CE218 RESIT ASSIGNMENT - SPACESHIP GARBAGE RECYCLER

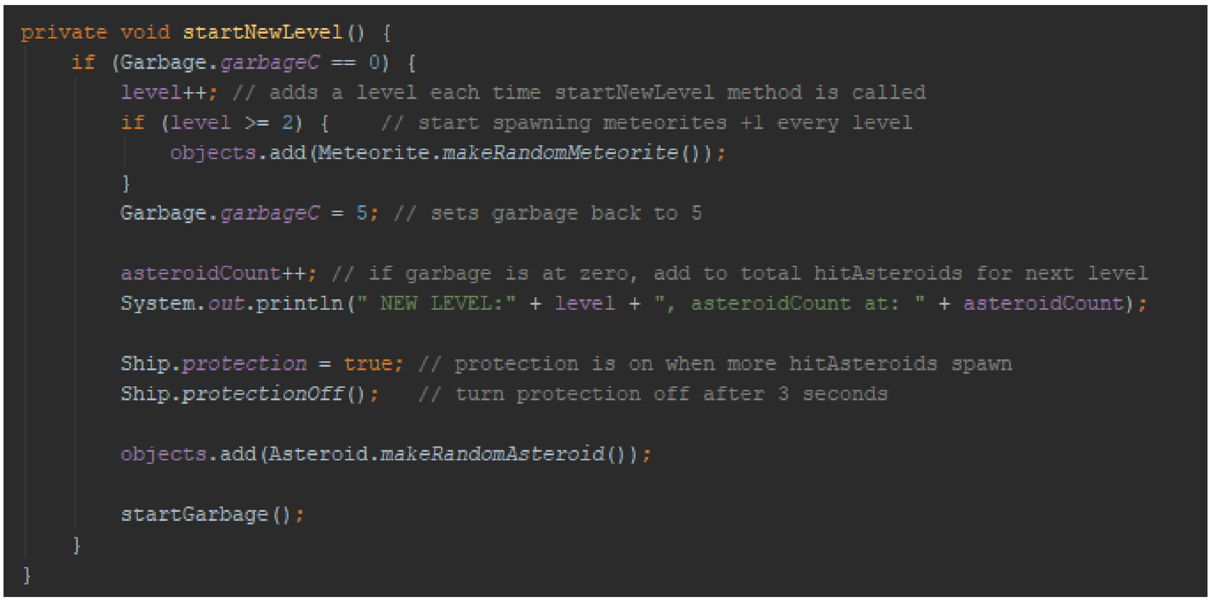
1603930

UNIVERSITY OF ESSEX
COMPUTER GAME PROGRAMMING

Main Design Choices

The number of asteroids and meteorites increasing by level

This allows the levels to get harder throughout the game as the player is required to move around more obstacles. The asteroids themselves do not actually get destroyed and act as obstacles. Also, the asteroids do not reset in position after a new level is reached which makes gameplay more fluid.



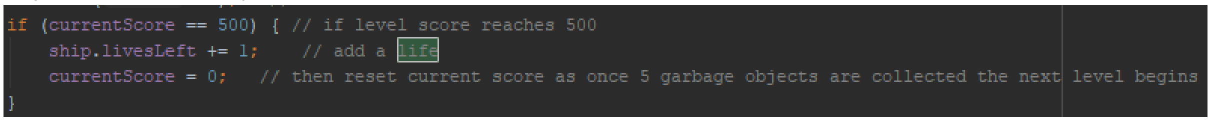
```java
private void startNewLevel() {
    if (Garbage.garbageC == 0) {
        level++; // adds a level each time startNewLevel method is called
        if (level >= 2) {    // start spawning meteorites +1 every level
            objects.add(Meteorite.makeRandomMeteorite());
        }
        Garbage.garbageC = 5; // sets garbage back to 5

        asteroidCount++; // if garbage is at zero, add to total hitAsteroids for next level
        System.out.println(" NEW LEVEL:" + level + ", asteroidCount at: " + asteroidCount);

        Ship.protection = true; // protection is on when more hitAsteroids spawn
        Ship.protectionOff();   // turn protection off after 3 seconds

        objects.add(Asteroid.makeRandomAsteroid());

        startGarbage();
    }
}
```

In Game.java.

Meteorites are implemented however after level 2 Is achieved. The number of these objects also increase throughout gameplay. Meteorites are smaller but much faster on average than the asteroids, this makes the game play more interesting for the player. This game mechanic makes the game itself very scalable, the player as a result must keep playing until the ship is dead. Also, as a programmer this shows that the use of abstract classes allows for further swift implementation of game objects for instance more pickups or even enemies.

Player lives

The player starts with 3 lives, as the player picks up (collides with) 5 garbage objects the next level begins automatically but also gains a life. Here 500 score (in one level) equates to 5 objects and each object rewards 100 points.



