Project 3 Report

1. The following is my high-level descriptions, which sometimes reference earlier descriptions. They explain why functions are virtual or pure virtual in the host classes.

**class** BaseActor

BaseActor(**int** imageID, **double** startX, **double** startY, Direction dir = 0, **int** depth = 0, **double** size = 1.0);

This is the constructor which also initializes the GraphObject with similar parameters.

**virtual** **void** doSomething() = 0;

For each actor, it must implement this to be not abstract. This is why it is pure virtual and included in the base actor class.

*// traits of actor*

**virtual** **bool** blocksMovement() **const** { **return** **false**;}

**virtual** **bool** blocksFlame() **const** { **return** **false**;}

**virtual** **bool** canExit() **const** { **return** **false**;}

**virtual** **bool** isKilledByFlamePit() **const** { **return** **false**;}

**virtual** **bool** canBeInfected() **const** { **return** **false**;}

**virtual** **bool** explodesLM() **const** { **return** **false**;}

**virtual** **bool** isPerson() **const** { **return** **false**;}

**virtual** **bool** isZombie() **const** { **return** **false**;}

These functions set the traits of the class. They are all virtual because base classes may use different implementations. They are all set to false for more efficient code (less code in base classes). The purpose of each function is given by its name.

**int** checkAlive() **const** {**return** m\_isAlive;}

Returns whether the actor is alive or not for deletion.

**void** setAlive(**int** newAlive) { m\_isAlive = newAlive;}

Change the alive status when the actor should be removed.

**int** infectionCount() **const** {**return** m\_infectionCount;}

**void** setInfectionCount(**int** next) { m\_infectionCount = next;}

**bool** isSaved() **const** { **return** m\_saved;}

**void** setSaved(**bool** next) { m\_saved = next;}

**bool** isInfected() **const** { **return** m\_isInfected;}

**void** setInfected(**bool** next) { m\_isInfected = next;}

**bool** isParalyzed() **const** { **return** m\_isParalyzed;}

**void** setParalysis(**bool** next) { m\_isParalyzed = next;}

StudentWorld\* getWorld() **const** { **return** gw;}

**void** setWorld(StudentWorld\* newGW) { gw = newGW;}

These functions change statuses of the actor. Some actors may not use the functions, but their implementation is the same for all actors, which is why it is not virtual. These could be better used in a separate Agent class but since I did not use an Agent class, they must be included here.

**virtual** **void** doWhenDead() **const** {};

This function is virtual because some derived classes need to implement it but is not pure virtual because many do not. Any BaseActor could do something when dead so this is the host class.

**void** checkMove(**int**& x, **int**& y, Direction dir, **int** moves) **const**;

This function is here because several of the derived classes use the function.

**class** Penelope

Penelope(**int** x, **int** y, StudentWorld\* gw);

Constructor

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

Does all the functions of Penelope following the spec with arrow key directions.

**virtual** **bool** blocksMovement() **const** { **return** **true**;}

**virtual** **bool** canExit() **const** { **return** **true**;}

**virtual** **bool** isKilledByFlamePit() **const** { **return** **true**;}

**virtual** **bool** canBeInfected() **const** { **return** **true**;}

**virtual** **bool** explodesLM() **const** { **return** **true**;}

**virtual** **bool** isPerson() **const** { **return** **true**;}

**virtual** **void** doWhenDead() **const**;

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**class** Wall

Wall(**int** x, **int** y, StudentWorld\* gw);

Constructor

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

Does Wall stuff which is nothing.

**virtual** **bool** blocksMovement() **const** { **return** **true**;}

**virtual** **bool** blocksFlame() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**class** Exit

Exit(**int** x, **int** y, StudentWorld\* gw);

Constructor

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

Does Exit things which is checking overlap with People.

**virtual** **bool** blocksFlame() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**class** Pit

Pit(**int** x, **int** y, StudentWorld\* gw);

Constructor

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

Does Pit things which is check if Zombie or Person overlaps and kill them.

**class** Flame

Flame(**int** x, **int** y, StudentWorld\* gw, Direction dir);

Constructor

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

Does Flame things which is check overlap with things that can be flamed and kill them.

**void** ifOverlap(**int** x, **int** y);

**virtual** **bool** explodesLM() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**class** Vomit

Vomit(**int** x, **int** y, StudentWorld\* gw);

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

Does Vomit things such as check overlap and infect people.

