Hierarchy of Classes

Graph Object

Actor

Agent

Player

Robot

SnarlBot

KleptoBot

AngryKleptoBot

Exit

Wall

Boulder

Hole

Bullet

KleptoBotFactory

PickupableItem

Jewel

Goodie

ExtraLifeGoodie

RestoreHealthGoodie

AmmoGoodie

1. Public Member Functions

Actor:

Actor(StudentWorld\* world, int startX, int startY, int imageID, unsigned int hitPoints, Direction startDir);

Actor constructor requiring the start position, imageID, startDir for GraphObject and a pointer to the world it exists in and hit points

virtual ~Actor();

virtual destructor to properly destroy all classes derived from Actor

virtual void doSomething() = 0;

doSomething pure virtual as all actors do something different during a tick. This makes Actor abstract base class.

bool isAlive() const;

checks if the actor is still alive by returning bool member tracking if dead or not

void setDead();

sets the bool member to indicate player is dead

virtual void takeGoodie(int IID);

sets goodie of kleptobot to a goodie IID but this one does nothing

declared here because StudentWorld uses actor pointers

StudentWorld\* getWorld() const;

returns pointer to world object exists in

virtual bool allowsAgentColocation() const;

checks if actor allows another actor in the same space virtual because some do and others do not

virtual bool allowsBoulderColocation() const;

checks if actor allows boulder in the same space virtual because some do and others do not

virtual bool countsInFactoryCensus() const;

checks if actor counts in factory census virtual as it is true for kleptobots but nothing else and StudentWorld uses actor pointers to call this function

virtual bool stopsBullet() const;

checks if actor stops bullet virtual as not all do

virtual bool isDamageable() const;

checks if actor takes damage virtual because not all do

void damage();

deals two damage to actor

virtual int getGoodie();

returns goodie actor is holding only for kleptobots virtual because StudentWorld uses actor pointers

virtual bool isSwallowable() const;

checks if actor is swallowable true for boulder false for other classes so virtual as StudentWorld calls this function with actor pointer

virtual bool isStealable() const;

checks if actor is stealable true for goodies false for other classes so virtual as StudentWorld calls this function with actor pointer

int getHitPoints() const;

gets actors hit points

void setHitPoints(int amt);

set hit points of actor to int amt

virtual void tryToBeKilled();

if actor took damage checks if the actor is dead virtual as each actor does something when it dies

Agent:

Agent constructor

agents are moving actors abstract base class as doSomething is not defined for it

virtual ~Agent()

virtual as abstract base class for derived agents

Player:

Player constructor

constructs player object

virtual ~Player()

destroys player object

virtual void doSomething();

player does different things base on player input move, shoot, restart

unsigned int getHealthPct() const;

returns player health percentage for text at top

unsigned int getAmmo() const;

returns amount of ammo player has

void increaseAmmo();

increases amount of ammo player has

Robot:

Robot constructor

constructs robot enemies abstract class as doSomething is not defined

vitual ~Robot();

virtual as nots are derived from this class

bool ClearShot();

checks if robot has clear shot at player defined here as several robots shoot

int getCurrTick();

returns current tick

void incrTick();

increase tick by one

void resetTick();

set tick to one

tick functions defined here since all robot use them to meter their actions

SnarlBot:

SnarlBot constructor

constructs snarlbot with its direction

virtual ~SnarlBot()

destroys snarlbot

virtual void doSomething();

shoots at player if it can moves in its direction if it cannot switches direction if cannot move in current direction

virtual void tryToBeKilled();

if snarlbot dies increase player score 100

KleptoBot:

KleptoBot constructor

constructs KleptoBot

virtual ~KleptoBot();

destroys kleptobot virtual as angry kleptobot derived from this class

virtual void doSomething();

if on a goodie has a chance to steal it otherwise moves in its current direction unless blocked or reaches predetermined number of steps in direction then it calls ifBlockedOrDistance()

virtual bool countsInFactoryCensus() const;

does count in factory census

virtual void takeGoodie(int IID);

sets goodie to one stolen

void ifBlockedOrDistance();

picks new direction if kleptobot cannot move in current direction or steps reach distance it should move in the same direction

virtual void tryToBeKilled();

if killed drops goodie and increases score by correct value

virtual int getGoodie();

get goodie kleptobot has stolen

AngryKleptoBot:

AngryKleptoBot constructor

constructs angry kleptobot

virtual ~ AngryKleptoBot();

destroys angry kleptobot

virtual void doSomething();

if bot can shoot player fires a bullet otherwise does what a regular kleptobot does

