```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Collisions;

import GameData.Terrain;
import GameObjects.GameObject;
import GameObjects.Physics;
import java.util.ArrayList;

/**
 *
 * @author o_0
 * Handel all collision dispatch, and checks all types of objects
 */
public class Collisions {
    private ArrayList<GameObject> gameObjects;
    private Terrain terrain;
    /**
     *
     * @param terrain the tarrain to perfome collision check against
     * @param gameObjects all active game objects in the world
     */
    public Collisions(Terrain terrain, ArrayList<GameObject> gameObjects) {
        this.terrain = terrain;
        this.gameObjects = gameObjects;
    }
    /**
     * Checks if 2 objects overlaps
     * @param objA object A to check against B
     * @param objB object B to check againts A
     * @return true if they overlapp
     */
    private boolean checkIntersect(Physics objA, Physics objB) {
        double diffX = objA.getX() - objB.getX();
        double diffY = objA.getY() - objB.getY();

        double distance = Math.sqrt(diffX * diffX + diffY * diffY);
        if((objA.getBodyRadius() + objB.getBodyRadius()) > distance) {
            return true;
        }
        return false;
    }
    /**
     * Checks all GameObjects against ObjectA
     * @param objA gameObject to test
     * @param startIndex  the start index in gameObjects, to prevent recheck
     */
    private void checkCollisionWithObject(Physics objA, int startIndex) {
        for (int j = startIndex + 1; j < gameObjects.size(); j++) {
            GameObject objB = gameObjects.get(j);
            if (!objB.physicsEnable()) {
                continue;
            }
            if(checkIntersect(objA, (Physics) objB)) {
                objA.collisionWith((Physics) objB);
                ((Physics) objB).collisionWith(objA);
            }

        }
    }

    /**
     * Checks all collisions between all active GameObjects
     */
    public void checkAllCollisions() {
        int size = gameObjects.size();
        for (int i = 0; i < size; i++) {
            GameObject objA = gameObjects.get(i);
            if (!objA.physicsEnable()) {
                continue;
            }
            checkCollisionWithObject((Physics) objA, i );
        }
    }
    /**
     * Test all gameObjects if they are colliding with the terrain
     */
    public void checkTerrainCollisions() {
        for(GameObject obj : gameObjects) {
            if(obj.physicsEnable()) {
                terrain.checkCollision((Physics) obj);
            }
        }
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Controller;

import GameObjects.GameObject;
import GameObjects.Projectile;
import java.util.Observable;

/**
 * This is a Observable , that will notify all observers to a new explosion
 * @author o_0
 */
public class ExplosionObserver extends Observable {
    Projectile projectile = null;
    /**
     * This will alert all observers of an explosion
     * @param projectile
     */
    public void explode(Projectile projectile) {
        this.projectile = projectile;
        this.setChanged();
        this.notifyObservers();
    }

    /**
     * Checks if this is a explosive object like a Projectile,
     * if so then this will get triggerd
     * @param obj the object to check if it going to explode
     */
    public void checkTrigger(GameObject obj) {
        if (obj instanceof Projectile) {
            this.explode((Projectile) obj);
        }
    }
    /**
     * used to get the projectile that exploded
     * @return the projectile that exploded
     */
    public Projectile getProjectile() {
        return this.projectile;
    }
}

package Controller;

import GameObjects.GameObject;
import GameObjects.Physics;
import GameData.GameModel;
import java.util.ArrayList;

/**
 * The gameController for the game, it ties the gameModel and ExplosionObserver together
 * @author o_0
 */
public class GameController {

    private GameModel gameModel;
    private ExplosionObserver explosionObserver;
    private GameStatsObservable gameStatsObservable;

    /**
     *
     * @param gameModel the gameModel, that has all gamelogic
     * @param explosionObserver the explosion tracker
     */
```

```
public GameController(GameModel gameModel, ExplosionObserver explosionObserver) {
    this.gameModel = gameModel;
    this.explosionObserver = explosionObserver;