**class** BaseGoodie

BaseGoodie(**int** imageID, **int** x, **int** y, StudentWorld\* gw);

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

Does Goodie things which is check if it is overlapping with Penelope.

**virtual** **bool** isKilledByFlamePit() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**virtual** **void** doIfOverlap() = 0;

Extra function to be implemented by other goodies. It is pure virtual because the type of goodie determines this.

**class** VaccineGoodie: **public** BaseGoodie {

**public**:

VaccineGoodie(**int** x, **int** y, StudentWorld\* gw);

**virtual** **void** doIfOverlap();

Adds one vaccine

};

**class** LandmineGoodie

LandmineGoodie(**int** x, **int** y, StudentWorld\* gw);

**virtual** **void** doIfOverlap();

Adds two mines

**class** GasCanGoodie

GasCanGoodie(**int** x, **int** y, StudentWorld\* gw);

**virtual** **void** doIfOverlap();

Adds five flames

**class** Landmine

Landmine(**int** x, **int** y, StudentWorld\* gw);

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

Checks if overlaps with People or Zombie and if it is active.

**virtual** **void** doWhenDead() **const**;

Creates flames and pit when it’s dead.

**class** BaseZombie

BaseZombie(**int** x, **int** y, StudentWorld\* gw);

Constructor

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

Moves randomly with random direction and plan. Can vomit on People if they are close.

**virtual** **bool** blocksMovement() **const** { **return** **true**;}

**virtual** **bool** isKilledByFlamePit() **const** { **return** **true**;}

**virtual** **bool** explodesLM() **const** { **return** **true**;}

**virtual** **bool** isZombie() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**virtual** **void** doWhenDead() **const**;

Plays zombie sound.

**virtual** **void** smartPlan() = {}

Distinguishes between dumb and smart zombie. It is virtual so it can be overridden by smartZombie.

**class** DumbZombie

DumbZombie(**int** x, **int** y, StudentWorld\* gw);

Constructor

**virtual** **void** doWhenDead() **const**;

Can drop vaccine goodie sometimes. Adjusts score.

**class** SmartZombie

SmartZombie(**int** x, **int** y, StudentWorld\* gw);

Constructor

**virtual** **void** doWhenDead() **const**;

Adjusts score.

**virtual** **void** smartPlan();

Checks if Citizens or Penelope is close and sets direction based on their positions.

**class** Citizen

Citizen(**int** imageID, **int** x, **int** y, StudentWorld\* gw);

Constructor

**virtual** ~Citizen();

Decreases the citizen count

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

Moves if zombie is threat or if Penelope is close. Turns into zombie if infected 500 ticks.

**virtual** **bool** blocksMovement() **const** { **return** **true**;}

**virtual** **bool** canExit() **const** { **return** **true**;}

**virtual** **bool** isKilledByFlamePit() **const** { **return** **true**;}

**virtual** **bool** canBeInfected() **const** { **return** **true**;}

**virtual** **bool** explodesLM() **const** { **return** **true**;}

**virtual** **bool** isPerson() **const** { **return** **true**;}

These functions have been described earlier as functions that determine traits for StudentWorld functions.

**virtual** **void** doWhenDead() **const**;

Turns into Zombie if death was by infection. Plays sounds based on action and increases/decreases score based on death/escape.

StudentWorld(std::string assetPath);

Constructor

~StudentWorld()

Destructor that calls cleanUp

**virtual** **int** init();

**virtual** **int** move();

**virtual** **void** cleanUp();

These are virtual since they are in GameWorld

**int** getSquaredDist(**int** x1, **int** y1, **int** x2, **int** y2) **const**;

Returns squared distance between coordinates.

**bool** checkBlock(**int** x, **int** y, **const** BaseActor\* cur) **const**;

**bool** checkBlocks(**int** x, **int** y, **const** BaseActor\* cur) **const**;

**bool** checkBlocksFlame(**int** x, **int** y) **const**;

These functions check if certain things block.

**bool** checkOverlap(**int** x1, **int** y1, **int** x2, **int** y2) **const**;

**bool** checkOverlapAll(**int** x, **int** y) **const**;

**bool** checkOverlapPenelope(**int** x, **int** y) **const**;

**bool** checkOverlapCitizen(**int** x, **int** y) **const**;

**bool** checkOverlapPeople(**int** x, **int** y) **const**;

**bool** checkLandmineOverlap(**int** x, **int** y) **const**;

These functions check if certain things overlap with actors.

