**1-----Description of classes**

1. **class Actor**
   1. constructor
      * sets the state of every actor
   2. destructor
      * virtual because it has to be
   3. void restoreHealth()
      * to be used by StudentWorld to restore player's health
      * defined here because actor has health private data members so only actor an access these private data members
   4. bool moveActor(Direction d)
      * defined it here because most Actors move
      * returns true is the Actor moved
   5. StudentWorld\* getWorld() const
      * defined here so actors all able to tell StudentWorld about updates in the game
   6. void setDead()
      * defined here because actors can die
      * there is only one implementation of it
   7. bool isAlive() const
      * defined here to let the caller know if the actor is alive
   8. int getHealth() const
      * defined here because only Actor has access to private data member m\_health
   9. virtual void doSomething() = 0;
      * pure virtual because all actors do something different
      * a class of Actor is not allowed to be created
   10. virtual bool canActorMoveTo(int x, int y) const
       * virtual because moveActor() calls this function and different Actors are able to move in different circumstances
       * returns false so that I do not have to define it in every class that doesn't move
   11. virtual void damage()
       * virtual because when different players die different sounds are played
       * decrease the health of an Actor
   12. virtual bool push(Direction d)
       * declared virtual because StudentWorld calls it
       * almost all other Actors cannot be pushed so return false
   13. virtual bool shootThrough(), isWall(), isBoulder(), isActor(), isMoveAbleActor(), isHole(), isGoodie(), isExit(), isKleptobot(), isPlayer(), isFactory(), isPickable()
       * virtual because all redefined in their respective class
2. **class Wall**
   1. constructor
      * sets the state of Wall
   2. destructor
      * empty, does nothing
   3. virtual void soSomething()
      * virtual because it's good practice
      * a wall does nothing
   4. virtual bool isWall() const
      * virtual because good practice
      * redefines to return true
3. **class Exit**
   1. constructor
      * sets state of Exit
   2. destructor
      * virtual because it's good practice
   3. virtual void soSomething()
      * redefines to what an Exit is supposed to do
   4. virtual void isExit() construct, virtual bool shootThrough() const, virtual bool moveThrough() const
      * redefines to return true
4. **class Hole()**
   1. constructor
      * sets state of hole
   2. destructor
      * virtual good practice
   3. virtual void soSomething()
      * redefines to what a Hole is supposed to do
   4. virtual void isHole() construct, virtual bool shootThrough() const,
      * redefines to return true
5. **class Bullet**
   1. constructor
      * sets state of bullet
   2. destructor
      * virtual good practice
   3. virtual bool canActorMoveTo(int x, int y) const
      * redefines to true a bullet can move onto any Actor
   4. virtual shootThrough() const
      * redefines to return true
   5. virtual moveThrough() const
      * redefines to return true
6. **class Factory**
   1. constructor
      * sets state of constructor
      * angry by default is false
   2. destructor
      * virtual good practice
   3. virtual void doSomething()
      * refines to what a Factory does
   4. virtual bool shootThrough()
      * redefines to false
   5. virtual bool isFactory
      * redefines to true
7. **class Pickable**
   1. constructor
      * sets state of pickable
   2. destructor
      * virtual because good practice
   3. virtual void doSomething()
      * pure virtual its a base class for other classes
      * instance of Pickable not allowed
   4. void increasePlayerScore()
      * not virtual because all derived classed increase the player's score
   5. virtual bool shootThrough() const
      * redefined because all derived classes can be shot through
   6. virtual bool isPickable() const
      * redefined to true
   7. virtual bool moveThrough() const
      * redefined because all derived classes can be moved through
8. **class Jewel**
   1. constructor
      * sets state of a Jewel
   2. destructor
      * virtual because good practice
   3. virtual void doSomething()
      * redefined to do what a Jewel is supposed to do
9. **class Goodie()**
   1. constructor
      * sets state of a Goodie
   2. destructor
      * virtual because good practice
   3. virtual void doSomthing()
      * not redefined because it is a base class for other classes
      * an instance not allowed
   4. virtual bool isGoodie() const
      * redefined because all derived classes are Goodies
10. **class Ammo**
    1. constructor
       * sets state of ammo
    2. destructor
       * virtual good practice
    3. virtual void doSomething()
       * redefines to what Ammo is supposed to do
11. **class ExtraLife**
    1. constructor
       * sets state of ExtraLife
    2. destructor
       * virtual good practice
    3. virtual void doSomething()
       * redefines to what ExtraLife is supposed to do
12. **class RestoreHealth**
    1. constructor
       * sets state of RestoreHealth
    2. destructor
       * virtual good practice
    3. virtual void doSomething()
       * redefines to what RestoreHealth is supposed to do
13. **class MoveableActor**
    1. constructor
       * sets state of MoveAbleActor
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * pure virtual not redefined because it's a base class for other classes
    4. virtual bool isMoveAble Actor() const
       * redefined because derived class are MoveableActors
14. **class Boulder**
    1. constructor
       * sets state of Boulder
    2. destructor
       * virual good practice
    3. virtual void doSomething()
       * redefined a boulder does nothing during a tick
    4. virtual bool push(Direction d)