    this.gameStatsObservable = gameStatsObservable;
}

/**
 * Add more observers to the
 * @param newObservers explosionObserver
 */
public void addObservers(ArrayList<GameObject> newObservers) {
    for (GameObject obj : newObservers) {
        if (obj.physicsEnable()) {
            explosionObserver.addObserver((Physics) obj);
        }
    }
}

/**
 * Used to remove a observer from the explosionObserver
 * @param obj object to remove
 */
private void removeObserver(GameObject obj) {
    if (!obj.physicsEnable()) {
        return;
    }
    explosionObserver.checkTrigger(obj);
    explosionObserver.deleteObserver((Physics) obj);
}

/**
 * Removes all inActive objects from the explosionObserver
 * @param inActiveObs
 */
public void removeObservers(ArrayList<GameObject> inActiveObs) {
    if (inActiveObs.isEmpty()) {
        return;
    }
    for (GameObject observer : inActiveObs) {
        removeObserver(observer);
    }
}

/**
 * Returns a list of all newly inactive gameObjects
 * @return a list of all newly inactive gameObjects
 */
public ArrayList<GameObject> removeInactiveObjects() {
    return gameModel.reapInactiveObjects();
}

/**
```

```
 * Tells the model to do all collisionChecks
 */
public void gameCollisionUpdate() {
    gameModel.checkCollisions();
}

/**
 * Updates all game data, and logic
 * @param frameDelta the fraction of a seconde that has passed sence last update
 * @return all new objects that has spawned this updateCycle
 */
public ArrayList<GameObject> updateGame(double frameDelta) {
    gameModel.updateAiPlayers(frameDelta);
    return gameModel.updateGameobjects(frameDelta);
}

/**
 * Adds objects to the gameModel
 * @param newObjects
 */
public void addObjects(ArrayList<GameObject> newObjects) {
    gameModel.addObjects(newObjects);
}

/**
 * Respawns all inactive players, (if they died or something)
 */
public void playerRespawn() {
    if (gameModel.deathCheck()) {
        ArrayList<GameObject> respawned;
        respawned = gameModel.reSpawnPlayers();
        this.addObjects(respawned);
        this.addObservers(respawned);
    }

}

/**
 * Creates a new spawnBox
 * @return returns a spawnBox, with random stuff according to the model
 */
public GameObject makeSpawnBox() {
    return gameModel.spawnBox();
}

/**
 * Checks if a player has been inactive
 * @return true if someone is inactive
 */
public boolean deathcheck() {
    return gameModel.deathCheck();
}
```

```
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Controller;

import java.util.Observable;

/**
 *a Observable used to help keep track of health and ammo on
 * the players
 * @author mats
 */
public class GameStatsObservable extends Observable {

    /**
     * Method that gets call when changes have been made,
     * after this it notifies the observing classes
     */
    public void checkUIInfo(){
        this.setChanged();
        this.notifyObservers();
    }

}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Controller;

import GameObjects.Direction;
import GameObjects.Player;
import javafx.event.EventHandler;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;

/**
 * This is the main keyboard input controller, this tracks all keyboard settings
 * and builds all events handlers for it
 * @author o_0
 */
public class KeyboardController {

    private KeyCode moveLeft = KeyCode.A;
    private KeyCode moveRight = KeyCode.D;
    private KeyCode aimUp = KeyCode.W;
    private KeyCode aimDown = KeyCode.S;
    private KeyCode jetpackOn = KeyCode.SPACE;
    private KeyCode fireWeapon = KeyCode.SHIFT;

    /**
     * Create a new keyboardController, with these keys
     * @param moveLeft move unit left
     * @param moveRight move unit rigth
     * @param aimUp aim up
     * @param aimDown aim down
     * @param fire fire the weapon
     * @param jetpackOn turns on jetpack
     */
    public KeyboardController(KeyCode moveLeft, KeyCode moveRight, KeyCode aimUp,
KeyCode aimDown,KeyCode fire, KeyCode jetpackOn) {
        this.moveLeft = moveLeft;
        this.moveRight = moveRight;
        this.jetpackOn = jetpackOn;
        this.aimUp = aimUp;
        this.aimDown = aimDown;
        this.fireWeapon = fire;
    }

    /**
     * Setter for key action
     * @param moveLeft
     */
    public void setKeyMoveLeft(KeyCode moveLeft) {
        this.moveLeft = moveLeft;
    }

    /**
     * Setter for key action
     * @param moveRight
