Nathan Tjoar

005081232

Project 3 Report

1. StudentWorld

I was given the init(), move(), and cleanup() functions

I chose to make a getter function for getTick() so that zombie, character, and mine could access for timing parameters.

I chose to define finished() to return if the user was on the exit or not.

I chose to define penelopeX() so that we can get Penelope’s current x-position in the world.

I chose to define penelopeY() so that we can get Penelope’s current y-position in the world.

I chose to define ActorX(int i) so that we can get m\_actor[i] current x-position in the world.

I chose to define ActorY(int i) so that we can get m\_actor[i] current x-position in the world.

I chose to define ActorInt(int i) so that we can get m\_actor[i] current interacting value in the world.

I chose to define killObj(int i) so that we can kill m\_actor[i] in the world.

I chose to define cureObj(int i) so that we can cure m\_actor[i] in the world.

I chose to define infectObj(int i) so that we can infect m\_actor[i] in the world.

I chose to define exiter(int i) so that we can make m\_actor[i] exit the world.

I chose to define goToNextLevel() so that we can note that Penelope is overlapping with exit

I chose to define fireFlame() so that Penelope has the ability to make a flame fire.

I chose to define plantMine(double x, double y) so that Penelope can plant herself a mine in the possible.

I chose to define goBoom(double x, double y) so that the mine can set itself to explode in the given positions.

I chose to define vomit(double x, double y, Direction d) so that a zombie can vomit in its respective position and direction.

I chose to define jobNotDone() and jobFinished() as complements of each other.

I chose to define addFlame(), addMine(), and addVax() so that we can let the world know to update ammo values in the world.

I chose to define citizenHit() and zVomit() to let the move() function know when an object had been hit or infected so it can play a sound

I chose to define canMove(int ignoreValue, double x, double y, double currentX, double currentY) and distance(int x1, int y1, int x2, int y2) so they can detect object blocking and overlap, respectively.

Actor

I chose to define getWorld() so that the world can be gotten and we can access the Studentworld functions for our respective class.

I chose to define the interacting() as a pure virtual to allow each class to do its own thing.

I chose to define getStatus() because this is the same across all classes

I chose to define isInfected() because this is the same across all classes

I chose to define infectionCount() because this is the same across all classes

I chose to define getBirthtick() because this is the same across all classes

I chose to define pickedUp () because this is the same across all classes

I chose to define exited() because this is the same across all classes

I chose to define canDrop() because this is the same across all classes

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

I chose to define canDrop() because this is the same across all classes below it, except for dumb zombie

I chose to define cure() because this is the same across all classes and used

I chose to define objDeath() because this is the same across all classes

I chose to define incInfection() because this is the same across all classes

I chose to define exiting() because this is the same across all classes

Danger

I chose to define the interacting() as a pure virtual to allow each class to do its own thing

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

Goodie

I chose to define the interacting() to note the characteristics of this class

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

I chose to define the pickedUp() function here to acknowledge an object pickup

I chose to define a needToDo() because all objects to pickup have the same characteristic

People

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

I chose to define the interacting() as a pure virtual to allow each class to do its own thing

Projectile

I chose to define the interacting() to note the characteristics of this class

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

Zombie

I chose to define the interacting() to note the characteristics of this class

I chose to define the doSomething() as a pure virtual to allow each class to do its own thing

I chose to define a moveZombies(int type) function so that each zombie can move in itself

Exit

I chose to define interacting() because of its very specific return case

I chose to define doSomething() so we can call it later in the world

Wall

I chose to define interacting() because of its very specific return case

I chose to define doSomething() so we can call it later in the world

Pit

I chose to define interacting() because of its very specific return case

I chose to define doSomething() so we can call it later in the world

Landmine

I chose to define interacting() because of its very specific return case

I chose to define doSomething() so we can call it later in the world

GasCan

I chose to define doSomething() so we can call it later in the world

LandmineGoodie

I chose to define doSomething() so we can call it later in the world

Vaccine

I chose to define doSomething() so we can call it later in the world

Citizen

I chose to define interacting() because of it’s very specific return case

I chose to define doSomething() so we can call it later in the world

I chose to define turned() to let the world know cause of death

I chose to define exited() to let the world know the

I chose to define exiting() so that the value can be called by the Exit class

Penelope

I chose to define interacting() because of it’s very specific return case

I chose to define doSomething() so we can call it later in the world

I defined the getVaxCount(), getFlameCount(), getMineCount() to get ammo statistics.

