The object hierarchy is as follows:

GraphObject

Actor

Boulder

Robot

KleptoBot

AngryKleptoBot

SnarlBot

Player

Wall

Bullet

Hole

Exit

Pickupable

Jewel

ExtraLifeGoodie

RestoreHealthGoodie

AmmoGoodie

KleptoBotFactory

IShootAtPlayer

AngryKleptoBot

SnarlBot

Actor has the following public methods:

Actor(StudentWorld\* world, int imageID, int startX, int startY, Direction dir = none), world is used to initialize the class’s private pointer to the StudentWorld, and the rest are passed as parameters to the GraphObject base constructor.

A pure virtual destructor, so the StudentWorld can dynamically allocate and delete Actors without worrying about the details of the derived classes’ destructors.

doSomething(), which is the method that StudentWorld calls on every tick. It is pure virtual, as each actor behaves differently on each tick.

getHealth(), which returns the health of the actor. All actors have a health value, so this is not virtual.

getWorld(), which returns a pointer to the StudentWorld. All actors have this pointer, so this is not virtual.

isAlive(), which returns whether or not the actor is alive (health is greater than 0). This is not virtual, as all actors have a health value.

setHealth(int health), which sets the health of the actor. This is not virtual, as all actors have health.

takeDamage(int damage), which is a virtual void function. The default implementation does nothing, as every actor that can take damage has to have a custom implementation of the function, because damageable actors have different behaviors depending on what they are.

The rest are Actor’s public methods are pure virtual, as these methods are actor-specific:

doesBlockMovement(), which describes whether a robot or a player move onto the square that the actor resides on.

doesBlockBoulder(), which describes whether a boulder can occupy the same square the actor resides on.

doesBlockBullet(), which describes whether a bullet can be shot through the square the actor resides on. Is used for robots to determine if they can shoot at the player.

isStealable(), which describes whether a KleptoBot could steal this actor.

countsInFactoryCensus(), which describes whether this actor counts toward the KleptoBotFactory’s check for the number of surrounding KleptoBots.

canBeHitByBullet(), which describes whether a bullet can hit the actor or not.

IShootAtPlayer has the following public methods:

A virtual destructor, as it is a base class.

shoot(StudentWorld\* world, Direction dir, int startX, int startY), which fires a bullet in the specified direction at the specified coordinates. Requires a StudentWorld pointer in order to use StudentWorld’s methods to spawn a bullet and play the appropriate enemy fire sound.

bool tryShootAtPlayer(Actor\* actor), which returns true if it fired a bullet at the player from the actor’s position, and false it if it was not able to fire the bullet because the bullet would be blocked or the player was not in a position parallel to the actor.

Boulder has the following public unique methods:

A constructor, with a StudentWorld pointer, initial X, and initial Y positions for parameters. Passes these parameters onto the Actor constructor, along with the image ID of a boulder and a direction of none. Sets visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. Does nothing.

takeDamage(int damage), which overrides the one inherited from Actor. Sets health to getHealth() - damage.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as boulders do not allow other actors to occupy the same square.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as boulders do not allow other boulders to occupy the same square.

doesBlockBullet(), which overrides the one inherited from Actor. Returns true, as boulders do not allow bullets to travel through them.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns true, as boulders can be damaged by bullets.

Robot has the following public unique methods:

A constructor, with a StudentWorld pointer, image ID, initial X, initial Y, and initial direction parameters. These are passed to the Actor constructor. Initializes the private tick counter that all robots utilize to determine whether to be active on the current tick.

A virtual destructor

doSomething(), which manages the tick counter, and if the current tick is the one the robot should be active on, calls a protected method doWhenActive(), which is overridden by the classes that inherit Robot.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as robots do not allow other actors to occupy the same square.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as robots do not allow boulders to occupy the same square.

doesBlockBullet(), which overrides the one inherited from Actor. Returns true, as robots do not allow bullets to travel through them.

isStealable(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns true, as robots can be damaged by bullets.

KleptoBot has the following public unique methods:

A constructor, with a StudentWorld pointer, initial X, and initial Y positions for parameters. Passes these parameters onto the Robot constructor, along with the image ID of a KleptoBot, and the initial direction of right. Calls a private method to initialize the class, which sets the health, visibility, and initializes the distance counter.