     */
    public void setKeyMoveRight(KeyCode moveRight) {
        this.moveRight = moveRight;
    }

    /**
     * Setter for key action
     * @param jetpackOn
     */
    public void setKeyJetpackOn(KeyCode jetpackOn) {
        this.jetpackOn = jetpackOn;
    }

    /**
     * Setter for key action
     * @param aimUp
     */
    public void setKeyAimUp(KeyCode aimUp) {
        this.aimUp = aimUp;
    }
```

```
/**
 * Setter for key action
 * @param aimDown
 */
public void setKeyAimDown(KeyCode aimDown) {
    this.aimDown = aimDown;
}

/**
 * Setter for key action
 * @param fire
 */
public void setKeyFireWeapon(KeyCode fire) {
    this.fireWeapon = fire;
}

/**
 * This creates a new keyboard event handler, for a specifik player used for
 * key pressed down
 * @param player the player this input should be tied to
 * @return EventHandler<KeyEvent> to be used to control the player
 */
public EventHandler<KeyEvent> getPlayerKeyPressedHandler(Player player) {
    return new EventHandler<KeyEvent>() {
        @Override
        public void handle(KeyEvent event) {
            if (event.getCode() == fireWeapon) {
                player.fireWeapon();
            }

            if (event.getCode() == moveLeft) {
                player.setDirection(Direction.LEFT);
            }

            if (event.getCode() == moveRight) {
                player.setDirection(Direction.RIGHT);
            }

            if (event.getCode() == aimUp) {
                player.setAim(Direction.UP);
            }

            if (event.getCode() == aimDown) {
                player.setAim(Direction.DOWN);
            }

            if (event.getCode() == jetpackOn) {
                player.setJetpackState(true);
            }
        }
    };
    }
    /**
     * * This creates a new keyboard event handler, for a specifik player used for
     * key released
     * @param player the player this input should be tied to
     * @return EventHandler<KeyEvent> to be used to control the player
     */
    public EventHandler<KeyEvent> getPlayerKeyReleasedHandler(Player player) {
        return new EventHandler<KeyEvent>() {
            @Override
            public void handle(KeyEvent event) {
                if (event.getCode() == moveLeft) {
                    player.setDirection(Direction.NONE);
                }

                if (event.getCode() == moveRight) {
                    player.setDirection(Direction.NONE);
                }
                if (event.getCode() == aimUp) {
                    player.setAim(Direction.NONE);
                }

                if (event.getCode() == aimDown) {
                    player.setAim(Direction.NONE);
                }

                if (event.getCode() == jetpackOn) {
                    player.setJetpackState(false);
                }
            }
        };
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Controller;

import GameData.GraphicModels;
import GameData.Terrain;
import GameObjects.GameObject;
import GameTimers.RenderTimer.RenderingState;
import View.GameView;
import javafx.scene.image.Image;

/**
 * This is the controller that ties the view and the graphicModel together
 * This is where all things that needs to be rendered passes thru
 * @author o_0
 */
```

```
public class RenderController{
    private GameView gameView;
    private GraphicModels graphicModel;
    /**
     *
     * @param view the view used to render the scene
     * @param graphicModel contains all models in use, and animation data
     */
    public RenderController(GameView view, GraphicModels graphicModel) {
        super();
        this.gameView = view;
        this.graphicModel = graphicModel;
    }
    /**
     * will trigger the view to render the terrain
     * @param terrain to be rendered
     */
    public void displayTerrain(Terrain terrain) {
        gameView.drawterrain(terrain.getTerrainImage());
    }
    /**
     * will trigger the view to render the one object
     * @param obj the object to be rendered
     * @param state state data used for scaling, and keyframe animation info
     */
    public void displayModel(GameObject obj,RenderingState state) {
        Image model = graphicModel.getModel(obj.getModelID(), 0);
        double scaling = state.getScaleFactor(model.getWidth(), model.getHeight());
        gameView.drawmodel(model, obj.getX(), obj.getY(), scaling);