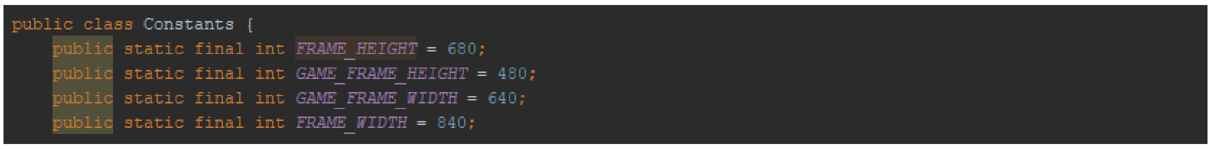
```java
if (currentScore == 500) { // if level score reaches 500
    ship.livesLeft += 1;    // add a life
    currentScore = 0;   // then reset current score as once 5 garbage objects are collected the next level begins
}
```

In Game.java

This mechanic rewards the player for completing the main objective which is collecting garbage, and if done well the player would be able to save up lives and as a result survive into further levels of the game. This effort would be displayed on the leader-board.

Frame extension

A quick frame extension allowed for space which was utilised with drawn strings that displayed variables such as total score.



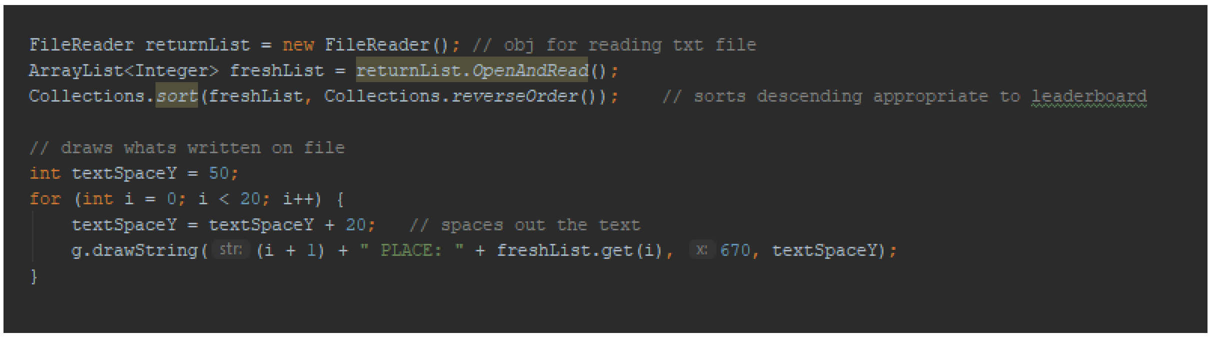
```
public class Constants {
    public static final int FRAME_HEIGHT = 680;
    public static final int GAME_FRAME_HEIGHT = 480;
    public static final int GAME_FRAME_WIDTH = 640;
    public static final int FRAME_WIDTH = 840;
```

In Constants.java

This is not crucial for game design but does allow variables from being drawn outside of the actual playing field, this does improve gameplay experience as drawn out variables can be seen by the player but do not obstruct actual gameplay.

Leader-board

This implementation with use of a file reader and writer accesses a leaderboard.txt file, this is then translated at the end game (gameOverState) and constantly displayed in the View.java class.



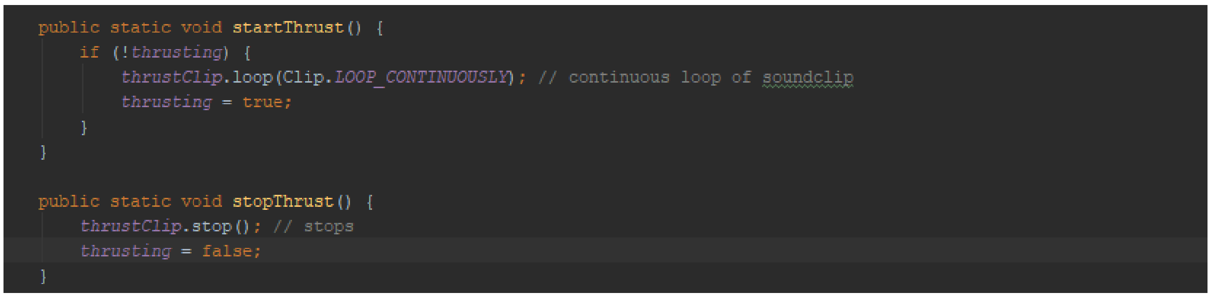
```
FileReader returnList = new FileReader(); // obj for reading txt file
ArrayList<Integer> freshList = returnList.OpenAndRead();
Collections.sort(freshList, Collections.reverseOrder());    // sorts descending appropriate to leaderboard

// draws whats written on file
int textSpaceY = 50;
for (int i = 0; i < 20; i++) {
    textSpaceY = textSpaceY + 20;   // spaces out the text
    g.drawString( str: (i + 1) + " PLACE: " + freshList.get(i),  x: 670, textSpaceY);
}
```

In View.java.