       * returrns true is the boulder was pushed
    5. virtual bool canActorMoveTo(int x, int y) const;
       * redefined to return is a boulder can move to x and y
    6. virtual bool isBoulder()
       * redefined because class is a Boulder
15. **class Shooter**
    1. constructor
       * sets state of Shooter
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * pure virtual because it is an abstract base class for other classes
    4. void shoot(bool thisTick = true) const;
       * not virtual because derived classes can shoot
16. **class Player**
    1. constructor
       * sets state of Player
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * redefined to do what a Player is supposed to do
    4. virtual bool canActorMoveTo(int x, int y) const
       * redefined returns if a player can move
    5. virtual void damage()
       * redefined to play sounds
    6. int getAmmo() const
       * not virtual because only Player has limited ammo
    7. void increaseAmmo(int n)
       * not virtual because only Player can increase its ammo
    8. virtual bool isPlayer() const
       * redefined because it's a Player
17. **class Robot**
    1. constructor
       * sets state of Robot
       * sets number of ticks that the robot can move
       * sets all robots to visible
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * not redefined still pure virtual because its an abstract base class
    4. virtual bool canActorMoveTo(int x, int y) const
       * redefined to see is a Robot can move
    5. bool canShootAtPlayer() const
       * not virtual because dervived classes need to see if they can shoot at the Player
    6. virtual void damge()
       * redefined to play correct sounds
    7. bool canRobotMove()
       * not virtual because all derived Robots need to check if they can move during the given tick
       * return true is Robot can doSomething during current tick
18. **class Snarlbot**
    1. constructor
       * sets the state of Snarlbot
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * refined to what a Snarlbot does during a given tick
    4. void reverseDirection()
       * not virtual because only Snarlbots reverse their direction
19. **class Kleptobot**
    1. constructor
       * sets state of a Kleptobot
       * sets the distance before a kelptobot turns
    2. destructor
       * virtual good practice
    3. vitual void doSomething()
       * redefined to do what a Kleptobot is supposed to do
    4. virtual void damge()
       * redefined to play see if it has a Goodie
    5. void moveKbot()
       * not virtual because all derived classe move the same
    6. bool pickedUpGoodie()
       * not virtual because all derived classes can pick up a goodie in the same way
    7. bool haveGoodie()
       * not virtual because all derived classes need to check if they have a goodie
    8. virtual bool isKleptobot() const
       * redefined because this class is a Kelptobot
20. **class AngryKleptobot**
    1. constructor
       * sets the state of an AngryKleptobot
    2. destructor
       * virtual because good practice
    3. virtual void doSomething()
       * redefined to do what an AngryKleptobot is supposed to do
21. **class StudentWorld**
    1. constructor
       * sets the state of StudentWorld
    2. destructor
       * virtual because good practice
    3. virtual int init()
       * redefined to initilize a level
    4. virtual move();
       * redefined to tell all Actors to do something
    5. virtual void cleanup()
       * redefined to deallocate memory
    6. void jewelPickup()
       * not virtual because only StudentWorld keeps track of the jewels
    7. void addActor(Actor\* newActor)
       * not virtual because only StudentWorld keeps track of Actors
    8. void setFinishedLevel()
       * not virtual because only StudentWorld keeps if the Player finished the level
    9. Actor\* getContentsAt(int x, int y, const Actor\* a) const, bool canBoulderMoveTo(int x, int y) const, bool canPlayerMoveTo(int x, int y) const, bool canRobotMoveTo(int x, int y) const, bool checkBulletSquare(int x, int y) const, void increasePlayerAmmo() const, void restorePlayerHealth() const, bool existsClearShootsAtPlayer(int x, int y, GraphObject::Direction d) const, int countKbots(int x, int y, bool& kBot) const, Actor\* checkGoodie(int x, int y) const
       * not virtual because only StudentWorld keeps track of actors

**2---------List of functionality that I failed to implement**

**I think that I was able to implement all the functionallity**

**3------------Decisions and Assumptions**

1. For a Kleptobot it was not specified that after a Kleptobot moved after hitting an obstruction or when distance it can moved got to zero, if the move should count as a unit distance moved, so after it moves I do not decrement
2. It was not specified if during a new tick if a new Actor is added to my vector if they are allowed to doSomething. New Robots are allowed to. Bullets created during a current tick aren't allowed to.
3. Did not specify when sounds should play, so my reveal exit sound plays at the same time I get my last jewel.

**4---------Testing classes**

For each class I created a level to test each non-abstract class:

* 1. I created a level with walls to see if it can move into walls. Then I added boulders to see if it could push them. This also tested to see boulders are able to be pushed by a Player. Then I created stationary Snarlbots to see if it my Player's hit points were decremented and that he was killed after reaching 0 hit points. This also tested if a Snarlbot recognized whether it had a clear shot at my Player. Then I added Jewels, Goodies, and a Factory to see if it could move into the Jewels and Goodies but not into the Factory. Then I checked to see if my Player can shoot. Then I shot at the Snarlbots to see if my Bullet acted the way it was supposed to and if the Snarlbots would die after reaching zero hits points.