Andrew Kan Project 3 Report

1. A high-level description of each of your public member functions in each of your classes, and why you chose to define each member function in its host class; also explain why (or why not) you decided to make each function virtual or pure virtual. For example, “I chose to define a pure virtual version of the sneeze() function in my base Actor class because all actors in Zombie Dash are able to sneeze, and each type of actor sneezes in a different way.”
2. A list of all functionality that you failed to finish as well as known bugs in your classes, e.g. “I didn’t implement the Flame class.” or “My smart zombie doesn’t work correctly yet so I treat it like a dumb zombie right now.”
3. A list of other design decisions and assumptions you made; e.g., “It was not specified what to do in situation X, so this is what I decided to do.”
4. A description of how you tested each of your classes (1-2 paragraphs per class).

} }

1. **High-level description of each public member function of each class**
   1. **Destructor for each Actor class**
      1. **High-level description:** destructor, none of them do anything
      2. **Why defined in class:** they are the destructors
      3. **Why non-virtual/virtual/pure virtual:** virtual because you should always make destructors in inheritance virtual
   2. **Constructor for each Actor class**
      1. **High-level description:** constructs actor of that imageID class, at X and Y coordinates with direction startDirection and depth depth, with a pointer to StudentWorld
         1. Landmines have 30 safety ticks and are inactive
         2. Hazards, citizens, and zombies start with 0 ticks
         3. Humans start off uninfected, infection count = 0
         4. Penelope starts off with 0 flamethrowers/landmines/vaccines
      2. **Why defined in class:** they are the constructors
      3. **Why non-virtual/virtual/pure virtual:** nonvirtual because you can’t make constructors virtual
   3. **StudentWorld**
      1. **StudentWorld(std::string assetPath)**
         1. **High-level description:** student world constructor, sets citizens to 0, Penelope pointer to nullptr, and bool completed\_level to false
         2. **Why defined in class:** constructs student world
         3. **Why non-virtual/virtual/pure virtual:** not virtual because constructor
      2. **Virtual int init()** 
         1. **High-level description:** loads level and adds actors to collection of actors
         2. **Why defined in class:** creates the initial level
         3. **Why non-virtual/virtual/pure virtual:** virtual from gameworld class
      3. **Virtual int move()** 
         1. **High-level description:** has each of the actors doSomething(), cleans up dead actors after, updates game stat text, and either continues game, goes to next level, or player diess
         2. **Why defined in class:** handles each tick of the game because it has all the actor pointers
         3. **Why non-virtual/virtual/pure virtual:** virtual from gameworld class
      4. **Virtual void cleanup()**
         1. **High-level description:** goes through all actors and deletes them and removes them from the actor list, only deletes Penelope pointer if it isn’t nullptr already
         2. **Why defined in class:** cleans up its actor list
         3. **Why non-virtual/virtual/pure virtual:** virtual from gameworld
      5. **~StudentWorld()**
         1. **High-level description:** calls cleanup() function
         2. **Why defined in class:** student world destructor
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because no classes derived from studentworld
      6. **Void decreaseCitizens()**
         1. **High-level description:** decreases number of citizens variable by 1
         2. **Why defined in class:** need to keep track of how many citizens there are
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses it
      7. **Int numCitizens() const**
         1. **High-level description:** returns number of citizens
         2. **Why defined in class:** need to keep track of how many citizens there are
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses it
      8. **Void completeLevel()**
         1. **High-level description:** sets bool level\_completed variable to true
         2. **Why defined in class:** need to know when level is over
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses it
      9. **Int findEuclidean(int x1, int x2, int y1, int y2)**
         1. **High-level description:** finds Euclidean distance squared between 2 coordinates
         2. **Why defined in class:** used in checkBlock() and checkOverlap() functions
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because no need to change
      10. **Bool checkBlock(Actor\* a, int dir, int move)**
          1. **High-level description:** checks if an actor can move ‘move’ pixels in a direction by iterating through actors that isBlock() and penelope
