Henry Trinh

Project 3

**Zombie Dash Report**

**Functions in Classes:**

**StudentWorld:**

**virtual int init();** : The init function is a virtual function derived from a pure virtual function in GameWorld.h that initializes each level in ZombieDash. This function works by putting the level name in proper name format for each level the StudentWorld initializes. The function then checks to see if the file is not found, which returns the player winning, the file has a bad format, which returns a level error, or if the level is loaded successful, where the function checks every coordinate in the level file and constructs the proper objects into the game. The function also initializes the number of goodies that Penelope has to 0 and displays the game stats on top of the window.

**virtual int move();** : The move function is a virtual function derived from a pure virtual function in GameWorld.h that initializes each level in ZombieDash. This function is responsible for updating the game every tick. It calls for each actor and Penelope to do their actions, and if the player dies, returns that the player dies, and if the level is finished, return that the level is finished. The function also iterates through a vector of all Actors and deletes any actor that is not alive anymore as well as the player, where the function decreases a life. The move function also updates the game stats.

**virtual void cleanUp();** : The cleanup function is a virtual function derived from GameWorld.h that cleans up a level when the level is finished or ended. It deletes the player and deletes each Actor using a while loop.

**void addActor(Actor\* obj);** : This is a mutator function that adds an actor to a vector of Actors whenever it is called.

**void setFinishedState(bool n);** : This is a mutator function that allows functions outside of StudentWorld to be able to set the game to finish (The Exit class uses this).

**void increaseVaccines(int n);** : This is a mutator function that increases the number of vaccines that Penelope has by any amount.

**void increaseGas(int n);** : This is a mutator function that increases the number of vaccines that Penelope has by any amount.

**void increaseLandmines(int n);** : This is a mutator function that increases the number of vaccines that Penelope has by any amount.

**bool isFinished() const;** : This is an accessor function that returns whether the game is finished or not.

**int getVaccines() const;** : This is an accessor function that returns the number of vaccines that the player is holding at the time it is called in the current level.

**int getGas() const;** : This is an accessor function that returns the number of flames that the player is holding at the time it is called in the current level.

**int getLandmines() const;** : This is an accessor function that returns the number of landmines that the player is holding at the time it is called in the current level.

**Penelope\* getPenelope() const;** : This is an accessor function that returns the player.

**double Euclidean\_distance(int x1, int y1, int x2, int y2) const;** : This is a helper function that returns a double. This function takes in 4 parameters, which are the x and y coordinates of two different objects. The function returns the Euclidean distance between two objects.

**bool CitizensGone() const;** : This is a helper function that iterates through a vector of Actors and checks if there is one citizen still alive. If there is at least one citizen still alive, the function returns false. Otherwise, true.

**Actor\* doesOverlapCitizen(int x, int y) const;** : This is a helper function that checks if, at a certain coordinate, an object overlaps with a citizen. The function calculates the center pixel coordinate of the object and for each citizen in Actors, does the same too. The function then calculates the Euclidean distance using the corresponding helper function and sees if the distance is less than or equal to 10. If so, it returns that citizen, if not, it returns nullptr.

**Actor\* doesOverlapKillables(Actor \*a) const;** : This is a helper function that checks if, at a certain coordinate, an object overlaps with a killable actor. The function calculates the center pixel coordinate of the object and for each killable actor in Actors, does the same too. The function then calculates the Euclidean distance using the corresponding helper function and sees if the distance is less than or equal to 10. If so, it returns that killable actor, if not, it returns nullptr.

**bool OverlapKillables(Actor\* a) const;** This is a helper function that checks if, at a certain coordinate, an object overlaps with a killable actor. The function calculates the center pixel coordinate of the object and for each killable actor in Actors, does the same too. The function then calculates the Euclidean distance using the corresponding helper function and sees if the distance is less than or equal to 10. If so, it returns true, if not, it returns false.

