to ensure that it would only move to valid positions. I also checked that it dug earth appropriate, created and passed along water squirts to the gameworld for animation, and was crushed by falling rocks. The TunnelMan can be annoyed to death by falling rocks and by protestors. It is also blocked by boulders.

**Regular Protestor:** The Regular Protestor class is currently in the development and testing phase. To test it, I created a protestor at the upper right-hand corner. The protestor moves rapidly toward the player if on the same row, but not if on the same column. The protestor can annoy the player by shouting. Note: the Hardcore Protestor, which would have been drived from this class, was never implemented and thus never tested.

**Squirts of Water:** To test the squirts, I gave the player 500 squirts and ran around shooting at various things. The squirts do not behave as intended, and this is likely due to the relationship between the Earth pointer field and the coordinates of the squirt. This issue may also be the reason why Protestors do not ‘see’ the player when there is a clear vertical path.

**Barrels of Oil:** To test the barrels, I made them visible during test plays and collected them. They interface with the StudentWorld appropriately to increase the player’s score and determine a level’s victory conditions.

**Gold Nuggets:**  I made them visible during test plays and collected them. They interface with the StudentWorld appropriately to increase the player’s score; however, they cannot be dropped by a player and cannot be used to bribe protestors.

**Sonar Kits:** While sonar kits cannot be used for their intended purpose, they interface with other objects correctly, and appear at the correct location. They also disappear appropriately

**Water Refills:** To test water refills, I simply attempted to get them. They interface and disappear appropriately.

**Earth:**