          2. **Why defined in class:** used a lot in movement functions
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      11. **Bool checkOverlap(int x1, int y1, int x2, int y2)**
          1. **High-level description:** checks if euclidean distance between 2 points is less than 10 pixels (100 squared)
          2. **Why defined in class:** used a lot in movement functions
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      12. **Void doExit(Exit\* ex)**
          1. **High-level description:** iterates through actors,
             1. if citizen overlaps: increases score by 1000, plays sound, decreased num citizens, kill citizen
             2. if Penelope overlaps and num citizens = 0, completeslevel
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      13. **Void doPit(Pit\* pi)**
          1. **High-level description:** iterates through actors,
             1. if isPitable(), damage() them
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      14. **Void doVomit(Vomit\* vo)**
          1. **High-level description:** iterates through actors,
             1. if isVomitable (), infect them (and play infected sound if citizen)
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      15. **Void doGoodie(Goodie\* go)**
          1. **High-level description:** if overlaps with Penelope, increases score by 50, makes goodie dead, increases goodie count, and plays got goodie sound
          2. **Why defined in class:** needs to determine overlap with penelope
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      16. **Void doLandmine()**
          1. **High-level description:** iterates through actors,
             1. if isPitable(), damage() them
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      17. **bool shootFlame(int x, int y)**
          1. **High-level description:** determines if you can shoot a flame at (x, y), iterates through actors,
             1. if blocksHazard() and overlaps, return false
             2. return true
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      18. **void shootVomit(int x, int y)**
          1. **High-level description:** determines if you can shoot a vomit at (x, y), iterates through actors,
             1. if isHuman() and overlaps, 1/3 chance to shoot a vomit there and play vomit sound
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      19. **Void doSmartZombieDirection(SmartZombie\* zo)**
          1. **High-level description:** iterates through actors
             1. Finds distance to closest human
             2. If human is within 80 pixels, sets direction closest towards human
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      20. **Void doCitizen(Citizen\* ci)**
          1. **High-level description:** iterates through actors
             1. Finds distance to closest zombie and Penelope
             2. If Penelope is closer and within 80 pixels, moves in direction that gets the citizen closest
             3. If zombie is closer and within 80 pixels, moves in direction that gets the citizen farthest
          2. **Why defined in class:** needs to iterate through actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
      21. **Void placeLandmine/Flame/Pit/VaccineGoodie/DumbZombie/SmartZombie (int x, int y)**
          1. **High-level description:** creates new actor of said type at coordinates x and y, adds actor to actor list
          2. **Why defined in class:** some actors create these actors
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses these
      22. **increaseVaccine/Flamethrower/Landmine()** 
          1. **High-level description:**calls Penelope pointer toincrease charges by 1/5/2 when corresponding goodie is picked up
          2. **Why defined in class:** some actor classes increase number of these charges and no accessor function allowed to Penelope pointer
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses these
      23. **String updateGameStatText()**
          1. **High-level description:** makes a stringstream and creates the gamestat text of score, level, lives, vaccines, flamethrowers, landmines, infected count, has a checker if score is < 0
          2. **Why defined in class:** updates gamestattext for move function,
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because no other class uses this
   4. **Actor**
      1. **Virtual void doSomething() = 0;**
         1. **High-level description:** what each actor does during each tick
         2. **Why defined in class:** every actor has a dosomething for each tick
         3. **Why non-virtual/virtual/pure virtual:** pure virtual because every actor doSomething is different
      2. **Void makeDead()**
         1. **High-level description:** sets an actor’s alive Boolean to false (dead)
         2. **Why defined in class:** every actor starts off alive, after they are “made dead”, they can’t do anything anymore and they are cleaned up by Student World
