Report for CS 32 Project 3

StudentWorld class:

////implementations//

1.void killPlayer(int m\_health);

deduct health from the player with the amount m\_health passed in to the function.This should be in student World class because the whole game share the same health bar. No matter what other object do to the player will result in the change in the same location as the health. It should not be a virtual function because, no other classes are inheriting this class.

2.bool generatelocation(int x,int y);

generate a random location for each gold, oil and boulders so that the distance between two objects is greater than 6. If this can be done than the function will return true. This should be in StudentWorld class because we are generating the location in the studentworld.

3.bool encounterobject(int ID,int x, int y,int radius);

in this function, we will pass in the ID of the object we want to check the distance with. Int x and int y are the coordinate of the actor that implement this function. Radius is the distance we intend to check. For example, if we want to check the distance between player and gold in the gold class. ID will be IID\_PLAYER; x, y are the coordinate of the gold. This should be in StudentWorld class because, each actor have a pointer to world and can easily access this by calling a pointer to this function.

4.void pushitem(Actor\*item);

This function is to push the item in to the actor vector in the StudentWorld class. When player unleash the squirt by pressing “space” in the players dosomething(). This function will push the generated squirt into the actor vector. Since Actor vector is in studentworld class, this function should be in studenWorld class

///water implementation///

5.bool fireable(int x,int y,Actor::Direction dir);

This function is to check whether the squirt is firable in its direction. X,y is the starting location of the squirt. If the squirt encounters the dirt or boulder on its journey, it will not be constructed and the water count will decrement by one.

//Dirt implementation//

6.bool Dirtoverlap(int x,int y);

This function is to check whether the actor that is calling this function is encounters dirt at its position(x,y). returns true if it encounters dirt and false if it does not. This class should be in StudentWorld class because it can be accessed by multiple actors.And It is dealing with the Studentworld object dirt.

7.void removeDirt(int x,int y,int dir);

This function is to remove Dirt on the map when player is moving around and encounters the dirt. This should be in StudentWorld function because it modifies the dirt on the StudentWorld Map.

//Boulder implementations//

8.bool DirtunderBoulder(int x,int y);

This function is to check whether there is dirt under boulder. It returns true if there is dirt under boulder. This function should be in StudentWOrld class because it is checking map object dirt.

9.bool movedownward(int x,int y);

This function is to check whether can we move the boulder downward one step. This function will return true if boulder will not go out of bound or encounter dirt or encounter boulder.

//Gold implementation//

void setgold(char method,int num);

this function is to add or subtract gold from the student world class. Method is whether we are subtracting or adding the gold. Num is the amount of gold we want to add. This should be in studentworld class because gold is a private member of student World.

int getgold(){return goldnum;}

This function enables other actor class to access the amount of gold currently possessed by player. This should be in studentworld class because gold is a private member of student World.

//oil implementation

void addoil(){oil\_left--;}

this function is to subtract oilleft from the student world class. Each time the player finds an oil it will call this function to update the count of oil. This should be in studentworld class because oil\_left is a private member of student World.

//sonar implementation

void addsonar(char method){if(method=='+')sonar++;if(method=='-')sonar--;}

this function is to add or subtract sonar from the student world class. Method is whether we are subtracting or adding the sonar. This should be in studentworld class because gold is a private member of student World.

int getsonar(){return sonar;}

This function enables other actor class to access the amount of sonar currently possessed by player. This should be in studentworld class because sonar is a private member of student World.

void detect();

This function is designed to detect whether there is hidden objects within the radius of 12. This function will set all the invisible object within the radius to visible. This should be in StudentWorld class because it modifies the studentworld map.

//protester implementation//

void intialMaze(int maze[][64]);

This function is transform the Studentworld map into an integer maze. The dirt and boulder will be set as -1. And the other slot will be initialized as 0.

void explore(int maze[][64],std::queue<std::pair<int,int>> &toDo,int x,int y,int level);

If the coordinate passed into this function (x,y) is moveable, in other word, maze[x][y] is zero. Then the function will push this pari of coordinate into the queue passed in. and update the value of maze[x][y]to the level value passed in.

int exitmaze(int x, int y,int endx,int endy,std::queue<std::pair<int,int>> some,int &getx,int &gety);

This function finds the optimum path for the actor to move from x,y to endx,endy.The program searches from the endpoint and mark each coordinate with the level number in the queue. If the queue reaches the start position. Then the function will set getx and gety to the step before the start position and returns the level of that step.

