1.

**Actor Functions:**

Virtual Actor\* sValidMove(Direction d)

If there is no object obstructing it, or the object in the desired movement spot is something that cannot block movement, then return nullptr

Return pointer to blocking actor

the function is in the Actor class because many of the derived classes call the base move function along with additional specifications in their own implemented functions, thus it is not pure virtual but needs to be virtual so it can be overwritten

virtual void doSomething()=0

It is in this class because all actors need to do something in each tick. It is not just virtual because there is no base behavior that needs to be called by every class

Void setActive

Intuitive

Virtual int priority() return 0;

It is in this class because by default, it returns 0. Other classes may have higher priority so they overwrite this function.

Virtual bool hittable() return false;

It is in this class because all Actors need to say if it can be hit. It is virtual because some classes need to overwrite the function and perform some action when hit

StudentWorld\* world() return world pointer

Intuitive

Void deltaDir(int& x, int&y, Direction d)

Changes the x and y coordinates into a desired spot to move the actor

No need for the function to be overwritten. Performs the same task for all classes.

Virtual void fill()

Does nothing by default

Is only in the actor class because the boulder needs to fill the hole while only having an actor pointer

**Player Functions:**

void doSomething

if player inputs directional key and player call isValidMove is nullptr, move to that spot

if player inputs space, check if it can fire a bullet then push one into the list

if player inputs escape, decrement a life

Intuitive why a player doSomething is in the player class

Bool hittable

Decrements health

If health is less than or equal to zero, set player to dead

Intuitive location

Actor\* isValidMove(Direction d)

Call Actor isValidMove

If Actor pointer is nullptr, return nullptr

Else if Actor pointer is a boulder and the boulders’s isValidMove is nullptr, return nullptr

Return actor pointer.

Intuitive location

Void addAmmo

adds ammo

only Actor with limited ammo

int getAmmo

return ammo

only actor with limited ammo

int getHealth

return health

not all actors have health, so it doesn’t make sense to store health in the actor class

int priority

return 2

if the player occupies the same spot as another object that allows a bullet to pass through, the bullet’s isValidMove should return the player, not the other object.

**Robot Functions:**

Bool hittable

Decrements health by 2

If the robot is dead, play sound and set to inactive

Descendants of robot call this function and add on additional specifications. Snarlbots will not overwrite this function.

Void setHealth

Intuitive

bool shoot

if obstacleCheck in the direction of movement, return true

function use is the same for all descendant classes

int getHealth

return health

intuitive

bool obstacleCheck

go in the direction passed in and see if theres any actors in the way that would block a bullet

if so, return false

return true

descendants of robot class can implement the same methodology.

Int priority

Return 2

Same reasoning for the player implementation of the function.

**Snarlbot Functions:**

Void doSomething

If shoot, then push a bullet and check if the spot is occupied

Else, if it can move in the direction it points in, then move

Else turn around

Intuitive location

Bool hittable

Robot::hittable();

If it dies, then increase score by appropriate amount

Hittable has this extra implementation here because each robot gives a different point value

**Kleptobot Functions:**

Void doSomething

If it is on a goodie and it can pick it up, then 10% chance of taking it

Else if it can move in the current direction and is less than the number of moves possible, move

Else, select new direction and move in that direction if possible

Implementation is in kleptobot so that Angrykleptobot can call the same method.

Void decDistance

Decreases distances it should travel in the direction

Angry kleptos need this too

Void setDistance

Sets distance it should travel in the direction

Angry kleptos need this too

Int getDistance

Intuitive

Angrys need this too

Hittable

Calls robot hittable function

If health is less than or equal to 0, set to dead, add points, and drop goodie if it’s holding one

It is in here because the angryklepto uses the same method but adds 10 extra points more than this one

Int tickMove

Returns the turn the kleptobot should move on

**AngryKleptobot Functions:**

Void doSomething

If robot::shoot(), then spawn in the bullet and check if it hits an actor

Else, call kleptobot::doSomething()

Bool hittable

Call kleptobot::hittable

If health is less than 0, add 10 points

When kleptobots die, they add 10 points so the angry needs to add another 10 to make it 20

**Factory Functions:**

Void doSomething

If there are less than 3 kleptos around and none on it and satisfies the random condition