         3. **Why non-virtual/virtual/pure virtual:** this function is nonvirtual because making an actor dead is the same for every class
      3. **Bool isAlive() const**
         1. **High-level description:** returns if the actor is alive or not
         2. **Why defined in class:** every actor has an alive member variable and this function accesses it
         3. **Why non-virtual/virtual/pure virtual:** nonvirtual because it is an accessor for a member variable
      4. **Virtual bool isBlock()/isHuman(), isZombie(), isFlammable(), isPitable, isVomitable(), blocksHazard() const (7 different but very similar functions)**
         1. **High-level description:** returns if the actor blocks other objects/is human/is a zombie/is damageable by flames/is damageable by pits/is damageable by vomit, initially all set to false
         2. **Why defined in class:** uses polymorphism to determine how actor subclasses (such as walls/humans/flames) interact with actors
         3. **Why non-virtual/virtual/pure virtual:** I made this function virtual and set them all to false to save code space. If a subclass of actor isHuman()/isFlammable(), etc., I redefine the function to be true in that subclass
      5. **Virtual void setInfected(bool status)** 
         1. **High-level description:** returns, in Human class makes actors infected by vomit/uninfected
         2. **Why defined in class: :** this function is used by the Human class but must be declared to be used to create actor objects since I use it in a StudentWorld function(shootVomit())
         3. **Why non-virtual/virtual/pure virtual:** I made this function virtual so that it would do nothing for all classes except Human, which redefines it, I didn’t make it pure virtual since even though only Human uses this function, it would be a waste of code space to make it just return for every other actor class
      6. **Virtual bool isInfected() const**
         1. **High-level description:** returns false, in Human class returns if actor is infected
         2. **Why defined in class:** this function is used by the Human class but must be declared to be used to create actor objects since I use it in a StudentWorld function (shootVomit())
         3. **Why non-virtual/virtual/pure virtual:** I made this function virtual so that it would do nothing for all classes except Human, which redefines it, I didn’t make it pure virtual since even though only Human uses this function, it would be a waste of code space to make it just return false for every other actor class
      7. **Virtual int getScore()** 
         1. **High-level description:** gets score of killing actor, initially 0
         2. **Why defined in class:** every actor increases score when they die
         3. **Why non-virtual/virtual/pure virtual:** virtual because every actors score is different, not pure virtual because to save code of just setting it to 0 here
      8. **Virtual int getKillSound()**
         1. **High-level description:** plays kill sound if actor has one such as sound\_landmine\_explode, initially sound\_none
         2. **Why defined in class:** lot of actors have kill sounds
         3. **Why non-virtual/virtual/pure virtual:** virtual because actors have different kill sounds, not pure virtual because to save code of setting it to sound\_none here
      9. **void damage()**
         1. **High-level description:** what an actor does when it is damaged by flames/pits/etc. Increases score (using a virtual private getScore() function), makes the actor dead (using makeDead()), does their dead action (using doDead()), and plays the actor’s kill sound (using getWorld()->playSound(a virtual private getKillSound() function)) if they have one
         2. **Why defined in class:** every actor does something when they are damaged: increasing/decreasing score, make themselves dead, doing a dead action, and playing a kill sound
         3. **Why non-virtual/virtual/pure virtual:** this function is non-virtual because every actor increases score, becomes dead, does a dead action, and plays a kill sound when they are damaged. While the score/dead action/kill sounds are different for each actor, they are handled by other virtual functions
      10. **Virtual void doDead()**
          1. **High-level description:** returns, similar to destructor of an actor, ex: decreases num\_citizens when a citizen dies, places flames when a landmine is used, etc.
          2. **Why defined in class:** Some actors such as citizens, flames, and dumb zombies do an action when they die. I didn’t make this function part of the destructor because when Student World is destructing itself, leftover landmines/dumb zombies will try to introduce new actors using Student World which has already destructed, returning an error.
          3. **Why non-virtual/virtual/pure virtual:** I made this function virtual because different actors have different death actions. I didn’t make this function pure virtual to save code space.