**bool doesOverlapPenelope(int x, int y) const;** : This is a helper function that checks if, at a certain coordinate, an object overlaps with Penelope/the player. The function calculates the center pixel coordinate of the object and Penelope. The function then calculates the Euclidean distance using the corresponding helper function and sees if the distance is less than or equal to 10. If so, it returns true, if not, it returns false.

**bool doesOverlapWall(int column, int row) const;** : This is a helper function that checks if, at a certain coordinate, an object overlaps with a wall. The function calculates the center pixel coordinate of the object and for each Actor that can’t be killed, does the same. The function then calculates the Euclidean distance using the corresponding helper function and sees if the distance is less than or equal to 10. If so, it returns true, if not, it returns false.

**bool canZombieWalk(int column, int row, Actor\* zombie);** : This is a helper function that checks, if at a certain coordinate, a zombie is able to walk there. For each actor in the vector of Actors that is not the zombie itself and is unable to be walked over, the function determines if the bounding box of the zombie intersects with the bounding box of the given actor. If it does, the function returns false, if not, the function returns true. The function also checks with Penelope, since it is not included in the vector of Actors.

**bool canHumanWalk(int column, int row, Actor\* human);** : This is a helper function that checks, if at a certain coordinate, a citizen or Penelope is able to walk there. For each actor in the vector of Actors that is not the citizen/Penelope themselves and is unable to be walked over, the function determines if the bounding box of the human intersects with the bounding box of the given actor. If it does, the function returns false, if not, the function returns true. The function also checks with Penelope if the human is not Penelope, since it is not included in the vector of Actors.

**Actor\* nearestHuman(Actor\* zombie);** : This is a helper function that returns the nearest human to a zombie. The function first initializes a local pointer to an Actor, closeHuman, as nullptr and an integer, humancount, which keeps track of how many humans are left in the vector of Actors. For each human in Actors, if it is the first actor in the vector, which is determined by if the humancount is equal to 0, closeHuman is set to equal that human. If it is not the first human in the list, the function then determines, using the Euclidean distance formula, if the human is closer to the zombie or if the closeHuman is. If the former, the closeHuman is set to the new human. If, by the end of the iteration of the vector of Actors, closeHuman is still nullptr, which means there are no more humans left, closeHuman is set to the player. If not and the zombie is closer to the human than to the closest citizen, closeHuman is set to the player, also. The function then returns the closest human to the zombie.

**double distanceToPenelope(Actor\* a);** : This is a helper function that returns the Euclidean distance to Penelope from the given Actor.

**double distanceToNearZombie(int column, int row);** : This is a helper function that returns the distance to the nearest zombie to the given coordinates. The function first initializes an integer dis equalling -1. And a bool set to false that keeps track if there is a zombie still alive. For each zombie in the vector of Actors, if the bool is set to false, the function sets dis equal to the Euclidean distance from that zombie to the given coordinates and sets the bool to true.. If the bool is true, which means the function iterated through at least one zombie already, it determines if the Euclidean distance to the current zombie is less than dis. If so, it sets dis to that Euclidean distance. Once iterated through all zombies, the function returns dis.

**void updateGameStat();** : This is a helper function called by the init and move function of the StudentWorld class used to update the game stats on the top of the window of the game. This function works by first properly formatting the score, making sure there are at least 6 digits in the score, where if the score is less than 6 digits, the function fills in leading 0’s. If the score is negative the function also correctly formats a negative sign in front of the score. The function then formats the Level, the amount of lives left, the number of current vaccines, the amount of current flames, mines, and the infection level of Penelope. After this, the function then stores all this information properly formatted into a single string and calls a function with the string from GameWorld.h named setGameStatText to update the game stats.

**Actor: public GraphObject**

**virtual void uponDeath() = 0;** : This is a pure virtual void function because each subclass has its own unique mechanic that triggers when it dies, while Actor itself does not have a mechanic.

**virtual void doSomething() = 0;** : This is a pure virtual void function because each subclass has its own actions to perform every ticks, while Actor itself is an abstract class, so it does not have its own actions to perform.

