Ronaq Virdi

904449591

Project 3 Report

**High Level Function Descriptions:**

**Actor Class:**

Actor(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

int getHealth() const – Many of the classes that are derived from Actor have a health so I made this function to easily access the actor’s health.

int getAmmo() const – Some of the class derived from actor have ammo, so this function returns the amount of ammo.

void setHealth(int health)- This function takes in a parameter to set the health of objects such as the player, robots, and boulders.

void setAmmo(int ammo) – This function takes in a parameter to set the ammo for the player.

virtual void doSomething() = 0 – I made this function pure virtual because I wanted the actor class to be abstract. Also, every class derived from actor needs to have a different variation of this method.

StudentWorld\* getWorld()- This function returns a pointer to the student world that the object is found in. Every class needs access to its student world, which is why I put this function in the actor class.

bool isDead() const- This function is made to check if a particular object is dead or not.

void setDead() – This function will set the current object to dead so it can be removed from the game in the next tick.

int returnID() – This function returns the image ID for each specific object. I later found out that this function was already provided in the gameWorld.h.

virtual void damage() – This function was made virtual because each object in the game has a special way of being damaged by bullets.

**Item Class:**

Item(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void doSomething() = 0- This class is derived from the Actor class and has a pure virtual doSomething method because all the classes that are derived from this class have their own special doSomething code.

**Goodie Class:**

Goodie(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething()- Since each goodie that is derived from this class has very similar code, I place the doSomething code for each goodie in this base class.

virtual void reward() = 0 – This function is pure virtual in order to make the goodie class abstract. Each goodie that is derived from this class has a different reward, such as more points or ammo, etc.

**Class Player:**

Player(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void doSomething()- For each direction I first check if the player is next to a boulder and if the player can move the boulder. If not I check if the direction specified is clear of any obstruction. If it is then I move the player in that direction. If the user hits escape the player is set to dead and loses a life.

virtual void damage()- Every time the player is attacked by a bullet it will lose two hitPoints and make a sound that it has been hit. If the player’s hitpoints reach zero then the player is set to dead.

**Robot Class:**

Robot(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething() = 0 Since both the snarlbots and kleptobots have their own doSomething methods, I made this method pure virtual so that the robot class would be abstract.

void hasShot(bool x) – This function sets the parameters to true is the robot has shot.

bool getShot() – This function checks to see if a bullet has fired a bullet.

**Class Snarlbot**

Snarlbot(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) – takes in 5 parameters and initializes variables

virtual void doSomething() – This function first checks to see if the snarlbot is dead. If not then the snarlbot will check to see if it can attack. If the bot attacks then the function will simply return. However if the bot does not attack then it will move.

virtual void shoot()=0 – The horizontal and vertical snarlbots have their own methods for shooting so I made this class abstract.

virtual void move() = 0 - The horizontal and vertical snarlbots have their own methods for moving so I made this class abstract.

virtual void damage()-Every time the snarlbot is attacked by a bullet it will lose two hitPoints and make a sound that it has been hit. If the snarlbot’s hitpoints reach zero then the snarlbot is set to dead.

**Class Horizontal Snarlbot**

HorizontalSnarlbot(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void shoot() – Checks if the robot is facing right or left. If the player is in the robot’s sight then the robot should fire and set the hasShot method to true.

virtual void move()-Checks if the robots next position is free from any obstruction. If it is empty then move the robot left or right. Otherwise change the robots direction to either right or left.

**Class Vertical Snarlbot**

VerticalSnarlbot(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void shoot()-Checks if the robot is facing up or down. If the player is in the robot’s sight then the robot should fire and set the hasShot method to true.

virtual void move()-Checks if the robots next position is free from any obstruction. If it is empty then move the robot up or down. Otherwise change the robots direction to either up or down.

