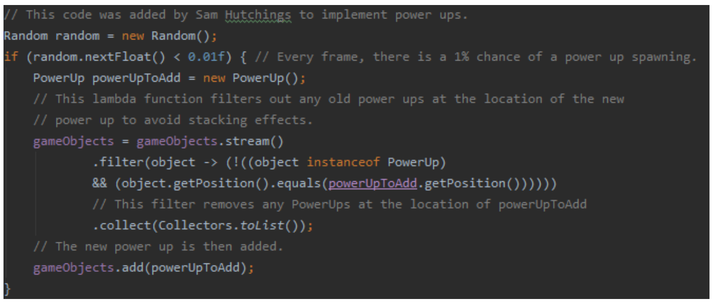
For assessment 4, we were required to make three mandatory changes:

1. Implement five types of power ups that give fire engines different abilities when picked up
2. Add support for different difficulty levels
3. Add the ability to save and load the game.

For powerups, there were three key tasks that had to be completed: Creating the powerup entity which firetrucks would run over to pick them up, changing the firetrucks code to implement different status effects depending on which type was picked up, and GameScreen spawning powerups when required. The powerups have a set number of predefined locations that they can spawn in to ensure they can't spawn out of bounds or inside a building, and GameScreen.Java has a new method PowerUp() which has a 1% chance of running (which spawns a powerup at that location when it does).

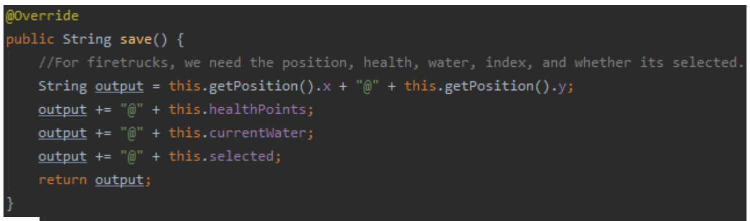


```java
// This code was added by Sam Hutchings to implement power ups.
Random random = new Random();
if (random.nextFloat() < 0.01f) { // Every frame, there is a 1% chance of a power up spawning.
    PowerUp powerUpToAdd = new PowerUp();
    // This lambda function filters out any old power ups at the location of the new
    // power up to avoid stacking effects.
    gameObjects = gameObjects.stream()
            .filter(object -> (!((object instanceof PowerUp)
            && (object.getPosition().equals(powerUpToAdd.getPosition())))))
            // This filter removes any PowerUps at the location of powerUpToAdd
            .collect(Collectors.toList());
    // The new power up is then added.
    gameObjects.add(powerUpToAdd);
}
```

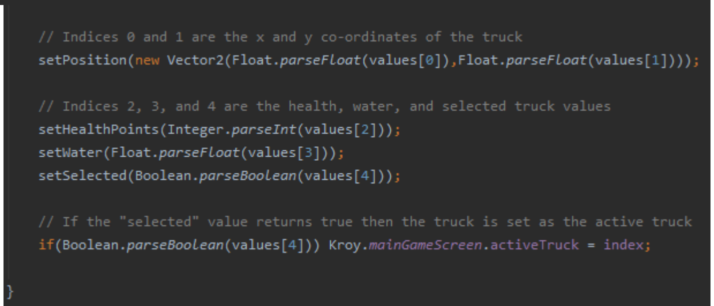
The powerup abilities are quite simple effects such as increased speed or increased firerate. We decided this was a good idea because powerups are a benefit that are worth going for, but not too powerful that they make the game too easy.

For difficulty levels, we added three difficulty options when a new game is started. Depending on which option was selected, fortresses will have a certain amount of health and do a certain amount of damage (more if a harder difficulty is selected). This is the only change made to implement difficulty levels.

Finally, the ability to save and load the game. We've made it so you have to be in the game first to save/load, rather than being able to also load before a game has started. When a game is saved, we save three separate dictionaries. The first is for entities that haven't died yet, the second is for those that have died, and the third is for non-entity data (time left until the fortress is destroyed and the player's score). A big change we had to make for this is keeping track of all destroyed entities. Now, instead of just removing entities from the list when their health becomes zero, we also insert them into the "deadObjects" array. Another crucial change was what we had to do to ensure entities could be saved, but only those that had to be saved. While we save fortresses and firetrucks, we don't save bullets nor patrols. The Entity class now has a new attribute UUID and a new method Save(). The UUID attribute is a unique ID which is used as the key for the entity in the dictionary, while the save() method converts any important attributes to a string format. While fortresses only need their health saved (since their attacks are infinite and their position is constant), firetrucks need their health, position, and water levels saved.



```java
@Override
public String save() {
    //For firetrucks, we need the position, health, water, index, and whether its selected.
    String output = this.getPosition().x + "@" + this.getPosition().y;
    output += "@" + this.healthPoints;
    output += "@" + this.currentWater;
    output += "@" + this.selected;
    return output;
}
```



```java
    // Indices 0 and 1 are the x and y co-ordinates of the truck
    setPosition(new Vector2(Float.parseFloat(values[0]),Float.parseFloat(values[1])));

    // Indices 2, 3, and 4 are the health, water, and selected truck values
    setHealthPoints(Integer.parseInt(values[2]));
    setWater(Float.parseFloat(values[3]));
    setSelected(Boolean.parseBoolean(values[4]));

    // If the "selected" value returns true then the truck is set as the active truck
    if(Boolean.parseBoolean(values[4])) Kroy.mainGameScreen.activeTruck = index;

}
```

Once data has been saved, it can be loaded at any time during a game. To load data: for each entity in the game, we fetch the data in the saved dictionary with the same UUID, and then load the data in by passing the values to the Load() method. The method splits a string value into its separate values, and adjusts an object's attributes accordingly. Since every entity is going to have a specific UUID that doesn't change on new games/restarts, accessing its data in a save file can be done in O(1) time.

SInce the code we inherited was up to a gold standard, we did not have to make any non-essential changes i.e changes in code not pertaining to the adjustments required for assessment 4.