A virtual destructor

takeDamage(int damage), which overrides the one inherited from Actor, as a KleptoBot has unique behavior when it is damaged. Reduces health by the amount specified by damage. If the KleptoBot does not die, play the appropriate sound, otherwise, drop the goodie the KleptoBot stole, if it did steal one, then play the appropriate sound, and increase the score by 10.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns true, as KleptoBots count in the factory census.

AngryKleptoBot has the following public unique methods:

A constructor, with a StudentWorld pointer, initial X, and initial Y positions for parameters. Passes these parameters onto a protected KleptoBot constructor, along with the image ID of an AngryKleptoBot. Sets health to 8 after the KleptoBot constructor has been called.

A virtual destructor

takeDamage(int damage), which overrides the one inherited from KleptoBot. Calls KleptoBot’s takeDamage, and if it has died, adds an additional 10 points to the player’s score, as AngryKleptoBots add 20 points instead of 10 points to the player’s score when dying.

SnarlBot has the following public unique methods:

SnarlBot(StudentWorld\* world, int startX, int startY, Direction dir), whose parameters are passed onto the Robot constructor, along with the image ID of a SnarlBot. Sets health to 10 and sets visibility to true.

A virtual destructor

takeDamage(int damage), which overrides the one inherited from Actor. If the SnarlBot survives the damage, plays the appropriate sound, otherwise, plays the death sound and increases the player’s score by 100.

countsInFactoryCensus(), which returns false, as a SnarlBot does not count in the factory census.

Player has the following public unique methods:

Player(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Actor constructor, along with the player’s image ID and a direction of right. Initializes the player’s ammo count, sets health to the appropriate ammount, and sets visibility to true.

A virtual destructor

shoot(Direction dir, int startX, int startY), which spawns a bullet at the specified location with the specified direction, subtracts from the ammo counter, and plays the appropriate sound.

doSomething(), which gets key presses from StudentWorld and responds appropriately, either by moving in the direction specified by the user’s input, killing the player, or shooting a bullet.

takeDamage(int damage), which overrides the one inherited from Actor. If the player is not killed by the bullet, play the appropriate sound, otherwise, play the player death sound.

addAmmo(int ammo), which adds ammo to the player’s ammo count.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as the player does not allow other actors to occupy the same square.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as the player does not allow boulders to occupy the same square.

doesBlockBullet(), which overrides the one inherited from Actor. Returns true, as the player does not allow bullets to travel through them.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns true, as players can be damaged by bullets.

getAmmo(), which returns the player’s ammo count.

Wall has the following public unique methods:

Wall(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Actor constructor, along with the wall image ID and a direction of none. Initializes health to 1 so the wall isn’t dead and sets visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. Does nothing.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as walls do not allow other actors to occupy the same square.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as walls do not allow boulders to occupy the same square.

doesBlockBullet(), which overrides the one inherited from Actor. Returns true, as walls do not allow bullets to travel through them.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns true, as walls destroy bullets when they are hit by them but take no damage from bullets.

Bullet has the following public unique methods:

Bullet(StudentWorld\* world, int startX, int startY, Direction dir), which passes these parameters onto the Actor constructor, along with the bullet image ID. Initializes health to 1 so the bullet isn’t dead and sets visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. Checks to see if any actors on the current square can be hit by a bullet. If any actors can be hit, apply damage to that actor and kill itself. Otherwise, move to the next square and repeat the check.

doesBlockMovement(), which overrides the one inherited from Actor. Returns false, as bullets do allow other actors to occupy the same square.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns false, as bullets do allow boulders to occupy the same square.

doesBlockBullet(), which overrides the one inherited from Actor. Returns false, as bullets do allow bullets to travel through them.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns false, as bullets cannot hit each other.

Hole has the following public unique methods:

Hole(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Actor constructor, along with the hole image ID. Initializes health to 1 so the hole isn’t dead and sets visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. Checks to see if a boulder is occupying the same position as the hole. If so, it kills itself and the boulder.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as actors cannot walk onto holes.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns false, as boulders can be pushed onto holes.

doesBlockBullet(), which overrides the one inherited from Actor. Returns false, as bullets travel over holes, so a robot could shoot at a player even if a hole is between the two.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns false, as holes are not damaged by bullets.