**Class Kleptobot**

Kleptobot(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething()- The kleptobot will firt check to see if it is dead. If not, it will call the pickUpGoodie function. If the pickUpGooide function is not called then the kleptobot will check to see if it can shoot. If the shoot function is not called, the kleptobot will lastly check if it can move.

virtual void pickUpGoodie()-If the robot has not yet picked up a goodie there is a one in ten chance that the robot will pick up a goodie. The robot will call a separate function to remember the specific type of goodie that it picked up.

virtual void moveKleptobot() – If the robot has not taken distanceBeforeTurning number of steps then the robot will move in that direction and then return. If the robot has moved distanceBeforeTurning number of steps or has faced an obstruction, then it will set a new distanceBefore turning and new direction. The robot will randomly check which direction is open and then try to move in that new direction.

virtual void damage() – Everytime a kleptobot is shot it will lose 2 points. Otherwise if the health of the robot is less than zero, then the robot will be set to dead. If the robot had a goodie of any type, then it should create the goodie in the same position that the robot died on.

virtual void reward() = 0 – This function makes the kleptobot class abstract. This function gives a different reward for killing either type of kleptobot.

virtual void shootBullet() = 0 – Each type of kleptobot has its own shoot method.

int getDistanceBeforeTurning() – This function simply returns the int value of getDistanceBeforeTurning.

void setDistanceBeforeTurning(int d) – this function sets a new distanceBeforeTurning value.

void setDistanceCounter(int s)-sets the number of steps the robot has taken since setting the new distanceBeforeturning value.

void hasPickedUpGoodie(bool x) – changes the parameter to true if the robot picked up a goodie.

bool didRobotPickUpGoodie() –checks to see if a robot picked up a goodie

void rememberTypeofGoodie(int imageID) – This function remembers the type of goodie that a kleptobot has picked up.

int getTypeofGoodie() – This function returns the type of goodie that a kleptobot picked up

StudentWorld\* getStWorld() – returns a pointer to the robot’s studentWorld.

void hasRobotMoved(bool m)- If the robot has moved in the tick this function is set to true.

bool didRobotMove()- this function checks to see if a robot moved during the tick.

bool didRobotShoot()-this function checks to see if a robot shot a bullet during the tick.

void didShoot(bool p)-If the robot shot a bullet during the tick this function is set to true

**Class RegularKleptobot**

RegularKleptobot(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void shootBullet()- this function simply returns because the regular kleptobot cannot shoot bullets

virtual void reward() – This function increases the player’s score by 10 points.

**Class AngryKleptobot**

AngryKleptobot(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void shootBullet()- This function checks to see what direction the robot is facing. If the player is in the same row or column as the robot, then the robot will fire a bullet at the player.

virtual void reward() - This function increases the player’s score by 20 points.

**Class Wall**

Wall(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething() – The wall’s doSomething method does nothing.

**Class Hole**

Hole(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething() – It first checks if the hole is dead. If not the hole checks if a boulder is on its same location. If a hole and boulder are in the same spot, then both the hole and boulder are set to dead.

**Class Boulder**

Boulder(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void doSomething() – The boulder’s doSomething does not do anything.

**Class Jewel**

Jewel(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void reward()- increase the player’s score by 50 points

**Class Extra Life Goodie**

ExtraLifeGoodie(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void reward()- increases the player’s score by 1000 and increases the number of lives left

**Class Restore Health Goodie**

RestoreHealthGoodie(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void reward()- increase score 500 and reset the player’s health back to 20

**Class Ammo Goodie**

AmmoGoodie(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void reward() – increase score by 100 and re-set the amount of ammo to 20

**Class Exit**

Exit(int imageID, int startX, int startY, Direction dir,StudentWorld\* sw) - takes in 5 parameters and initializes variables

virtual void doSomething() – If the player is on the same spot as the exit and if the exit is visible, then play a sound, increase the score by 2000, and give the rest of the bonus because the level is over.

**Class Bullet**

Bullet(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw)- takes in 5 parameters and initializes variables

virtual void doSomething()- If the bullet is not dead, then call the attack function. If the bullet does not attack then call the move function and then call the attack function again.

void attack() – check if the bullet is on the same square as an attackable object. If the bullet is, then set the bullet to dead and set the object is attacked to dead.

void moveBullet() - check the direction of the bullet, if there is no obstruction in the way then move the bullet in that direction.

void didAttack(bool a)-returns true if the bullet called its attack function

bool bulletHasAttacked() – checks to see if the bullet attack function has been called.

**Class Factory**

Factory(int imageID, int startX, int startY, Direction dir, StudentWorld\* sw, string type)- takes in 6 parameters and initializes variables

virtual void doSomething() – if the count function returns less than 3, then the factory has a 1 in 50 chance to create a robot. If the factory is a regular factory then it will create a regular robot. Otherwise if it is an angry kleptobot factory, then it will make angry kleptobots.

int count(int x, int x2, int y, int y2) – This function uses a nested for-loop to iterate through every position 3 spots left, right, up , and down. If it finds any type of kleptobot in any of those positions then it will increase the counter. At the end this function returns the number of kleptobots in that region.

