2b. At the beginning of this project, I initially wanted to make a game similar to the popular game Geometry Dash. However early in to development, I realized that this would require making a custom stage by hand. This lead to me to create an algorithm that generates obstacles automatically, at a gradually increasing rate. However, shortly after, this caused a memory issue for my game. This lead to my deallocation algorithm, where I would delete all objects after leaving the screen. After that, the main engine of the game was all done, and I spent the rest of the time, debugging and adding graphics.

2c. For my game, *actionPerformed()* is the method that is called by the program every frame. This means that, because of the 50 frame per second speed of the game, this method is called 50 times per second.
One of the methods called by *actionPerformed()*, *gravity()*, is a method included in the *Player* class, which controls the y-coordinate of the player. The *sprite* variable is an integer used in another algorithm (that controls the sprite of the player). The next line is the change in velocity at every frame. At 50 frames per second, which the game runs at, the change in velocity, or the acceleration due to gravity, is 0.378 pixels per frame squared. And the change in position changes at *velocity* + 0.0072 pixels per frame. The last if statement is the check to make sure that the y-coordinate does not go past the floor coordinate 300.
Another method called by *actionPerformed()*, is *status()*, the method that checks all game parameters to determine the game's state. This consistently checks the collision of the player and obstacles and also deallocates the obstacles outside of the screen.

2d. My game used many abstractions during development for each element of the game. There are 9 classes total, including the death screen, the obstacles, the player, and even the floor. This was necessary because each of these elements have distinct attributes. This was also necessary for creating multiple of the same type of object, such as the obstacles. The *Obstacle* object, for example, is created every time a previous one has been deallocated, and thus needed to be abstracted for such a complex memory issue. This is especially true with the image processing for each of the sprites. I used the *javax.imageio* library to process images into my game, rather than using byte data interpreting.