Appendix - Dictator

```
package src;

import java.time.Clock;
import java.util.*;
import java.awt.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.net.URI;
import java.net.URL;
import java.util.List;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.swing.*;

/**
 * Dictator of all game aspects, and manager of inputs and the what not.
 *
 * @author Sky Johnson
 *
 */
public class Dictator extends JFrame {
	private static final long serialVersionUID = -3535839203565039672L;
	private static final int FRAMES = 60;

	private static final int GUN_CAP = 1;

	public final int BULLET_MAX = 8;

	public double bulletSpeed;

	public final int BULLET_REGEN = 10;
```

```
private int NUMBER_STARS = 100;

public boolean paused;
private boolean exit;

private boolean generated;
private boolean restart;
public boolean gameOver;
private boolean isGame;
public boolean seedPlay;
public boolean musicPlay;
public boolean selectDecision;
public boolean seedtypeing;
public boolean initseedstring;
public boolean firstenteredseedreleased;
public boolean secondenteredseed;
public boolean waitingforChoose;
public boolean musicstart;
public boolean musicplaying;

public boolean keyListenerOn;

public int lives = 3;
public int score;

public int bulletCount;
public int highScore;
private List<Actor> actor;
private List<Actor> toAddActor;

public String seed;
public File song;

protected ArrayList<Star> starlist;

public ArrayList<Actor> asteroids;
private Watch StarTimer;
public final int SIZE_X = 600;
public final int SIZE_Y = 600;

public Mouse mouse;

public boolean mouseDown;

public Point mousePoint;
```

```
public boolean noselect;

public boolean entered;

public boolean Mpress;

public boolean musicgame;

public boolean songselected;

public boolean jFileChoseOpen;

public boolean Spress;

public boolean seedgame;

public boolean endGame;

protected Player StarCaptain;
private SpaceMap Constellation;

public Random rand = null;

public int framesSoFar;

public int lineSoFar = 0;

public int TotalLines = 0;

public SpawnController spawner;

public MediaPlayer songplayer;

/**
 * Constructor for objects of class Dictator
 */
public Dictator() {
        // initialize instance variables
        super();
        new javafx.embed.swing.JFXPanel();

        bulletCount = 0;
        framesSoFar = 0;
        score = 0;
        bulletSpeed = 1;
```

```
mouse = new Mouse();
setLayout(new BorderLayout());
setDefaultCloseOperation(EXIT_ON_CLOSE);
setResizable(false);

// create SpaceMap and set Window and jazz
add(this.Constellation = new SpaceMap(this), BorderLayout.CENTER);
setContentPane(Constellation);

// Create Spawn Object on init, to be changed at Start Game methods
spawner = new SpawnController(this);

addKeyListener(new KeyAdapter() {
	public void keyPressed(KeyEvent e) {
		if (seed.length() > 1
				&& e.getKeyCode() == KeyEvent.VK_BACK_SPACE
				&& seedtypeing) {
			String temp = seed;
			seed = temp.substring(0, temp.length() - 1);
		}
		if (e.getKeyCode() == KeyEvent.VK_BACK_SPACE
				|| e.getKeyCode() == KeyEvent.VK_SHIFT) {

		} else if (seedtypeing) {
			seed += e.getKeyChar();

		}
		// Menu Keys
		if (e.getKeyCode() == KeyEvent.VK_P) {
			if (!checkForRestart() && !seedtypeing) {
				pause();
			}
		}
		if (e.getKeyCode() == KeyEvent.VK_M) {
			if (!checkForRestart() && !seedtypeing) {
				Mpress = true;
			}
		}
		if (e.getKeyCode() == KeyEvent.VK_S) {
			if (!checkForRestart() && !seedtypeing) {
				Spress = true;
			}
		}
		if (e.getKeyCode() == KeyEvent.VK_R) {
			if (!checkForRestart() && !seedtypeing) {
				restart();
```

```
                        }
                    }
                    if (e.getKeyCode() == KeyEvent.VK_ESCAPE) {
                        if (!checkForRestart()) {
                            exit = true;
                        }
                    }
                    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                        if (!checkForRestart()) {
                            entered = true;
                        }
                    }

                }

                public void keyReleased(KeyEvent e) {
                    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                        if (!checkForRestart()) {
                            entered = false;
                            if (seedtypeing && !musicgame) {
                                firstenteredseedreleased = true;
                            }
                        }
                    }
                    if (e.getKeyCode() == KeyEvent.VK_M) {
                        if (!checkForRestart()) {
                            Mpress = false;
                        }
                    }
                    if (e.getKeyCode() == KeyEvent.VK_S) {
                        if (!checkForRestart()) {
                            Spress = false;
                        }
                    }
                }

        });

        // Resize
        pack();
        setLocationRelativeTo(null);
        setVisible(true);

}

/*
```

```
 * Check to see if restart has been pressed or not
 *
 * @return If the key to restart has been pressed or not
 */
public boolean checkForRestart() {