Exit, Hole

virtual bool stopsBullet() const;

do not stop bullets so returns false instead of default true

Hole doSomething() destroys boulder and self if collocated

Exit doSeomthing() checks if jewel count if it is zero makes it self visible if player is collocated with it and it is visible end level and increase score properly

Exit constructor makes exit invisible

Hole allowsBoulderColocation(); returns true

Boulder, Wall

doSomething() does nothing

Wall isDamageable() returns false

Boulder isSwallowable returns true;

Factory

doSomething()

if it can it produces a new kleptobot

PickupableItem

Abstract base class doSomething is not defined

allowsAgentColocation() returns true

stopsBullet() returns false

pScore() returns score each goodie provides

Jewel

doSomething() increases score appropriately tells StudentWorld there is one less jewel

Goodie

Abstract base class doSomething is not defined

isStealable() returns true

Ammo

doSomething() increases ammo if player picks it up

Restore

doSomething() restores health if player picks it up

ExtraLife

doSomething() increases player lives if player picks it up

StudentWorld

int loadLevelAndCreate();

creates level from level file

virtual int init();

initialize level and all actors

virtual int move();

calls all actors do something

virtual void cleanUp();

clean up level if finished or restarted by destroying all actors

bool ifActorBlocks(Actor\* actor, int x, int y, int dx, int dy);

check if actor can move to a position

bool canBoulderMoveTo(int x, int y);

check if a boulder can move to a position

bool BulletIsBlocked(Actor\* a, int x, int y);

check if bullet is blocked at a position

bool isPlayerColocatedWith(int x, int y) const;

check if player is collocated with something at the square

void setDisplayText();

set display at top

int getCurrentLevelBonus();

return level bonus

bool swallowSwallowable(Actor\* a);

check if boulder is swallowed by hole

int getJewelCount();

return number of jewels left on level

void decrJewel();

decrease number of jewels on the level

Player\* getPlayer();

get the player in this world

void addActor(Actor\* a);

add actor to list of actors for those created by actors such as bullet

bool colocatedGoodie(Actor \*a);

check if kleptobot is on a goodie

bool colocatedFactory(Actor \*a);

check if kleptobot is on a factory

bool census(Actor\* a);

check if there are 3 kleptobots around a factory

2.

No known functionalities failed to implement or bugs

3.

StudentWorld handled all movement and cases where things are on the same square e.g. boulder swallowed and kleptobot stealing a goodie. StudentWorld also handled bullet movement in the sense it ran through the actors to ensure it does interact with anything along its path. The robot checked for clear shot with a for loop to check every square between it and the player calling StudentWorld blocked bullet function.

4.

Test Cases

I tested all the classes by creating unique level files and testing their functionalities.

Player was tested along with wall, boulder, and hole. Tested player movement by making sure it could not walk through walls, boulders, and holes. Boulders were pushed onto holes in order to make sure both died when collocated. Boulders were pushed around to make sure they moved correctly and stopped the player from moving if it was pushing it into something that did not allow boulder collocation.

Goodies were tested by creating a level with them and picking them up and making sure they did the correct thing, i.e. ammo increased ammo count by 20, restore life returned health to 100%, and extra life gave an extra life. Boulder were pushed towards goodies to make sure they did not pass through them.

Jewels and exit were tested by collecting all the jewels and making sure the exit appears and going through the exit gives the proper increase in score and loads the next level. Boulder was also pushed next to exit to make sure it did not collocate with the exit.

All the enemies were tested by creating a level with them and making sure they interacted with the environment correctly. Robots cannot walk through boulders, walls, holes, and other agents which was tested by creating a mini square of boulders and placing the bots in them. The player was also stood in front of shooting robots to make sure they fired at it and then hid the player behind boulder and wall to make sure bot would move instead of shooting as there was no longer a clear shot. Player can also not walk through bots, which was tested by running it into them. The score was also checked when each type of enemy was destroyed. Also ensured robots moved properly by watching behavior in the unique levels, e.g horizontal snarl bot only moved left and right and only switched if hitting obstruction, and cerr steps and distance for kleptobots.

KleptoBot factory was tested by creating a 7x7 arena of boulders to ensure only three bots would be created in the area described by the spec. The arena was filled with goodies and jewels to make sure the bots only stole goodies. They were then destroyed to ensure the goodies dropped properly.

StudentWorld was tested by playing the game.