**Class Student World**

StudentWorld(std::string assetDir)- Initializes the game world

~StudentWorld()-iterates through the list and deletes every object and deletes it from the list

virtual int init()- initializes the gameplay. This function loads the level and loads all the actors in the correct spot.

virtual int move()-This function is called in every tick. This function makes every actor do something. If the player dies then the level is reset. If the player completes the level then this function calls the init to reset the next level. Also this function calls a remove dead objects function.

virtual void cleanUp()-iterates through the list and deletes every object and deletes it from the list

Actor\* findObjectAt(int x, int y, int object)-this function iterates through the entire list and returns a pointer to the actor whose x, y and image Id matches the parameters passed into the function.

Actor\* findAnythingAt(int x, int y)-this function simply iterates through the list and looks for any actor at the give x and y position.

void removeDeadObjects()- This function iterates through the list. If any of the actor’s isDead function is true, then this method deletes that actor and erases it from the list

bool playerCollectedAllJewels()- This function iterates through the list and if it finds a jewel that it returns false. If no jewels are found then this function returns true.

void exposeTheExit()- Iterates through the list and finds the exit actor. If the exit is not visible then this function sets the exit to visible and plays a sound.

void reduceLevelBonusByOne() – decrements the level bonus by one in every tick

unsigned int getBonusPoints()- This function gets the number of bonus points

void pushToList(Actor\* p) – This function pushes the actor to the front of the list.

int robotTick() – returns the ticks that the robots should move at depending on the level

void increaseTicks() - increments the number of ticks

int getTick()- returns the current tick

Actor\* getAccessToSpecific(int imageID)- returns a pointer to the actor with the passed in image id

bool checkBtwnPlayerHorizontal(int a, int b , int c) – This function iterates through all the positions between the player and robot in the horizontal direction and returns true if there are no obstructions in the way.

bool checkBtwnPlayerVertical(int a, int b , int c)- This function iterates through all the positions between the player and robot in the vertical direction and returns true if there are no obstructions in the way.

bool NoObstruction(int dx,int dy)- this function checks if there are any obstructions on the given position

bool NoBlockForPlayer(int x, int y) – Checks if there are any objects at the given position that block the player

void setDisplayText()-This function updates the scoreboard after every tick

string FormatStringNicely(int score,int level,int lives,int health, int ammo, int bonus)- this function uses string streams to set the scoreboard in the proper format

bool PlayerCompletedLevel() – This function checks to see if the player has completed the level.

void setLevelComplete(bool d)-This function returns true when the player completes the level.

string FormatLevel()- this function properly formats the level in the format of the game.

**Functionality:**

To my knowledge all of my actors are working correctly. To the best of my understanding and after all testing, I think that my game works like the game in the spec.

**Assumptions:**

I assumed that all actors would be loaded within the bounds of the 15 x 15 maze grid. I also assumed that if a robot shot a bullet at the player and the player moved out of the way, then that bullet could hit another robot or a boulder. Another assumption I made was that after the player completes a certain level, his ammo and health go back to 100%. Other than that, I followed the spec throughout the project.

**Testing:**

**Player:**

When I was checking the player class I first checked if the player would move in the desired direction if there was an obstruction in the way. I made sure that the player did not walk through any objects. Also another test I checked was whether the player could properly move a boulder to the desired direction. I made sure the player could not move a boulder into any obstruction. Also I checked to make sure the player could not move more than one boulder at a time.

In regards to the player shooting bullets, I made sure that the player’s bullet did not attack more than one object. Also I checked to make sure that the bullet that the player fired did not go through more than one object. When I was testing the player’s damage function I made sure that the player died after 10 shots. I made sure the player made a sound after it died.

**Snarlbots:**

I made sure that my snarlbot did not move and shoot in the same tick. I made sure that if the snarlbot was shooting at the player then it would stay at the same spot and simply fire more bullets. I made sure the snarlbot moved in its respective row and column and did not run through any objects. I tested whether the snarlbot switched direction and moved to the next position after hitting an obstruction.

