a.1.Class Actor:

virtual void doSomething();// return immediately, it is virtual because the subclasses do different things during a tick and not all the actors have to do something during a tick

StudentWorld\* getWorld() const;//get a pointer of the studentWorld

virtual int getHealth() const;//return 1 since it’s used to judge if an actor is dead, it is virtual because different actors may have different hit points

virtual void pushed(Direction x);//return immediately, it is virtual because a boulder has its own way of being pushed

virtual bool blockActor() const;//return true, but not all the actors will block others, so it is virtual

virtual bool blockBoulder() const; return true, but since a hole will not block a boulder, it is virtual

virtual bool isExit() const; return false, this function is used to find the exit when all the jewels are collected. Only in the exit object will it return true, so it is virtual

virtual bool eatBullet() const; return false, bullets can pass through some of the objects, whereas some actors kill the bullet. It is therefore virtual.

virtual void attacked();//return immediately, not all the actors can be attacked, those can be attacked have different behavior. So it is virtual.

virtual bool isBoulder() const;

return false.It is used to indicate if what blocks the player is a boulder.

Only boulder class will return true; It is a virtual function.

virtual bool eatBoulder() const;

return false.it is used to indicate if the hole is in the front of spot of the boulder being pushed. Only a hole will return true. It is virtual.

virtual void setDead();

set actor’s health to 0. This only set non-agent actors to dead. Agents health are stored in somewhere else.

Virtual void setvisible();return immediately, only the exit class will have to set itself visible during the game. So it is a virtual function.

virtual bool isKlepto() const; return false

it is used to help count the number of kleptoBots near the factory. Only KleptoBot and angryKleptoBot will return true. It is virtual

virtual bool isGoodie() const; return false

it is used to judge if an object can be picked up by kleptoBots,

only return true if it is any of the goodies. The function is virtual

void nextPoint(int &x,int &y,Direction);

set x,y to the next spots in the given direction, it works the same for all the class, so it is not virtual.

2. class mobileActor

virtual int getHealth() const;

derived version of getHealth in the Actor class, return different hit points of agents

virtual void attacked();

derived version of attacked in the Actor class, decrement the hit points by two

virtual void recover();

set the player’s hit points back to 20

Only for Player class, but player class can’t access the variable represents his hit points.

virtual bool eatBullet() const;

derived function of eatBullet in the Actor class, return true because all the agents can kill the bullet.

virtual void setDead();

derived function of setDead in the Actor class, set an agent’s hits point to zero

virtual bool LegalMove(int &x,int &y,Direction dir) ;

return if the next step in the given direction is viable, if it is, update the x,y to the next location

3.class Player

virtual void doSomething();

derived version of doSomething in the Actor class, get the key and behave accordingly, call pushBoulder function before any action

int getAmm() const;

return the number of ammunication, used in the updateText function

void pushBoulder(int x,int y, Direction dir);

judge if a boulder is in the front, if it is, try to push the boulder.

void getAmmo();

player has pick up Ammo, increment ammunication number accordingly

virtual void attacked();

call the mobileActor version of attacked first, and judge if the player is dead or not, behave accordingly

4.class Wall

virtual bool eatBullet() const;

derived class of the eatBullet function in the Actor class

return true because it can kill a bullet.

5. class Boulder

virtual void pushed(Direction x);

derived version of pushed function in the Actor class. Move itself in the direction it is pushed if it not blocked. If in front of it is a hole, kill the itself and kill the hole

virtual bool LegalMove(int &x,int &y,Direction dir,Actor\*& pnt) const;

derived version of LegalMove function in the mobileActor class. The difference is a boulder is not blocked by the hole.

virtual bool isBoulder() const;

derived version of isBoulder function in the Actor class, return true.

6. class Bullet

virtual void doSomething();

derived from the Actor class judge if it is in the same location as an object that kills it, if it is, kill itself and attack that object, if it is not, move to the next spot. Then, again, judge if it is in the same location as an object that kills it, if it is, kill itself and attack that object

bool legalMove(int x,int y, Actor\* &p)const;

the function used to judge if the bullet is in the same location as an object that kills it

virtual bool eatBullet() const;

derived version of eatBullet function in the Actor class, return false, it does not kill itself

virtual bool blockActor()const;

derived version of blockActor function in the Actor class, return false, it doesn’t block Actor, it can attack player.

void moveToADir(int x,int y,Direction dir);

move to the next spot in the given function

7. class Hole

virtual bool blockBoulder() const;

derived from the Actor class, return false, it does not block the boulder

virtual bool eatBoulder() const;

derived from the Actor class, return true, it can kill the boulder

8.class pickUpable

bool blockActor() const;

derived version of blockActor function in the Actor class, return false, it doesn’t block Actor

bool eatenByPlayer() const;

it can be picked up and killed by being on the same spot with a player, it is non-virtual because it is the same for all the pickUpables.

bool something(int score);

it is common part that every pickupable will do, increase the score, play the sound

9. class Jewel

virtual void doSomething();

derived from the Actor class, call something function

virtual ~Jewel();

decrease the jewel count when it is deleted

10. class Exit

virtual bool blockBoulder() const;

derived from the Actor class ,when it is invisible return false, when it is visible return true

virtual bool blockActor() const;

derived from the Actor class ,return false

virtual void setvisible();

derived from the Actor class, set itself visible, play the sound

virtual bool isExit() const;

derived from the Actor class, return true

virtual void doSomething();

derived from the Actor class, if it is visible and is on the same spot as the player, inform studentWorld the level is finished