      11. **StudenWorld\* getWorld() const**
          1. **High-level description:** returns pointer to the student world
          2. **Why defined in class:** actor functions sometimes need to access student world functions, this function lets them do that
          3. **Why non-virtual/virtual/pure virtual:** non-virtual because student world is same for each actor class
   5. **Human**
      1. **Virtual bool isBlock(), isFlammable(), isPitable(), isVomitable(), isHuman() const**
         1. **High-level description:** humans are all of these, returns true
         2. **Why defined in class:** because humans are all of these
         3. **Why non-virtual/virtual/pure virtual:** virtual function from actor
      2. **Bool isInfected() const**
         1. **High-level description:** returns the bool member variable m\_infectionStatus, aka if the human is infected
         2. **Why defined in class:** humans can become infected, some functions need to know that
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because the same for every human
      3. **Void setInfected(bool status)**
         1. **High-level description:** sets the m\_infectionStatus variable to status, true when human is infected, false when Penelope uses a vaccine
         2. **Why defined in class:** humans can become infected, this function does that
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because the same for every human
      4. **Int getInfectionCount() const**
         1. **High-level description:** returns m\_infectionCount variable, aka how long the human has been infected
         2. **Why defined in class:** only humans can become infected/have a time of infection
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because the same for every human
      5. **Void setInfectionCount(int x)**
         1. **High-level description:** sets m\_infectionCount to x. when Penelope uses a vaccine, x = 0
         2. **Why defined in class:** Penelope (and with more features other humans) can change their infection count
         3. **Why non-virtual/virtual/pure virtual:** non-virtual because the same for every human
      6. **Virtual void doSomething()** 
         1. **High-level description:** 
            1. checks if human is alive,
            2. checks if they are infected, if they’re infected, increase m\_infectionCount

if m\_infectionCount == 500, make the human dead and do their doInfect function

* + - * 1. doDifferent movement for each human
      1. **Why defined in class:** these are the common actions of all humans
      2. **Why non-virtual/virtual/pure virtual:** virtual because different humans have different doDifferents
  1. **Penelope** 
     1. **Int getLandmines()**
        1. **High-level description:** returns number of landmines penelope has
        2. **Why defined in class:** penelope can have landmines
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope uses landmines
     2. **Int getFlamethrowers()**
        1. **High-level description:** returns number of flamethrowers penelope has
        2. **Why defined in class:** penelope can have flamethrowers
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope uses flamethrowers
     3. **Int getVaccines()**
        1. **High-level description:** returns number of vaccines penelope has
        2. **Why defined in class:** penelope can have vaccines
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope uses vaccines
     4. **Void increaseVaccines(int x)**
        1. **High-level description:** increases number of vaccines Penelope has by x
        2. **Why defined in class:** penelope can increase her vaccines by picking up goodies
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope can increase her vaccines
     5. **Void increaseFlamethrowers(int x)**
        1. **High-level description:** increases number of flamethrowers Penelope has by x
        2. **Why defined in class:** penelope can increase her flamethrowers by picking up goodies
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope can increase her flamethrowers
     6. **Void increaseLandmines(int x)**
        1. **High-level description:** increases number of landmines Penelope has by x
        2. **Why defined in class:** penelope can increase her landmines by picking up goodies
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only Penelope can increase her landmines
     7. **Virtual void doDifferent()**
        1. **High-level description:** gets user key input, then moves her up/right/left/down or shoots flames/uses a vaccine/places a landmine
        2. **Why defined in class:** the player can make Penelope do something every tick
        3. **Why non-virtual/virtual/pure virtual:** virtual because from human class
     8. **Virtual void doInfect()** 
        1. **High-level description:** plays SOUND\_PLAYER\_DIE when Penelope becomes a zombie
        2. **Why defined in class:** Penelope should do something when she becomes a zombie
        3. **Why non-virtual/virtual/pure virtual:** virtual because from human class
  2. **Citizen**
     1. **Virtual void doDead();**
        1. **High-level description:** calls student world function to decrease number of citizens when a citizen dies
        2. **Why defined in class:** citizens should tell the world when they die