	return restart;
}

// Restarts the game(Literrally runs threw startup again)
private void restart() {
	bulletCount = 0;
	framesSoFar = 0;
	bulletSpeed = 10;
	score = 0;
	lives = 3;

	if (musicgame == true) {
		songplayer.pause();
	}

	selectDecision = false;

	entered = false;
	seedtypeing = false;
	secondenteredseed = false;
	songselected = false;
	firstenteredseedreleased = false;
	waitingforChoose = false;
	setGenerated(false);
	keyListenerOn = false;
	initseedstring = false;
	keyListenerOn = false;
	jFileChoseOpen = false;
	musicstart = false;
	musicplaying = false;

	seedgame = false;
	musicgame = false;
	endGame = false;
	// for Seed Play
	seed = "Insert String";

	// For Music Play
	song = null;

	mousePoint = new Point();
```

```
        mouseDown = false;

        // Set star timer to refresh at every frame value
        this.StarTimer = new Watch(FRAMES);

        spawner.reset();

}

public void addRandoms() {
        this.setActor(new ArrayList<Actor>());
        this.toAddActor = new ArrayList<Actor>();
        this.StarCaptain = new Player(this);
        this.starlist = new ArrayList<Star>();
        this.asteroids = new ArrayList<Actor>();

        // this.Constellation = new SpaceMap(this);
        isGame = true;
        gameOver = false;
        restart = false;
        noselect = true;

        // Starlist Generation
        for (int i = 0; i < NUMBER_STARS; i++) {
                double a = (rand.nextDouble() * SIZE_X);
                double b = (rand.nextDouble() * SIZE_Y);
                double c = rand.nextDouble();
                int group = rand.nextInt(16);

                Star startemp = new Star(this, a, b, c, group);
                starlist.add(startemp);
        }

        toAddActor.add(StarCaptain);

        restart = false;
        setGenerated(false);

        if (musicgame) {

                new javafx.embed.swing.JFXPanel();
                String uriString = song.toURI().toString();
                songplayer = new MediaPlayer(new Media(uriString));
                songplayer.pause();
                songplayer.play();
        }
```

```
addMouseMotionListener(new MouseAdapter() {
	public void mouseMoved(MouseEvent e) {
		mouse.update(e.getPoint());
	}

	public void mouseDragged(MouseEvent e) {
		mouse.update(e.getPoint());
		mousePoint = e.getPoint();
		mouseDown = true;
	}

});

addMouseListener(new MouseAdapter() {
	public void mousePressed(MouseEvent e) {
		mouse.update(e.getPoint());
		mousePoint = e.getPoint();
		mouseDown = true;
	}

	public void mouseReleased(MouseEvent e) {
		mouseDown = false;
		mouse.update(e.getPoint());
	}
});

// Add Key Listener, only modifies player

addKeyListener(new KeyAdapter() {

	// key mapping for press
	public void keyPressed(KeyEvent e) {

		// Controlling Keys
		if (e.getKeyCode() == KeyEvent.VK_A) {
			if (!checkForRestart()) {
				StarCaptain.thrustingLeft(true);
			} else {
				StarCaptain.thrustingLeft(false);
			}
		}
		if (e.getKeyCode() == KeyEvent.VK_D) {
			if (!checkForRestart()) {
				StarCaptain.thrustingRight(true);
			} else {
```

```
                                        StarCaptain.thrustingRight(false);
                                }
                        }
                        if (e.getKeyCode() == KeyEvent.VK_S) {
                                if (!checkForRestart()) {
                                        StarCaptain.thrustingDown(true);
                                } else {
                                        StarCaptain.thrustingDown(false);
                                }
                        }
                        if (e.getKeyCode() == KeyEvent.VK_W) {
                                if (!checkForRestart()) {
                                        StarCaptain.thrustingUp(true);

                                } else {
                                        StarCaptain.thrustingUp(false);
                                }
                        }
                }

                // key mapping for release
                public void keyReleased(KeyEvent e) {

                        if (e.getKeyChar() == 'a') {
                                StarCaptain.thrustingLeft(false);
                        }
                        if (e.getKeyChar() == 'w') {
                                StarCaptain.thrustingUp(false);
                        }
                        if (e.getKeyChar() == 's') {
                                StarCaptain.thrustingDown(false);
                        }
                        if (e.getKeyChar() == 'd') {
                                StarCaptain.thrustingRight(false);
                        }
                }

        });
}

private void pause() {
        if (paused == false) {
                this.paused = true;
                isGame = false;
                if (musicgame) {
                        songplayer.pause();
```

```
                }
            } else if (paused == true) {
                this.paused = false;
                isGame = true;
                if (musicgame) {
                    songplayer.play();
                }
            }

        }

        /*
         * Starts the game, including game loop and update system.