**void setAliveStatus(bool status);** : This is a helping mutator function that takes in a bool(true/false) as a parameter and sets whether or not an Actor is alive. This is to help other functions kill an Actor when the Actor dies.

**bool isAlive() const;** : This is a constant accessor function that determines whether or not an Actor is alive.

**StudentWorld\* get\_sworld() const;** : This is a constant accessor function that returns the StudentWorld that the Actor belongs to in order to access the StudentWorld functions.

**bool isAbleToWalkOver() const;** : This is a constant accessor function that returns whether or not the Actor is able to be walked over by zombies/citizens/Penelope.

**bool isLiving() const;** : This is a constant accessor function that returns whether or not the Actor is a human- citizens and Penelope.

**bool isKillable() const;** : This is an accessor function that returns whether or not the Actor is able to be killed by something- goodies, humans, zombies, landmines.

**bool isZombie() const;** : This is an accessor function that returns whether or not the Actor is a zombie.

**virtual int getInfectCount() const { return -1; }** : This is a virtual function because it will be used later on to call on a Human.

**virtual bool isInfected() const { return false; }** : This is also a virtual function used to call on a Human.

**virtual void increaseInfection() {}** : This is also a virtual function used to call on a Human.

**virtual void setInfected(bool n) {}** : This is also a virtual function used to call on a Human.

**virtual void setInfectCounter(int n) {}** : This is also a virtual function used to call on a Human.

**WalkingThing: public Actor**

**bool isParalyzed();** : This is a bool function that increases the ParalysisTick by one every tick and returns true when the tick is even and false when the tick is odd. This is used with Citizens and Zombies to determine which ticks they are paralyzed for.

**Human: public WalkingThing**

**virtual int getInfectCount() const;** : This is a virtual function from the Actor class that returns how much a human is infected.

**virtual bool isInfected() const;** : This is a virtual function from the Actor class that returns whether or not a Human is infected.

**virtual void increaseInfection();** : This is a virtual function from the Actor class that increases the infection of a human.

**virtual void setInfected(bool n);** : This is a virtual function from the Actor class that can change if a human is infected or not.

**virtual void setInfectCounter(int n);** : This is a virtual function from the Actor class that can set how infected a human is.

**Zombie: public WalkingThing**

**virtual ~Zombie();** : This destructor calls, when killed by another object in the game, a sound to play when the zombie dies.

**void computeVomitCoordinates(int& x, int& y) const;**  : This is a helper function that determines, for the corresponding direction and position of the zombie, the coordinates of where vomit would be spawned, which would be SPRITE\_HEIGHT or SPRITE\_WIDTH away depending on direction, and stores it in the parameters.

**bool ZombieVomit();** : This is a helper function that uses computeVomitCoordinates to determine what coordinates vomit would appear on. The function then determines if a citizen or penelope would overlap these coordinates and if they do, the function has a 1 in 3 chance to create a new vomit at that location and play the sound of the zombie vomiting. The function returns true if the zombie vomited and false if not.

**int getMovePlan() const;** : This is a helper function that returns the move plan distance that zombie has left.

**void setMovePlan(int distance);** : This is a helper function that sets the MovePlanDistance of a zombie to any distance given in the parameters.

**void setRandomDir();** : This is a helper function that sets the zombie’s direction randomly to either up, down, right or left. This is used in a Dumb Zombie or when a Smart Zombie has nothing to track on to.

**void moveZombie();** : This is a helper function that is called when a zombie’s direction has already been determined and uses the current direction to move a zombie a tick in that direction if the zombie is able to walk in that coordinate. If the zombie is blocked, it resets the MovePlan to 0.

**Goodie: public Actor**

**virtual void doSomething();** : This is a virtual void function that is derived from the pure virtual from the Actor class. All goodies use this doSomething function in addition to the actions that they have. This function checks to see if the goodie overlaps with Penelope, and if it does increases the player’s score, plays the sound SOUND\_GOT\_GOODIE and sets the the goodie to not alive.