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
     2. **Void moveCitizen(Direction dir)**
        1. **High-level description:** if citizen isn’t blocked, moves citizen 2 pixels in that direction
        2. **Why defined in class:** to move citizens
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because only citizens should move themselves
     3. **Virtual void doDifferent()**
        1. **High-level description:** if citizen tick is 0 mod 2, returns, else calls the student world function doCitizen() that determines how the citizen moves
        2. **Why defined in class:** determines how citizens move
        3. **Why non-virtual/virtual/pure virtual:** virtual from human class
     4. **Virtual void doInfect()** 
        1. **High-level description:** decreases score by 1000, decreases citizen number, plays zombie born sound, 70% chance places dumb zombie and 30% chance places smart zombie at current location
        2. **Why defined in class:** citizen does something when it turns into a zombie
        3. **Why non-virtual/virtual/pure virtual:** virtual from human class
  3. **Zombie**
     1. **Virtual bool isBlock(), isFlammable(), isPitable(), isZombie() const**
        1. **High-level description:** zombies are all of these
        2. **Why defined in class:** because zombies are all of these
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor
     2. **void doRandomDirection()** 
        1. **High-level description:** chooses random direction for zombie to move in
        2. **Why defined in class:** zombies can move in random directions
        3. **Why non-virtual/virtual/pure virtual:** non-virtual because choosing a random direction is the same for all zombies
     3. **Virtual void doDifferentDirection()** 
        1. **High-level description:** selects what direction for zombie to move in
        2. **Why defined in class:** zombies move in directions
        3. **Why non-virtual/virtual/pure virtual:** pure virtual because different for all zombies
     4. **Virtual void doSomething()**
        1. **High-level description:** checks if alive, every other tick don’t do anything, if it can shoot vomit, it will, else it will move until movement plan distance is 0 then choose a new direction using doDifferentDirection() and movement plan distance
        2. **Why defined in class:** zombies move and shoot vomit
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
  4. **Dumb Zombie**
     1. **Virtual void doDead()**
        1. **High-level description:** 1/10 chance for dumb zombie to fling a vaccine goodie SPRITE\_HEIGHT/WIDTH away when it dies
        2. **Why defined in class:** dumb zombies drop vaccines on death
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
     2. **Virtual void doDifferentDirection()** 
        1. **High-level description:** calls doRandomDirection() because dumb zombies only move randomly
        2. **Why defined in class:** dumb zombies move in directions
        3. **Why non-virtual/virtual/pure virtual:** virtual from zombie class
  5. **Smart Zombie**
     1. **Virtual void doDifferentDirection()** 
        1. **High-level description:** calls student world doSmartZombieDirection() because smart zombies move smartly
        2. **Why defined in class:** smart zombies move in directions
        3. **Why non-virtual/virtual/pure virtual:** virtual from zombie class
  6. **Wall**
     1. **Virtual bool isBlock(), blocksHazard() const**
        1. **High-level description:** wall does these
        2. **Why defined in class:** wall does these
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor
     2. **Virtual void doSomething()**
        1. **High-level description:** returns because walls don’t do anything
        2. **Why defined in class:** walls don’t do anything
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
  7. **Exit**
     1. **Virtual bool blocksHazard() const**
        1. **High-level description:** exit does this
        2. **Why defined in class:** exit does this
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor
     2. **Virtual void doSomething()**
        1. **High-level description:** checks if alive, calls student world doExit() function
        2. **Why defined in class:** exits do things
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
  8. **Pit**
     1. **Virtual void doSomething()** 
        1. **High-level description:** checks if alive, calls student world doPit() function
        2. **Why defined in class:** pits do things
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
  9. **Hazard**
     1. **Virtual void doSomething()**
        1. **High-level description:** checks if alive, kills hazard on second tick, calls hazard doDifferent() function
        2. **Why defined in class:** hazards do things
        3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
     2. **Virtual void doDifferent()**
        1. **High-level description:** does appropriate action of hazard
        2. **Why defined in class:** hazards do things
        3. **Why non-virtual/virtual/pure virtual:** pure virtual because every hazard does something different
  10. **Flame**
      1. **Virtual void doDifferent()**
         1. **High-level description:** calls student world doFlame() function
         2. **Why defined in class:** flames do things