         */
        private void startGame() {
            restart();
            // Game Loop
            while (true) {
                long start = System.nanoTime();

                // Tick
                StarTimer.update();
                for (int i = 0; i < 5 && StarTimer.hasPassedTicks(); i++) {
                    updateGame();
                }

                // render
                Constellation.repaint();

                //frame stall
                long frameerror = FRAMES - (System.nanoTime() - start);
                if (frameerror > 0) {
                    try {
                        Thread.sleep(frameerror / 10000L, (int) frameerror %
10000);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }

            }
        }

        /*
         * when dies, stops everything and then waits for restart
         */
```

```
private void endGame() {
	isGame = false;
}

/*
 * the update loop for the game
 */
private void updateGame() {
	if (exit) {
		System.exit(0);
	}
	// For In Game

	// Waiting for game selection
	if (!isGenerated()) {
		if (initseedstring) {
			if (seed.equals("Insert String")) {
				seed = "";
			}
			initseedstring = false;
		}

		if (waitingforChoose) {
		}
		spawner.update2(this);
	}

	if (isGame && !paused && !restart && isGenerated() && !endGame) {

		getActor().addAll(toAddActor);
		toAddActor.clear();
		checkMouseDown();
		// Time based stuff
		spawner.update();
		lineSoFar = spawner.getLineSoFar();
		TotalLines = spawner.getlevelsize();
		// Score Counter every second
		if (StarTimer.getSinceStart() % 60 == 0) {
			score += 10;
		}
		// Bullet Regeneration
		if (StarTimer.getSinceStart() % BULLET_REGEN == 0) {
			if (BULLET_MAX - bulletCount < BULLET_MAX) {
				bulletCount--;
			}
		}
```

```
                        StarTimer.addSinceStart();
                        framesSoFar++;
                        // Update Methods for Stars (Twinckles
                        for (Star i : starlist) {
                                i.update(this);
                        }
                        // Updates All actors Players,Astroids, Bullets,
                        for (Actor i : actor) {
                                i.update(this);
                        }
                        // Scans through all current actor objects for collisions, by comparing if
two objects have collided
                        for (int i = 0; i < getActor().size(); i++) {
                                Actor temp1 = getActor().get(i);
                                for (int i2 = i + 1; i2 < getActor().size(); i2++) {
                                        Actor temp2 = getActor().get(i2);
                                        if (i != i2 && temp1.colliding(temp2)) {
                                                temp1.collided(temp2, this);
                                                temp2.collided(temp1, this);
                                        }
                                }
                        }
                        Iterator<Actor> iter = actor.iterator();
                        while (iter.hasNext()) {
                                Actor lookingat = iter.next();
                                if (lookingat.getRemoval()) {
                                        iter.remove();
                                }
                        }
                }
        }

        private void checkMouseDown() {
                // TODO Auto-generated method stub
                if (mouseDown) {
                        if (framesSoFar % GUN_CAP == 0)
                                shoot();

                }
        }

        private void shoot() {
                if (bulletCount < BULLET_MAX) {
                        if (!gameOver) {
                                Bullet bullet = new Bullet(this);
                                bulletCount++;
```

```
                        toAddActor.add(bullet);
                    }
                }
            }
        }

        /*
         * restart loop for the game,
         */

        /*
         * main methods, starts the entire program
         */
        public static void main(String[] args) {

            Dictator newAsteroidsGame = new Dictator();
            newAsteroidsGame.startGame();

        }

        public String getScore() {
            return Integer.toString(score);
        }

        public List<Actor> getActor() {
            return actor;
        }

        public void setActor(List<Actor> actor) {
            this.actor = actor;
        }

        public void addToAddActors(Actor actor) {
            toAddActor.add(actor);
        }

        public int getTime() {
            return StarTimer.getSinceStart();
        }

        public void crash() {
            // TODO Auto-generated method stub
            if (lives == 0) {
                endGame();
                gameOver = true;
            } else {
                lives--;
```

```
                        StarCaptain.center(this);

                }
        }

        public Position getPlayerPosition() {
                // TODO Auto-generated method stub
                Position a = StarCaptain.getPosition();
                return a;
        }

        public boolean isGenerated() {
                return generated;
        }

        public void setGenerated(boolean generated) {
                this.generated = generated;
        }

}
```