Actor::Direction getplayerdirection(int x,int y);

This function find the position of the player to the protestor.

This function returns the direction

bool canmove(int x,int y);

this function checks whether the protestor and player are in the same line and there is not obstacle between them. It returns true if protestor can move to player.

bool movable(int x,int y,Actor::Direction dir);

this function checks whether the actor at x,y can move one step to the direction passed in. it will return true if the actor is moveable.

void moveProtestor(int x,int y,Actor::Direction dir);

this function will move the Protestor at x,y one step in the direction passed in

Actor::Direction randDirection();

This function select a random direction for the protestor to move to

int countProtestor();

This function returns the number of protestor currently in the map

bool intersection(int x,int y,Actor::Direction dir);

this function check whether the protestor is at a intersection. This function is in the StudentWorld class because we are examining the StudentWorld map.

bool visualonPlayer(int x,int y);

This function checks whether the protestor can visualize This function is in the StudentWorld class because we are examining the StudentWorld map.

Actor::Direction pickperpendir(int x,int y,Actor::Direction dir);

This function pick a direction perpendicular to the current direction dir. This function is in the StudentWorld class because we are examining the StudentWorld map.

void getPlayerloc(int &x,int &y);

This function get the player coordinate. This function is in the StudentWorld class because this is the only way we can access player’s information.

Actor class

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

This function construct the actor and its private member

virtual ~Actor();

This is a destructor of Actor class, set the actor to invisible. This should be a virtual function because we should always make the function virtual.

virtual void doSomething()=0;

This is a virtual function because all the derived class has a function of doSomething and has different implementations.

bool isAlive();

this checks whether the actor is alive. This should not be a virtual function because every other derived classes uses the same definition of isAlive

void setDead();

this set the actor to dead. This should not be a virtual function because all the derived classes share the same method of setting themselves to die

StudentWorld\*getworld() const;

This get the studentWorld the actor is contained in. this should not be a virtual function because every other derived class share the same implementation of getworld function.

FrackMan class

void moveplayer(int dir);

This function takes in the direction to move and move the player one step to the direction desired. This should not be a virtual class because only frackman class are using this function.

The Plublic functions from the rest of the classes are just as described in the problem spec. Therefore ,I will not elaborate it here

Design decisions

The disappearing time of the gold after it is released by player is 100 ticks.

The pause of protestor when they attacked the player is 15\* the tick time.

When considering distance between two objects we are only considering the distance between their lower left points.

Test for classes.

For Dirt class

The test for dirt class is done in the first pass. Since the dirt class did nothing but showing dirt. I tested it simply by examine whether the dirt is shown properly on the map.

For FrackMan class

The test for FrackMan class is separated into two parts, the first part is done at the beginning where I tested the move function of the frackman by pressing the direction key and see if the frackman move correctly. For the other implementation, since it requires the construction of other actor class. I add the implementation after I finish constructing each clasees.

For water class; sonar class,oil class

Since they are very similar in its content. I will only discuss one of them. Take water class as an example. I first test the constructor of water class to see if it can be construct correctly. Then I add the generate water class in the StudentWorld class. To see if they are generated correctly and within the correct amount of time. Finally, after I finish the dosomething part, I add the corresponding implementation code in frackMan class and test whether the program worked correctly.

For Boulder class

Each time I finish a step specified in the spec, I will run the program to see if the boulder is performing functionally. For example, when I finish the dirtunderboulder function, I will check if the boulder will disappear after I remove the dirt.

For Gold class and squirt class those are two more complicated class. As it interacts with both player and protestor. Following the same step by step process, I first test the construction of each class, and then add the corresponding response of frackman in the frackman class. And leave the interaction with protestor. I will come back to this class after I finish the construction of protestor class and test whether the function work correctly

For Protestor class. After I constructed the constructor of the protestor class, I add the protestor in the move function of StudentWorld class. Then for each step the spec specifies, I will test the function to see whether it worked correctly. When encountering the interaction with other classes, I will go back to that class and add the corresponding implementations.

StudentWorld class. I first constructe it and compare it to the example to see if I get the correct interface. Then, for every actor class I added , I will modify the corresponding function in the studentworld class. By the time I finish all the classes, the studentWorld class is already done and tested.