Create a corresponding kleptobot on the factory

Factory works the same with angry bots and regular ones

Int priority

Return 1

Factory returns over other objects, but a robot/player will have higher priority

Bool hittable

Return true

Intuitive

**Bullet Functions:**

Actor\* isValidMove

Call base actor is valid move pointer = p

If p is a hole, or is nullptr then return nullptr

Return p

Intuitive location

Void doSomething

if the current spot is occupied and is hittable, then kill the bullet

else move the bullet

if the new spot is hittable then kill the bullet

intuitive location and

**Exit Functions:**

Void doSomething

If it is invisible and all the jewels have been collected

Play sound

Set visible

If visible and player is on the same spot

Set game to finished

**Wall Functions:**

Void doSomething

Does nothing

Int priority

Return 1

Bool hittable

Return true

**Boulder Functions:**

Bool hittable

Decrement health

If health is less than or equal to zero, set it to inactive and invisible

Return true

Actor\* isValidMove

Actor base validmove =p

If p is nullptr, Return nullptr

Else if p is a hole, fill p and set the boulder to inactive; return nullptr

Return p

Needs extra base code because it can move into holes

Int priority

Return 2

Void doSomething

Do nothing

**Hole Functions:**

Void doSomehing

If filled, set to inactive and invisible

Void fill

Set to filled

Only class that uses the function

**Goodie Functions:**

Void doSomething

Sets to invisible and inactive and plays sound

All goodies perform this function, so it is in the base goodie class

**Jewel Functions:**

Void doSomething

if the player is on the same spot,

goodie::doSomething

decrease jewel count and increase score

**Heart Functions:**

Void doSomething

if player is on the same spot

goodie::doSomething

increase lives and increase score

**Ammo Functions:**

Void doSomething

If player is on the same spot

Goodie::dosomething

Increase ammo and increase score

**RestoreHealth Functions:**

Void doSomething

If player is on the same spot

Goodie::dosomething

Increase health and increase score

2. There are no known bugs in the code at the moment

3. One assumption that was made is that the order in which objects are displayed in a single spot (ie, one object on top of another) does not matter as long as the program can correctly interact with all the objects on that spot.

4.

**Player:**

The player class was tested through the following scenarios

1. Directional movement
   1. Player was moved in any cardinal direction into an empty space
   2. Player was attempted to move in any cardinal direction into a blocked space (ie, occupied by a robot, hole, wall, unmovable boulder, factory)
      1. Player was still able to turn into the inputted direction
   3. Player moved a boulder into an empty spot. Boulder moved a spot in response
   4. Player moved a boulder into a hole. Boulder moved a spot in response and the hole disappeared.
   5. Player moved into a bullet. Player was hit by the bullet.
2. Space
   1. Player spawned a bullet into empty space and the bullet moved accordingly in the correct direction
   2. Player shot a bullet into the wall
   3. Player spawned a bullet on top of a robot. Robot was hit by the bullet
   4. Player shot without any ammo remaining. No bullet was spawned
3. Escape
   1. Player lost a life when escape was pressed with more than 1 remaining life.
   2. Game ended when escape was pressed with only one remaining life

**Wall**

The wall class was tested through the following scenarios

1. Hittable test
   1. A bullet was shot at the wall. The bullet was deleted and the wall remained in place.
2. Immovable test
   1. Tested if actor classes could move through the wall, with the wall blocking all of them.

**Boulder**

The boulder class was tested through the following scenarios

1. Movement Test
   1. Boulder to empty space, refer to Player:1:c
   2. Boulder to hole, refer to Player:1:d
   3. Boulder to Boulder. Boulder was unable to move because its path was blocked by another boulder.
2. Damage test
   1. Player shot bullets at the boulder. Boulder was deleted after being hit several times

**Hole**

The hole class was tested through the following scenarios

1. Blocking Test
   1. Actors attempted to move into the hole but were unable to cross
   2. Bullets were shot across the hole and passed without error
   3. Boulders were able to move into holes
2. Deletion test
   1. Boulders filled holes, refer to Player:1:d. When the boulder filled the hole, the hole’s doSomething deleted the hole from the list

**Bullet**

The bullet class was tested through the following scenarios

1. Blocking Test
   1. Bullets were shot at walls and factories. Neither walls nor factories could be destroyed by bullets. However, the bullets were deleted on impact.
   2. Bullets were shot at robots, players, and boulders. All objects took damaged from the bullets and were appropriately deleted on impact. Bullets also disappeared after dealing damage.
   3. Bullets were shot at goodies, jewels, and holes, and exits. Bullets were able to pass over these actors without problem.
   4. Bullets were shot at other bullets. Both were able to pass through one another without issue.
2. Damage Test
   1. Bullets were spawn onto robots to test if they dealt damage on creation if they were made in a spot occupied by another actor.
   2. The player ran into the bullet after the bullet moved to test if the player would take damage from moving into the bullet.
   3. The bullet moved into a hittable object to test if it would deal damage on impact.