Exit has the following public unique methods:

Exit(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Actor constructor, along with the exit image ID. Initializes health to 1 so the exit isn’t dead. Does not set visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. If the exit is visible and the player is on the same location as the exit, signal to StudentWorld that the level has been completed.

doesBlockMovement(), which overrides the one inherited from Actor. Returns false, as actors can walk over the exit.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as boulders cannot be pushed onto an exit.

doesBlockBullet(), which overrides the one inherited from Actor. Returns false, as bullets can travel over exits.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns false, as exits are not damaged by holes.

Pickupable has the following public unique methods:

Pickupable(StudentWorld\* world, int imageID, int startX, int startY), which passes these parameters onto the Actor constructor. Initializes health to 1 so the pickupable isn’t dead. Sets visibility to true.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. If the pickupable is visible and the player is standing on the same square, kill itself, play the got goodie sound, and call giveBonus(), a virtual protected function that allows derived classes to define the appropriate bonus.

doesBlockMovement(), which overrides the one inherited from Actor. Returns false, as actors can walk over pickupables.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as boulders cannot be pushed onto pickupables.

doesBlockBullet(), which overrides the one inherited from Actor. Returns false, as bullets can travel over pickupables.

isStealable(), which overrides the one inherited from Actor. Returns true. Pickupables are stealable, except for Jewel.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns false, as pickupables cannot be damaged by bullets.

Jewel has the following public unique methods:

Jewel(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Pickupable constructor.

A virtual destructor

isStealable(), which overrides the one inherited from Pickupable. Returns false, as Jewels cannot be stolen by KleptoBots.

ExtraLifeGoodie has the following public unique methods:

ExtraLifeGoodie(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Pickupable constructor.

A virtual destructor

RestoreHealthGoodie has the following public unique methods:

RestoreHealthGoodie(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Pickupable constructor.

A virtual destructor

AmmoGoodie has the following public unique methods:

AmmoGoodie(StudentWorld\* world, int startX, int startY), which passes these parameters onto the Pickupable constructor.

A virtual destructor

KleptoBotFactory has the following public unique methods:

KleptoBotFactory(StudentWorld\* world, int startX, int startY, bool angry), which passes the first three parameters onto the Actor constructor, along with the KleptoBotFactory image ID, and a direction of none. angry determines if the KleptoBots produced by the factory are angry or not.

A virtual destructor

doSomething(), which overrides the one inherited from Actor. Performs the census in the region defined by the spec, and if there are less than 3 KleptoBots in the region and no KleptoBot on the factory itself, then there is a 1 in 50 chance of it spawning a KleptoBot.

doesBlockMovement(), which overrides the one inherited from Actor. Returns true, as actors cannot walk onto a factory.

doesBlockBoulder(), which overrides the one inherited from Actor. Returns true, as boulders cannot be pushed onto a factory.

doesBlockBullet(), which overrides the one inherited from Actor. Returns true, as bullets cannot travel over a factory.

isStealable(), which overrides the one inherited from Actor. Returns false.

countsInFactoryCensus(), which overrides the one inherited from Actor. Returns false.

canBeHitByBullet(), which overrides the one inherited from Actor. Returns true, although a factory does not take damage from the bullet that hits it.

StudentWorld has the following public methods:

StudentWorld(std::string assetDir), which sets the class’s player pointer to nullptr.

init(), which loads the level, sets the bonus to 1000, and checks to see if the level is over 99. If the level is malformed or doesn’t exist, it returns the appropriate values.

move(), which updates the display text, reveals the exit if there are no more jewels, calls all of the actor’s doSomething()’s if they are alive, handles level completion or player death, removes dead actors, adds newly spawned actors to the list of actors, and subtracts from the bonus score per tick.

cleanUp(), which deletes the player, deletes all of the actors in the list of actors, and then clears the list of actors.

addActor(Actor\* actor), which adds an actor to the list of actors.

getBonus(), which returns the bonus score.

getActorsAtXY(int X, int Y), which returns a list of actors at the specified coordinates.

isPlayerAtXY(int X, int Y), which returns true if the player is at the specified coordinates.

getPlayer(), which returns a pointer to the player.

signalCompletion(), which tells the StudentWorld that the level has been completed.

decJewels(), which decreases the count of jewels as tracked by the StudentWorld.