**Projectile: public Actor**

**virtual void doSomething();** : Most projectiles have a limited time they are alive when they are spawned into the game in addition to the actions they must perform. This function keeps track of how long they are alive for. The function increases the timer by one every tick and when the timer reaches 2, the function sets the projectile to not alive.

**void Kill();** : This is a helper function used by most projectiles that have the ability to kill an Actor. The function determines if the projectile overlaps with a killable actor in the vector of actors. If the actor is a citizen, the function decreases the player’s score by 1000 and plays the sound of a citizen dieing. It then sets the actor to not alive and calls the uponDeath function to carry out whatever task it needs to do when the corresponding actor dies. The function also determines if the projectile overlaps with Penelope and if it does, sets Penelope to not alive and plays the sound of the player dieing.

**Penelope: public Human**

**virtual void doSomething();** : This is the main virtual function derived from the pure virtual in class Actor that Penelope uses to carry out her actions. The function first checks if Penelope is infected, and if she is, increases the infection, and if her infection reaches 500, kills her and plays the sound of the player dieing. If Penelope is not dead, the function receives a player input. For pressing tab, if the player has landmines, will spawn a landmine at Penelope’s location and decrease the amount of landmines she has. For enter, if the player has a vaccine, resets the infection count to 0 and sets Penelope’s infected status to false. For space, if Penelope has a flame charge, spawns three flames in front of Penelope using her direction and location coordinates if it does not overlap a wall. For each direction key input, the function sets the direction of Penelope using getDirection from GameObject if not done so already and moves Penelope 4 pixels in the corresponding direction if humans can walk into the associated coordinate.

**Citizen: public Human**

**virtual void doSomething();** : This function is the main function that the Citizen does every tick. The function sees if the citizen is infected and if so, increases the infection. If the infection reaches 500, the function plays the sound of zombie being born, decrease the player’s score by 1000, and spawns a zombie, having a 30% chance of being a smart and 70% chance of being a dumb, determined by the randInt function, and sets the status of the citizen to not alive. If the citizen does not die, the function calls isParalyzed to see if the citizen is paralyzed on that tick. If not paralyzed, the function determines if Penelope is closer to it than the nearest zombie using the associated functions from StudentWorld and if Penelope is close than 80 pixels, the function determines the closest direction to Penelope and moves 2 pixels in the direction. If the distance to the closest zombie is less than 80, the citizen decides to calculate the best direction to move away from the zombie without being blocked by anything that can block the citizen’s path and moves 2 pixels in that direction. If there is no path to get away from the zombie, the citizen stays still.

**Wall: public Actor**

**virtual void doSomething()** : This function has no lines of code inside of it because the wall does nothing but look pretty.

**Exit: public Actor**

**virtual void doSomething()** : The job of the exit is to serve as the end point of a level. For each tick, the function checks to see if the exit overlaps with a citizen, and if it does, the function increases the score by 500, and sets the citizen’s status to not alive in order for the StudentWorld to remove it from the game. It also plays a sound of the citizen being saved. If all the citizens are dead or saved and Penelope overlaps with the exit, the function sets the game to finished.

**DumbZombie: public Zombie**

**~DumbZombie();** : When the zombie is destructed by the game, and by that time it is not alive, meaning that the dumb zombie was killed by something, it increases the score by 1000.

**virtual void doSomething();** : This function carries out the main actions for every tick. The function calls isParalyzed to check if the zombie is paralyzed on the current tick. It also checks to see if the dumb zombie vomited on a human, and if it did, returns so that the zombie does nothing else. Otherwise the function checks to see if the movement plan of the zombie has reached 0 and if it does, uses randInt to set a new movement plan distance for the zombie and sets a random direction for the zombie. Then, the function calls moveZombie to move the zombie 1 pixel in the associated direction.