**Exit**

The exit class was tested through the following scenarios

1. Player Occupation Test
   1. Exit had no response to the player occupying the same spot if not all the jewels had been collected
   2. Exit ended the level once the player occupied the same spot once all the jewels had been collected
2. Other Actor Occupation Test
   1. Tested if other actors could pass through the exit without problem
   2. Actors treated the exit as if it were simply an open space
3. Exit Revel Test
   1. Exit revealed once all the jewels were collected
   2. Reveal code only fired once ( at the time when the jewels were revealed)

**Extra Life Goodie**

The Extra Life Goodie was tested through the following scenarios

1. Player Occupation Test
   1. Player walked onto the goodie and gained a life and the appropriate number of points
2. Robot Occupation Test
   1. Robots walked onto the goodie without response from the goodie
3. Kleptobot Occupation Test
   1. Kleptobots walked onto the goodie and would on occasion pick up the goodie and delete it from the game board.

**Restore Health Goodie**

The Restore Health Goodie was tested through the following scenarios

1. Player Occupation Test
   1. Player walked onto the goodie and restored its health to full capacity and gained the appropriate number of points
2. Robot Occupation Test
   1. Robots walked onto the goodie without response from the goodie
3. Kleptobot Occupation Test
   1. Kleptobots walked onto the goodie and would on occasion pick up the goodie and delete it from the game board.

**Ammo Goodie**

The Ammo goodie was tested through the following scenarios

1. Player Occupation Test
   1. Player walked onto the goodie and gained 20 ammo and the appropriate number of points
2. Robot Occupation Test
   1. Robots walked onto the goodie without response from the goodie
3. Kleptobot Occupation Test
   1. Kleptobots walked onto the goodie and would on occasion pick up the goodie and delete it from the game board.

**Snarlbot**

The Snarlbot was tested through the following scenarios

1. Shooting Test
   1. Player walked into the line of sight of the snarlbot and the snarlbot shot at the player
   2. Player walked into the same row as the snarlbot but had an obstacle inbetween and the snarlbot did not shoot at the Player
   3. Snarlbot performed no other actions on the turn that it shot a bullet
2. Movement Test
   1. Snarlbot walked in a straight line until it hit an obstacle and then turned around
   2. Snarlbot only acted on occasional frames and all snarlbots would act on the same tick.
3. Damage Test
   1. Upon destruction, snarlbots would play the appropriate sound and increase the points of the player

**Kleptobot**

The Kleptobot was tested through the following scenarios

1. Movement Test
   1. Kleptobots would walk in a straight line for a random number of spots when unobstructed and then select a random direction to turn.
   2. When obstructed, the kleptobot would turn to an unobstructed direction and move
   3. When there were no unobstructed directions of movement, Kleptobots would continue turning without moving
2. Action Test
   1. When walking onto goodies, there was a change that the kleptobot would pick up the item.
3. Death Test
   1. When a Kleptobot died, if it was holding a goodie it would drop it on the spot it died on
   2. If the Kleptobot didn’t hold a goodie, it would just delete itself.
   3. The Kleptobot correctly incremented the game score.

**Angry Kleptobot**

The Angry Kleptobot was tested through the following scenarios

1. Shooting Test
   1. Refer to same parameters as Snarlbot:1. Similar behavior occurred in both robot types.
2. Movement Test
   1. Refer to Kleptobot:1
3. Action Test
   1. Refer to Kleptobot:2
4. Death Test
   1. Refer to Kleptobot:3

**Kleptobot Factory**

The Kleptobot Factory was tested through the following scenarios

1. Spawn Test
   1. Kleptobot Factories would spawn on occasion. They would create the appropriate kleptobot type on top of the factory.
   2. When occupied by a kleptobot, factories would not create new Kleptobots.
   3. When there were more than 3 kleptobots within a 3x3 grid around the factory, the factory would not produce kleptobots.
   4. Kleptobot factories would produce the appropriate sound when spawning a Kleptobot into the game and would correctly push the object onto the Studentworld actor list.
2. Impact Test
   1. When hit by a bullet, the Kleptobot factory would not be damaged, but the bullet would disappear
   2. When there was a kleptobot on the factory, the kleptobot would be hit by the bullet and the robot damage sound would be played.
   3. Kleptobot factories also prevented actors from walking into it