Finished functionality to the best of my knowledge as defined by the spec, and was not able to find any bugs.

One design decision I made in order to achieve some code reuse was to have SnarlBot and AngryKleptoBot inherit from a second base class, which provides methods to the two robots that allow them to determine if they can shoot at the player, and if they can’t, do whatever they’re supposed to do otherwise.

I chose to implement the set of virtual booleans in the Actor base class as they are in order to move away from relying on dynamic\_cast or other such methods to determine if an Actor does not allow a certain behavior, e.g. blocking bullets or boulders. However, I still use dynamic\_cast to determine if an actor is a hole or a boulder.

The base classes Robot and Pickupable implement doSomething(), which then calls a protected method which derived classes define in order to determine actor-specific behavior. As all robots move to the same ticks this avoids duplicating code, and as all Pickupables work similarly this avoids duplicating code, too.

All actors added to the StudentWorld using addActor, are instead added to a secondary list. After doSomething() is called for all actors that existed and were not added to the StudentWorld during the same tick, the actors in the secondary list are moved into the actual list of actors so they become active during the next tick. This fixes bugs regarding bullets appearing to spawn a square past where they are supposed to spawn, and other such bugs.

KleptoBots and AngryKleptoBots steal items by setting the goodie’s visibility to false, moving the goodies when they move, and when they die, they set the visibility of the goodie to true. As such they do not need to keep track of which type of goodie they stole, they just need to hold a pointer to an Actor.

I tested the Actor class indirectly, through checking that all derived classes behaved appropriately.

I tested the Boulder class by seeing if I could push it around and not push it through walls, or pickupables, or the exit, but could push it onto a hole, and whether it blocked robot movement appropriately and whether or not robots would try to shoot at a player through a boulder.

I tested the Robot class by making sure that all derived robots are only active on the appropriate tick.

I tested the IShootAtPlayer base class by making sure that SnarlBots and AngryKleptoBots had the same behavior when trying to shoot at a player.

I tested KleptoBots using the first level, and comparing my bot’s behavior to the sample game’s behavior. In order to test stealing items, I corralled a KleptoBot into a corner where there was an ammo goodie, waited until it was stolen, allowed the KleptoBot to move far from where the goodie originally was, and then killed the KleptoBot to make sure it drops the item where it dies.

I tested AngryKleptoBots using the second level, making sure that the player was shot at whenever the bot could shoot at the player, and otherwise making sure that it behaved the same as KleptoBots with respect to movement and stealing goodies.

I tested SnarlBots using the first level, making sure that boulders blocked their movement and that they wouldn’t try to shoot at the player through a boulder. Using the second level, I made sure they could shoot at a player over a hole if it was the only thing between the player and the SnarlBot.

I tested the Player class by playing the game, making sure that the player could push boulders, shoot bullets, take damage, pick up goodies, and use the exit appropriately.

I tested Walls by making sure things couldn’t walk through them, they blocked movement, stopped bullets, could not be killed, and boulders could not be pushed into them.

I tested Bullets by making sure they only became active the tick after they were spawned, they dealt damage appropriately, could fly over holes, and disappeared when hitting walls, the player, robots, boulders, but not goodies.

I tested Holes by making sure they would disappear when a boulder was pushed on them, otherwise they block movement, and bullets can be shot over holes.

I tested the Exit by making sure it blocks boulders even if it’s invisible, and appears when all jewels are taken, and made sure that bullets can fly over it.

I tested Pickupables by making sure all of the derived classes behaved similarly. Bullets can be shot over them, and if the pickupable is visible then it disappears and gives the appropriate bonus to the player if the player walks over it.

I tested Jewels by making sure they couldn’t be stolen, and if they were all picked up the exit would then appear.

I tested all three goodies by making sure that when they were stolen they would reappear where the KleptoBot was killed.

I tested the KleptoBotFactory by making sure it only spawned KleptoBots if there weren’t 3 in the immediate radius of the factory as defined by the spec. In addition if the factory is hit by a bullet it doesn’t take any damage and makes the bullet disappear. Bullets cannot be shot over the factory, so a robot would not shoot at the player if the factory was between the two.