I defined incFlame(), decFlame(), incMineAmmo(), decMineAmmo(), incVax() and decVax() so that the world can control the variables later when we pickup/use an object

Flame

I chose to define interacting() because of it’s very specific return case

I chose to define doSomething() so we can call it later in the world

Vomit

I chose to define interacting() because of it’s very specific return case

I chose to define doSomething() so we can call it later in the world

DumbZombie

I chose to define canDrop() because this Zombie can drop a vaccine

I chose to define doSomething() so we can call it later in the world

SmartZombie

I chose to define doSomething() so we can call it later in the world

1. For this project, I was able to implement mostly everything. The implementation on one of my classes may still buggy however.
2. My implementation was able to succeed in pretty much most cases. This implementation when being tested did not seem to have any visible bugs and each character did their part quite well. I am unsure if I could test much more.
3. For the studentworld class, I simply tested throughout. To start, I tested by making sure it could implement all functions that were required regardless of the limitations specified in the spec. As I implemented the other classes, I started to work more and more on seeing how the implementation was able to come together better, fixing as I worked incrementally. Studentworld was able to successfully call and organize objects onto the screen and initialize the whole program.

For the Parent classes (i.e. Actor, Danger, Projectile, People, Zombies) I tested the classes by simply trying to call them in the StudentWorld program and make versions of them. In this, I managed to ensure that all actors were Abstract Base classes of the child classes. At first, I was able to call studentworld due to a mistake of not declaring a pure virtual.

To test Penelope, I simply placed her into a random level with objects, set fire death to not, and threw some vomit in to ensure that she would be able to navigate properly. In this, I was trying to ensure that Penelope would not run into a wall, overlap with any characters, be able to pick up and place objects, shoot fire, get killed, and exit the level. Additionally, with vomit, I was trying to ensure Penelope was able to get through the level and die after infection.

Then to test vomit, I ensured that it would stay alive for long enough such that Penelope would be able to get infected and citizens would be able to be infected. For efficiency, I decreased necessary infection count for citizens so that they would turn into Zombies quicker. I set Penelope to be the one shooting vomit at first, so I could will it. I tested shooting it into a never dying flame so that I knew flame did not block. Walls did however, and I tested that as well. This worked well.

Similarly, with the flame throwers, I set the flames to build in a loop. When testing this, I removed the limitations of requiring ammo to build the flames. I shot it against walls, exits, landmines, and even the characters, ensuring that even Penelope could die if she touched a flame. This part worked fine as well, doing everything as per the specification document.

Moving on to testing a landmine, I tested this in a way to ensure that all characters could trigger it, so I forced each distinct character to move onto it and see if it triggered. Additionally, I made Penelope move onto it next to walls and exits to make sure that it could not be planted there. Also, I was able to make sure that this was not planted on an exit for faulty measure.

Next, I did the testing of pits. For this, I made sure everyone was moving and allowed the program to run for a few minutes until everything was destroyed. The characters performed spectacularly and I was very amused to hear the different sound deaths. The pits were able to delete without being deleted as well.

Continuing on, I tested the different goodies, starting with vaccines. For vaccines, I was able to make sure that in the case that it was destroyed by fire, it was not able to be picked up. Additionally, I managed to make sure that Zombies were able to make new ones and throw them into a proper direction. I was also able to make sure that given the pick-up, score would be increased. Also, upon infection and even before, I was able to use the vaccine fine.

The next goodie I tested was the gas can. I had made sure of the same things before and made sure of the ammo increments that it provided. I made sure no one but Penelope could pick it up. When this had happened, I was able to ensure that all of this was functional.

Following up, I had checked for the final pickup, landmines. For this, I ensured that each landmine worked with their specific place. I wanted to ensure specifically that it could create flames, make a pit in its position and detect character overlap once set. I also made sure that Penelope was the only one that could pick up the landmines. This worked well and was fine with overlap once tested.

Proceeding on, I had to test the characters. I started with citizen and allowed the program to run for a little to see movement. In level04.txt, there is a bug in the bottom left corner that they can overlap on each other. However, they can still move to Penelope perfectly fine and avoid walls and zombies fine.

Nearly the last one, I allowed testing of the dumb zombies. I made sure this movement was randomized within its movement space. Additionally, I made sure that if a character was in front of it, that it would just vomit or move on with the 1/3 chance.

Finally, I tested the smart zombies. Smart Zombies are able to determine if a character is close enough and move towards it. It also would often double up on one character. The movement here worked well and it still could vomit on characters. I am quite overall pleased with this result.