         3. **Why non-virtual/virtual/pure virtual:** virtual from hazard class
  11. **Vomit**
      1. **Virtual void doDifferent()**
         1. **High-level description:** calls student world doVomit() function
         2. **Why defined in class:** vomits do things
         3. **Why non-virtual/virtual/pure virtual:** virtual from hazard class
  12. **Goodie**
      1. **Virtual void doSomething()**
         1. **High-level description:** checks if alive, calls student world doGoodie() function
         2. **Why defined in class:** goodies do things
         3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
      2. **Virtual bool isFlammable() const**
         1. **High-level description:** goodies die to flames
         2. **Why defined in class:** goodies die to flames
         3. **Why non-virtual/virtual/pure virtual:** virtual from actor
      3. **Virtual void doDifferent()**
         1. **High-level description:** does appropriate action of goodie
         2. **Why defined in class:** goodies do things
         3. **Why non-virtual/virtual/pure virtual:** pure virtual because every goodie does something different
  13. **Vaccine Goodie**
      1. **Virtual void doDifferent()**
         1. **High-level description:** calls student world increaseVaccine() function
         2. **Why defined in class:** vaccine goodies give vacciens
         3. **Why non-virtual/virtual/pure virtual:** virtual from goodie class
  14. **Gas Can Goodie**
      1. **Virtual void doDifferent()**
         1. **High-level description:** calls student world increaseFlamethrower() function
         2. **Why defined in class:** gas can goodies give flamethrowers
         3. **Why non-virtual/virtual/pure virtual:** virtual from goodie class
  15. **Landmine Goodie**
      1. **Virtual void doDifferent()**
         1. **High-level description:** calls student world increaseLandmine() function
         2. **Why defined in class:** landmine goodies give landmines
         3. **Why non-virtual/virtual/pure virtual:** virtual from goodie class
  16. **Landmine**
      1. **Virtual void doSomething()** 
         1. **High-level description:** checks if alive, reduces safety tick by 1 until 0, then always calls student world doLandmine() function
         2. **Why defined in class:** landmines do things
         3. **Why non-virtual/virtual/pure virtual:** virtual from actor
      2. **Virtual bool isFlammable() const**
         1. **High-level description:** returns true
         2. **Why defined in class:** landmines are flammable
         3. **Why non-virtual/virtual/pure virtual:** virtual from actor class
      3. **Virtual void doDead()**
         1. **High-level description:** places flame in 3x3 square around itself and pit in center if they can be placed there
         2. **Why defined in class:** only landmines do this when they die
         3. **Why non-virtual/virtual/pure virtual:** virtual from actor

1. **Functionality/known bugs**
   1. I completed all functionality listed in the spec and to my knowledge, my code does not have any known bugs.
2. **Design decisions/assumptions:**
   1. For my actor subclasses, I had Penelope and Citizens are part of a Human class, Dumb Zombies and Smart Zombies are part of a Zombie class, Flame and Vomit are part of a Hazard class, and Landmine/Gas Can/Vaccine goodies are part of a Goodie class. I decided to have a separate Penelope pointer because the spec recommended to do so. I followed the best OOP practices to the best of my ability. I used a list for my collection of actors. I made a findEuclidean() function that found the Euclidean distance squared between 2 (x, y) coordinates to help with my checkBlock() and checkOverlap() functions. I used a naming scheme for all of my StudentWorld functions: doX() did the doSomething of class X, shootX() determined if could a hazard X could be shot at that coordinate, placeX() inserted a new actor X at that coordinate. For the rest of my design decisions/assumptions, I checked the spec or interaction with the given ZombieDash.exe and modified my code to match it.
3. **Testing**
   1. **StudentWorld**
      1. I tested loading levels, making sure all the actors are cleaned up correctly, game completing when all the levels are done, game ending when player loses all 3 lives, score displaying correctly with negative score, trying to load illegitimate levels, destructor and cleanup working correctly together. I tested that the level ends when all citizens are gone and Penelope reaches the exit and that number of citizens decreases when a citizen dies/reaches the exits. I tested the game stat screen updated correctly and the goodies were reset at the start of each level.
   2. **Actor**
      1. Since we never create objects of just Actor, not much testing could be done with this class. I tested that you can’t create just an Actor because doSomething() is pure virtual, and that every actor was being set to dead when it was damaged.
   3. **Human**
      1. I tested that humans have an infection status and count, and that when the infection count reaches 500 something happens to them.
   4. **Penelope**
      1. I tested that Penelope is blocked by walls, citizens, and zombies. I tested that the level is reset when Penelope dies, she moves 4 pixels at a time, and all the key inputs work correctly.