**virtual void uponDeath();** : The dumb zombie has an action to do when it dies, which is to spawn a vaccine randomly by chance. This helper function carries out this task when the zombie dies. In order to do this, the function uses randInt to determine a 10% chance that the zombie drops a vaccine. If the dumb zombie does drop a vaccine on death, the function determines a random direction for the vaccine to spawn and spawns it SPRITE\_HEIGHT or SPRITE\_WIDTH away from the zombie in the associated direction if there is nothing overlapping those coordinates. This function is called when the zombie dies by a flame, landmine, or pit.

**SmartZombie: public Zombie**

**~SmartZombie();** : When the smart zombie’s destructor is called and the zombie is not alive, which means it was not deleted during the destruction of the current level, the destructor of the base class Zombie is called, causing a sound of the zombie dying and increasing the score by 2000 after.

**virtual void doSomething();** :This virtual function contains the main actions of the smart zombie for every tick. The function checks to see if the zombie is paralyzed in the current tick and if the zombie vomited in the current tick. If the smart zombie did not do either, it carries out the smart zombie’s movements. If the movement plan of the zombie is 0, it first sets a new movement plan using randInt. Since the smart zombie is smart and can track down humans, if the Euclidean distance to the nearest human is greater than 80, the function calls SetRandomDir to give a random direction. Otherwise, the zombie can track a human. In order to do this, the function determines the direction that would bring the zombie closer to the human and sets its direction to that direction. After the direction has been set or if the direction was already set, the function calls moveZombie to move the smart zombie in the associated direction.

**Vaccine: public Goodie**

**virtual void doSomething();** : The vaccine is a goodie used to cure Penelope of her infection. In order to do so, this function first calls the virtual function doSomething from the base class Goodie in order to allow the Vaccine to be picked up. The function then determines if it overlaps Penelope and if it does, increase Penelope’s vaccine count by 1.

**GasCan: public Goodie**

**virtual void doSomething();** : The gas can is a goodie used to allow Penelope to shoot flames. In order to do so, this function first calls the virtual function doSomething from the base class Goodie in order to allow the gas can to be picked up. The function then determines if it overlaps with Penelope and if it does, increases the flames for Penelope by 5.

**LandmineGoodie: public Goodie**

**virtual void doSomething();** : The landmine goodie is a goodie used allow Penelope to set down landmines. In order to do so, this function first calls the virtual function doSomething from the base class Goodie in order to allow the landmine goodie to be picked up. The function then determines if it overlaps with Penelope and if it does, gives her 2 landmines to use.

**Flame: public Projectile**

**virtual void doSomething();** : The flame is a projectile that is either spawned by Penelope or by a landmine being set off by another flame or a zombie/human stepping on it. The function calls the base class Projectile’s doSomething to give it a lifespan of 2 ticks. It then checks to see if the flame is alive after calling that. If it is, it calls the Projectile’s Kill function to be able to kill killables.

**Landmine: public Actor**

**virtual void doSomething();**  : The landmine is a “projectile” that is spawned when Penelope puts down a landmine after she obtains one from a landmine goodie. The landmine has an inactive period of 30 ticks, in which time the landmine does not activate. In order to do so the landmine has its own private integer called safetyTicks, which initializes with 30, and a bool called Acive to determine if the landmine is active to be blown up. For each tick, the safetyTicks decrease and when it reaches 0, the bool Active is set to true, which allows the landmine to be blown up. When it is active, if it overlaps with killabls or Penelope the function sets the landmine to not alive and calls its uponDeath function.

**virtual void uponDeath();** : The landmine uses the uponDeath function that select classes have in order to spawn a pit and a ring of flames around the landmine when it blows up. When this is called, the function spawns a ring of flames around the landmine if the flames do not overlap with a wall. It then spawns a pit where the landmine was.

**Pit: public Projectile**

**virtual void doSomething();** : The pit is similar to a wall in its function, it sits there, but it is able to be walked on. The function just calls the Kill function from the class Projectile to kill anything that walks on it.