A leader-board as such is important in game design as it allows the player to strive to beat others, in a sense adding competitiveness which makes the game ultimately more fun.

Sound effects

This feature is implemented through the class SoundManager in utilities, this allows the objects to play sounds which define characteristics such as the ship thrusting.



```
public static void startThrust() {
    if (!thrusting) {
        thrustClip.loop(Clip.LOOP_CONTINUOUSLY); // continuous loop of soundclip
        thrusting = true;
    }
}

public static void stopThrust() {
    thrustClip.stop(); // stops
    thrusting = false;
}
```

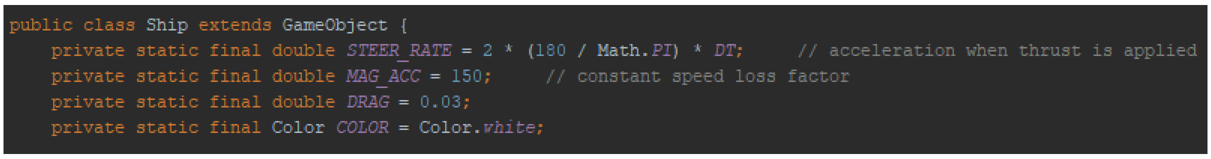
In SoundManager.java.

This makes the game experience much more enjoyable as not only is the game visually stimulating but also auditorily.

Parameters for Good Playability

Ship speeds

Adjusting these parameters are crucial for gameplay as the ship is controlled by the player.



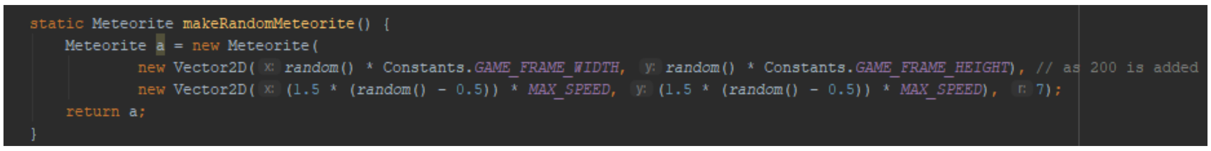
```
public class Ship extends GameObject {
    private static final double STEER_RATE = 2 * (180 / Math.PI) * DT;     // acceleration when thrust is applied
    private static final double MAG_ACC = 150;     // constant speed loss factor
    private static final double DRAG = 0.03;
    private static final Color COLOR = Color.white;
```

In Ship.java

Adjusting the steer rate essentially determines how quick the ship should turn. The speed lost factor determines the acceleration of the ship. The drag can be adjusted so that the ship slows down more or less when the ship is not thrusting. These were altered throughout developing and other parameters were decided from the basis of these – etc. are the asteroids too fast for the ship? Or too slow?

Obstacle parameters

These parameters determined what speeds and what direction the obstacles would travel in. Programmed to have random directions and speed within a range. (With use of java.lang.Math.random). The same was implemented to Garbage.java object but with lesser speeds as to make it easier for the player to pick up.



```
static Meteorite makeRandomMeteorite() {
    Meteorite a = new Meteorite(
            new Vector2D( x: random() * Constants.GAME_FRAME_WIDTH,  y: random() * Constants.GAME_FRAME_HEIGHT), // as 200 is added
            new Vector2D( x: (1.5 * (random() - 0.5)) * MAX_SPEED,  y: (1.5 * (random() - 0.5)) * MAX_SPEED),  r: 7);
    return a;
}
```

In Meteorite.java

Essentially playing around with these values allowed for "RNG" each game which makes each start unique, this adds to the "replayability" of the game as it makes it more fun.

Appraisal

How did the project go?

I personally enjoyed this project a lot. It honestly made me realise how scalable object-orientated programming can be – once I initiated the game with a foundation of the abstract class GameObject my working efficiency increased. It made me realise that making this sort of foundation in any program is important as it makes future programming much easier.

What was easy?

Implementing further objects after creating an abstract class.

What was difficult?

Making the high score chart only write once as it would flood the txt file with hundreds of entries of current scores, this was however countered with the use of Boolean values inserted in the correct space.

Another feature that was difficult to program was the restart game feature. I had to let the program know that spacebar resets the game but can only be reset once and that would be when the game is over, but the game can be over multiple times if it can be reset. Again, a painful process of figuring out where Boolean values belong. This led me to confusion many times.

Any unresolved problems in submission?

None that I can tell. I believe I have implemented features that I desired to successfully. However, due to personal time restraints I was not able to implement the features that I would like to. My next feature would be pick ups that allow the ship to be protected for 10 seconds and pick ups that speed the ship up. I then would of liked to add randomly generated fixed obstacles.

Any parts of the game that you are proud of?

I am proud of the leader board and filei/o, I think this adds a layer of complexity to the programming of the game.