   5. **Citizen**
      1. I tested that citizens reduce score by 1000 when they die and increase score by 500 when they reach the exit. I tested they reach the exit properly, move 2 pixels at a time, move towards Penelope correctly (vertically when on same column, horizontally when on same row, closest direction otherwise), usually stationary, and move away from zombies correctly. I tested that they are blocked by walls, citizens, and zombies. I tested that they turn into zombies after their infection count reaches 500, and the player loses 1000 score, and the zombieborn sound is played. I tested that the player loses 1000 score and the citizendie sound is played when they are killed by flame or pits or landmines.
   6. **Zombie**
      1. I tested that zombies are blocked by walls, humans, and other zombies. I tested that they die to flames, landmines, and pits and give the player 1000/2000 score when done so. I tested that they vomit with 1/3 chance by having them in vomit range a lot and shoot vomit correctly. I tested that they don’t do anything every other tick and move 1 pixel at a time.
   7. **Dumb Zombie**
      1. I tested that dumb zombies move randomly with their movement plan distance, and that they have a 1/10 chance to drop a vaccine goodie when they are killed at a random direction SPRITE\_WIDTH/HEIGHT away. I tested this by having the function print the chance and killing lots of zombies until I inferred that the randomness was working correctly. I tested that they gave 1000 score when they died.
   8. **Smart Zombie**
      1. I tested that smart zombies move towards humans correctly when they are in range. I mainly did this by locking a citizen and smart zombie in a room of walls for both my code and the given ZombieDash program and making sure they are similar. I tested that they gave 2000 score when they died.
   9. **Wall**
      1. I tested that walls didn’t do anything during their tick and that humans and zombies couldn’t run into them. I also checked that they blocked flames created from flamethrowers and flames created from landmines. I made sure that humans and zombies could move between 2 walls that were 1 wall apart.
   10. **Exit**
       1. I tested that landmines/pits could be made on exits. I made sure citizens disappeared when they overlapped with it, played the appropriate sound, and increased player score by 500. I made sure Penelope could usually walk over it and that the level ended when Penelope overlapped with one and all citizens were gone.
   11. **Pit**
       1. I tested that pits killed humans and zombies when overlapped with.
   12. **Hazard**
       1. I tested that hazards died on their second tick and did their appropriate action on their first tick. I tested that hazards could not be made on exits or walls.
   13. **Flame**
       1. I tested that flames are blocked by walls and exits but not by anything else by shooting them at everything. I tested that flames kill goodies, kills landmines, kills zombies, and kills citizens.
   14. **Vomit**
       1. I tested that the appropriate sound is played when vomit is created. I tested that they could only be created on humans and that they infected humans and started the infection count timer.
   15. **Goodie**
       1. I tested that goodies increase score by 50 when picked up and play the appropriate sound. I tested that this only happens when Penelope overlaps with them. I tested that they die to flames.
   16. **Vaccine Goodie**
       1. I tested that Penelope gets 1 vaccine charge when she picks up a vaccine goodie and that they decrease when she tries to use one. I tested that her infection count goes to 0 when she uses one. I tested that nothing happens when she tries to use one when she has 0 charges.
   17. **Gas Can Goodie**
       1. I tested that Penelope gets 5 flamethrower charges when she picks up a gas can goodie and that they decrease when she tries to shoot a flame. I tested that up to 3 flames in her direction are made and that she can’t shoot anymore flames when she has no flamethrower charges.
   18. **Landmine Goodie**
       1. I tested that Penelope gets 2 landmine charges when she picks up a landmine goodie and that they decrease when she tries to use one. I tested that she places a landmine when she presses tab. I tested that nothing happens when she tries to use one when she has 0 charges.
   19. **Landmine**
       1. I tested that landmines have 30 safety ticks when deployed. I tested that they create a pit in its location after and 9 flames surrounding it in a 3x3 square. I tested that they are triggered when a human or zombie steps on them and that they are triggered when a flame is shot on them. I tested that the flames created by one of them could chain onto other landmines triggering them. I tested that the flames created by a landmine can’t overlap with a wall or exit.