In regards to the snarlbot’s damage I made sure that the snarlbot died after five shots. After the snarlbot died I made sure that the game played a specific sound.

**Kleptobots:**

I made sure that my kleptobot did not move and shoot in the same tick. I made sure that if the kleptobot was shooting at the player then it would stay at the same spot and simply fire more bullets. I made sure the kleptobot moved in its respective row and column and did not run through any objects. I tested whether the kleptobot switched direction and moved to the next position after hitting an obstruction. In terms of the angry kleptobot, I made sure that the bot shot a bullet at the player only when the player was in the same row or column. I checked to make sure that the snarlbots did not move randomly or too many times in one tick.

In regards to the kleptobot’s damage I made sure that the kleptobot died after three or four shots. After the kleptobot died I made sure that the game played a specific sound. If the kleptobot died and had a goodie, I ensured that the goodie was made in the same spot that the kleptobot was originally at.

**Wall:**

I make sure that the wall does not move or does not shoot any bullets. I also make sure that the wall does not spawn any kleptobots and does not disappear in a particular tick. I also made sure that if a bullet hit a wall then it would die and not move on. Also I made sure that no object could move through a wall.

**Hole:**

I make sure that all the holes are visible and that they do not move. I check to make sure that the holes don’t shoot or create kleptobots. I ensure that no actor can walk over a hole. The only that can go on a hole is a boulder. I make sure that when the hole and boulder are in the same spot they disappear and are set to dead. The hole should never move. Bullets should be able to go through the holes.

**Boulder:**

When I try to move a boulder I make sure the boulder does not go through or move onto any obstruction. The boulder should be able to go on a hole but then I checked to make sure that the boulder disappeared along with the hole. I made sure the bullet could be shot and would be destroyed after 5 shots. The player and the robots should be able to shoot the bullet. I made sure the bullet never moved more than one spot at a time. If a boulder was next to a boulder and I tried to move into that direction then I ensured that the bullet would not move.

**Jewel:**

I made sure that the jewels were set to visible and that they would not move or shoot any bullets. I ensured that once I collected all the jewels that the exit would show up at the proper spot. I also checked to see if the player received 50 points for collecting a jewel. I made sure that all the objects that could walk on a jewel were allowed to move onto the same square as a bullet.

**Goodies(Ammo, Extra Life, Restore Health)**

For these actors I made sure that they never moved or shot bullets. I made sure that the goodies were set to visible and were loaded in the correct initial positions. I made sure that each bullet gave its respective reward to the player (extra lives, more points, more ammo, etc.). Also once the goodies were collected by the player I made sure that a sound was played. Also if a kleptobot picked up a goodie, I made sure that the goodie was set to dead and no longer part of the game. However, once the kleptobot died I ensured that the goodie was created in the location where the kleptobot died.

**Exit:**

I made sure that the exit was set to invisible at the beginning of the level. Only after all the jewels were collected did the exit show up. When the exit was displayed I made sure that the reveal sound only played once. Also when the player went onto the exit I ensured that the level complete sound played. Other than that I made sure the exit did not move, shoot, or spawn robots. I made sure that all actors could be moved onto the exit.

**Bullets:**

I make sure that the bullets only move in one line and do not move in any random direction. I check to make sure that the bullet is made in the spot next to the player or robot. I make sure that the bullet can attack robots, boulders, and the player. The bullet should not be able to attack walls or factories. After the bullet has attacked an actor it should not be able to do anything else, at that point the bullet should be set to dead. I ensure that there are no bullets on the maze unless the player or one of the robots fires a bullet. I make sure that the bullet can go over holes and over goodies and the jewels.

**Factories:**

I make sure that the factory never moves or shoots. I make sure that a regular factory only makes regular robots and that an angry factory can make angry kleptobots. I make sure that the factory only makes a new robot if there are no more than three bots in a 3 x 3 radius from the factory. I make sure that the bullets cannot attack the factory. The factory should never be set to dead by the bullets. The factory should not do any other behavior that is not specified in the spec.

**Student World and Miscellaneous:**

In the student world I make sure that the scoreboard is always correctly adjusted. The bonus always decrements in each tick. The score will increase when goodies and jewels are collected. The health of the player should decrement when the player is shot and the ammo count should fall when the player fires a bullet. After the player finishes a level the student world should call the init function and load the data from the new level. I make sure that no random or obscure behavior takes place throughout the game.