**void** flamepitOverlapActors(**int** x, **int** y);

**void** infectOverlapActors(**int** x, **int** y);

These do the action on overlapping actors.

**void** exitCitizens(**int** x, **int** y);

This removes the exiting Citizens.

**void** findPenelope(**int**& x, **int**& y);

Finds Penelope’s coordinates.

**void** findNearestPerson(**int**& x, **int**& y);

Finds nearest Citizen/Penelope

**void** findNearestZombie(**int**& x, **int**& y);

Finds nearest Zombie.

**void** addActor(BaseActor\* act);

Adds an actor to the actor list.

**int** getFlameCount() **const** { **return** m\_flameCount;}

**void** setFlameCount(**int** next) { m\_flameCount = next;}

**int** getVaccineCount() **const** { **return** m\_vaccineCount;}

**void** setVaccineCount(**int** next) { m\_vaccineCount = next;}

**int** getLandmineCount() **const** { **return** m\_landmineCount;}

**void** setLandmineCount(**int** next) { m\_landmineCount = next;}

**int** getCitizenCount() **const** { **return** m\_citizenCount;}

**void** setCitizenCount(**int** next) { m\_citizenCount = next;}

These adjust the statuses of Penelope’s objects

**void** setFinishedLevel(**bool** next) { m\_finishedLevel = next;}

Sets level to finished if done.

2. All functionality was finished and there are no known bugs.

3. I followed the spec and the game given, so I made no known assumptions.

4.

BaseActor

Could not test in game but was used in a list in StudentWorld

Penelope

Used all keys possible through picking up goodies, shooting flames, using vaccines, dropping landmines, getting vomited on, dying to landmines, running into walls, exits, citizens, zombies, dying to pits, dying to flames, dying to infection, leading smart zombies and citizens.

Wall

Ran into Wall with Penelope. Shot flames into wall. Checked to see if zombies and citizens stopped at walls.

Exit

Checked if Zombies did not escape and citizens did. Made sure pits and landmines could be placed on exits. Checked that flames could not be on flames. Checked if Penelope could escape if all citizens were gone.

Pit

Checked if it was generated by a landmine. Checked to see if Penelope, Citizen, and Zombie died in it. Checked to see it was not affected by Flames. Checked to see if overlapping pits was allowed with Landmines.

Flame

Checked to see if it killed citizens and zombies and goodies. Checked to see if it was stopped at exits and walls. It can pass through walls with perfect positioning. Checked to see if they were generated by landmines. Checked if lifespan was 2 ticks.

Vomit

Checked to see if it caused infection in Penelope and Citizens. Checked to see if it lasted 2 ticks. Made sure sound was played.

BaseGoodie

Could not be tested, but was tested through its derived classes.

VaccineGoodie

Testing to see if could die by flame and if it could be picked up by Penelope/awarded Penelope 1 vaccine. Tested to see if DumbZombies could drop them sometimes.

GasCanGoodie

Testing to see if could die by flame and if it could be picked up by Penelope/awarded Penelope 5 flames.

LandmineGoodie

Testing to see if could die by flame and if it could be picked up by Penelope/awarded Penelope 2 mines.

Landmine

Tested to see if it exploded with overlap of Penelope, Zombies, and Citizens. Tested to see if it generated flames around it. Tested the inactivity period of 30 ticks with Penelope.

BaseZombie

Could not be tested directly in game but was tested with Smart and Dumb Zombies

DumbZombie

Tested if it moved randomly and if it vomited. Tested if it dropped vaccine goodie randomly sometimes. Tested it dying and the sounds it played.

Smart Zombie

Tested if it moved randomly unless a Person was nearby. Tested the smart movement of the zombie with a specific test level of many smart zombies. Tested it dying and vomiting and its sounds/scores.

Citizen

Tested to see if it moved when Zombies were nearby and if it moved when Penelope was close. Tested if it could exit through the Exit and tested if it could die to pits, flames, landmines, infection of zombies. Tested the death of infection and by changing the dumb/smart zombie random variable.

StudentWorld

Tested each of its functions in the game through the above actors. The functions are used by Actor functions. Checked to see init, move, cleanup returned correct statuses.