**Vomit: public Projectile**

**virtual void doSomething();** : The vomit is unlike the other projectiles in the sense that it does not kill anything, so it does not call the Kill function from the class Projectile. The doSomething function calls the doSomething from the class Projectile to give it a lifespan of 2 ticks. If the vomit is still alive and it overlaps with a citizen or Penelope, the function calls the corresponding sound of the human being infected and sets the infected status of them to true.

**Known Bugs:**

* Sometimes on rare occasions, the zombie can phase through into Penelope, but it is hard to replicate. This has happened only once to me in testing.
* Sometimes the citizen when pressed against the player, for example when pressing on the player from below, it does not allow the player to move left or right, but only up. This has happened a few times.

**Assumptions and Design Decisions:**

* When a citizen is running away from a zombie and is next to wall, for example, when a wall is above the citizen and a zombie is below them, the citizen does not decide whether to move left or right.

**Testing Classes:**

**Penelope:** In order to test Penelope, I used the arrow keys to move her around and test whether or not she is blocked by a wall. I also obtained goodies for Penelope to use and tested whether the goodies worked, such as the flame shooting and landmine dropping and vaccine resetting the infected count. I also tested her dieing to multiple objects such as infection, pits, and landmines.

**Citizen:** For citizen, I subjected them to several cases where they would die such as putting them into a pit, shooting flames at them, and having landmines blow up on them. I also tested them onto the exit by leading them to one and seeing if it worked. To test whether the citizens ran away from zombies, I observed the citizen running away from zombie or towards Penelope when I placed her near them. I also tested to see when they got vomited by a zombie, they would turn into one eventually.

**Wall:** To test wall, I shot flames at it and made sure no actor could run inside the wall or spawn anything on top of the wall. I also blew up a landmine next to one to see if it prevented flames from spawning.

**Exit:** To test the exit I subjected it to many things such as flames and landmines to make sure the exits could not die. I also ran citizens onto it and Penelope to make sure the exits worked. I had a zombie vomit on the player to see if the exit could not block the vomit.

**Dumb Zombie:** To test the dumb zombie class, I had the dumb zombies move around to see if they moved around aimlessly. I also made sure that they could not walk over unwalkable things such as other zombies, humans, and walls. I also made sure they could vomit on humans that were right in front of them. Also, I killed zombies until they dropped a vaccine to test that feature.

**Smart Zombie:** In order to test the smart zombie class, I had the zombies chase the player, me, and the citizens. I also made sure they were still able to die by various methods, such as the pit, landmines, and flames. I had also made sure they can also vomit onto humans and that they could not run into the walls.

**Vaccine:** In order to test the vaccine, I made sure that the player can pick up the vaccine and that the vaccine count displayed went up. I also made sure that it would reset the infection to 0 and disinfect Penelope when used.

**Gas Can:**  Testing the gas can is similar to the Vaccine. I made sure it can be picked up and the corresponding count went up. I also made sure Penelope can shoot flames when she picked up one.

**Landmine Goodie:** Like the other goodies, I made sure that it can be picked up and the count for it went up. I also made sure Penelope can spawn the Landmine after picking it up.

**Flame:** I made sure that shooting the Flame can kill killable things, such as goodies, zombies, humans, and Penelope. I made sure that it was blocked by walls and the exit.

**Landmine:** In order to test landmine, I made sure that by standing still over it for a second, the Landmine activating timer worked and that I would be killed after it went active when stepping on it. I made sure that the flames produced by the landmine would be blocked by walls and that the pit would spawn. I made sure that the landmine can be activated to explode after shooting flames or if a human/zombie stepped on it.

**Pit:** I made sure that anything that stepped on the pit would die.

**Vomit:** I made sure that when only a human stepped in front of a zombie, it would be spawned on the human. I also made sure that the infection would kill Penelope from the vomit and that the citizens would eventually become either a dumb zombie or smart zombie after being vomited on for a while after. I also made sure it was not blocked by an exit.