11.class Goodies(a subclass of pickUpables)

virtual bool isGoodie() const;

derived from the Actor class,return true;

12.class restoreHealthGoodie

virtual void doSomething();

derived from the Actor class, call the something function in the pickUpable class, call the recover function of the player

13.class extraLifeGoodie

virtual void doSomething();

derived from the Actor class, call the something function in the pickUpable class and increment player’s life by one

14. class Ammo

virtual void doSomething();

derived from the Actor class, call the something function in the pickUpable class and increment player’s ammunication by 20

15. class Robot(a subclass of mobileActor)

void setTick(int tick);

calculate the random number

int getTick() const;

return the random number

bool fire(GraphObject::Direction dir);

if the player is in the same line as the robot, fire if it can

bool isObstacle(int x,int y) const;

check if the robot is blocked at the certain spot

virtual void attacked();

derived from the Actor class, play the proper sound regarding if the robot is still alive after the attack

virtual void dying()=0;

it’s pure virtual function, when different robots die, they behave different way.

16. class snarlBot

virtual void doSomething();

derived from Actor class, if it’s the tick to move, move and reset the tick, if it’s not, wait. If player is in the same direction the robot is facing and in the same line, fire the player.

virtual void dying();

derived from the Robot class increase the score

17. class KleptoBots

void distanceBeforeTurn();

generate a random number stands for the distance before turning

void pickupAGoodie();

set the goodie invisible

virtual void doSomething();

derived from the Actor class, if it isn’t the tick to move wait, if it is, call the function someThing

bool getPick() const;

return if the robot has picked up a goodie

int getCrtTick() const;

return the tickcount before the next move

void decCrtTick();

waiting and decrement the tickcount

void decDisBT();

move and decrement the distance before turing

void someThing();

if it’s not the tick to turn, move ahead. If it can’t move or it’s the tick to turn. Randomly turn to a direction and if the robot can’t move in that direction after turning, turning again until it turns back to the first direction

Actor\* getPointer();

Get the pointer of the goodie

virtual bool isKlepto() const;

derived from the Actor class, return true

virtual bool Regular() const;

to distinguish if it’s regular or angry, return true

virtual void dying();

derived from the robot class if it has picked up the goodie, make the goodie visible at the the spot it dies

void setCurrentTick(int tick);

make the current tick count to the random number.

18. class Angry Kleptobots

virtual void doSomething();

derived from the actor class, if it isn’t the tick to move wait, if it is, fire the player in the same line. If there is no fire

action, call something.

virtual bool Regular() const;

derived from the klepto class, return false.

19. class KleptoBotFactory

virtual void doSomething();

derived from the Actorc class, call robotCount and create robots accordingly.

bool robotCount(int x, int y);

count the kleptoBots around the factory, if it’s less than 3, return true.

virtual bool eatBullet() const;

derived from the Actor class, return true.\

20. class studentWorld

void updateDisplayText();

change the text proper to the current tick

void removeDeadObjects();

delete dynamically allocated dead Actors

void reduceLevelBonusByOne();

decrement bonus

bool AllOfJewelsCollected() const;

return if the jewel count is zero

void levelcompleted();

set levelCompleted to true

void creatABullet(int x, int y, Actor::Direction dir);

dynamically allocated a bullet object in the location specified and push the bullet object to the front of the list

void killAHole(Actor \* pnt);

a hole is killed by a boulder. Kill the dynamically allocated Hole object

Player\* getPlayer();

Return the pointer of the player

list<Actor\*> Ref(int x, int y) const;

return the list of the point of the actors that the place specified.

void decJewel();

decrement the jewel count

void createAKleptoBot(int x,int y);

dynamically allocate a KleptoBot at the specified place

void createAngryKleptoBot(int x, int y);

dynamically allocate an angryKleptoBot at the specified place

b. I finished all parts.

c. I pushed the bullet in the front of the list instead of the back.

I let the boulder check if it is on the same spot as the hole

d.

1. test player along with studentWorld:

play all the levels and see if there is difference between the sample game, pay extra attention to the text printed on the screen to check if the player is getting correct score, attacked properly if the bonus is working

2.test boulder:

test all the situations “@B?” with ? as a hole, an empty spot, an invisible exit, a visible exit and another boulder.

3. test Snarlbots

First,let player stand in front of it to test the fire function and push a boulder to test its turning movement. Then check its speed by observation.Use the player to attack it to check if the hit points is right. At last different things around to check if the blocking is functioning

4.test kleptoBots

First check its moving speed by observation, then change the map,build the factory near a lot of goodies to test its pickup function. Use the player to attack it to check if its hit points is right. Kill them to check if the drop goodie function works.

5.test AngrykleptoBots

the same as checking kleptoBots, besides have the player stand in the front to checking its attacking ability.

6.test Kleptofactory

Compare the factories that are built together and that are built apart to check if it is judging the KleptoBots nearby. Count the kleptoBot the in certain amount of time to test if it is producing the robots at the appropriate rate.

7.test others

these objects are tested during the test of the mobileActors