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameData;

import GameObjects.Direction;
import GameObjects.GameObject;
import GameObjects.Player;
import GameObjects.SpawnBox;
import java.util.ArrayList;
import javafx.geometry.Point2D;

/**
 * the main ai, based on a state machine, with a stack to do strategies
 * @author o_0
 */
public class Ai {

    private Player aiPlayer;
    private ArrayList<Player> enemyList;
    private ArrayList<GameObject> gameObjects;
    private double gameTime = 0;
    private Point2D collisionPoint;
    private Point2D destination;
    private double fireDelay;

    /**
     * diffrent ai states types
     */
    private enum AiState {

        HUNT, SEARCH, AVOID_TERRAIN, GETSPAWNBOX,MOVETOPOINT
    };

    /**
     * all states have a timespamp and a type this is used to model it
     */
    private class StateData {

        private AiState state;
        private double timeStamp;

        protected StateData(AiState state, double timeStamp) {
            this.state = state;
            this.timeStamp = timeStamp;
        }

        protected AiState getState() {
            return this.state;
        }

        protected double getTimeStamp() {
            return this.timeStamp;
        }
        protected void setTimeStamp(double time) {
            this.timeStamp = time;
        }
    }

    private ArrayList<StateData> stateStack = null;
    private GameObject target = null;

    /**
     * creates a new ai player
     * @param player the player the ai should control
     * @param enemyList a list with all the enemys to hunt
     * @param gameObjects all game objects
     */
    public Ai(Player player, ArrayList<Player> enemyList, ArrayList<GameObject> gameObjects) {
        this.aiPlayer = player;
        this.enemyList = enemyList;
        this.gameObjects = gameObjects;
        this.collisionPoint = new Point2D(0, 0);
        this.destination = new Point2D(0, 0);
        this.target = null;
        this.stateStack = new ArrayList<StateData>();
        this.stateStack.add(new StateData(AiState.HUNT, 0));

    }

    /**
     * moves to a point, xy
     * @param destX x coordinate
     * @param destY y coordinate
     */
    private void moveTo(double destX, double destY) {
        if (destY > 0) {
            aiPlayer.setJetpackState(true);
        } else {
            aiPlayer.setJetpackState(false);
        }
        double maxX = Math.abs(aiPlayer.currentDx());
        /*if(maxX > 10) {
            aiPlayer.setDirection(Direction.NONE);
        }*/
        if (destX > 0 && aiPlayer.currentDx() < 10) {
            aiPlayer.setDirection(Direction.LEFT);
        } else if (aiPlayer.currentDx() > -10) {
            aiPlayer.setDirection(Direction.RIGHT);
        } else {
            aiPlayer.setDirection(Direction.NONE);
        }

    }

    /**
     * Move to a point that was set this.destination
     */
    private void moveToPoint() {
        int index = stateStack.size() - 1;
        if (index < 0) {
            return;
        }
        StateData stateData = stateStack.get(index);
        double diffX = Math.abs(aiPlayer.getX() - destination.getX());
        double diffY = Math.abs(aiPlayer.getY() - destination.getY());
        if(diffX + diffY < 10 || gameTime - stateData.timeStamp > 1 ){
            stateStack.remove(index);
            return;
        }
        moveTo(destination.getX(),destination.getY());
    }
```

```
/**
 * Hunts a target, and moves towards it and tryies to shoot enemys
 */
private void huntTarget() {

    int index = stateStack.size() - 1;
    if (index < 0) {
        return;
    }
    if (this.target == null && stateStack.get(index).getState() != AiState.SEARCH) {
        stateStack.add(new StateData(AiState.SEARCH, gameTime));
        return;
    }
    StateData stateData = stateStack.get(index);
    double timeDiff = this.gameTime - stateData.timeStamp;
    if(timeDiff > 20 ) {
        stateData.setTimeStamp(gameTime); // reset this state clock
        this.destination = new Point2D(1000*Math.random(),1000*Math.random());
        stateStack.add(new StateData(AiState.MOVETOPOINT, gameTime));
        return;
    }
    double aiLocX = aiPlayer.getX();
    double aiLocY = aiPlayer.getY();
    double threatX = 0;
    double threatY = 0;
    for (Player enemy : enemyList) {
        threatX += aiLocX - enemy.getX();
        threatY += aiLocY - enemy.getY();
    }
    if(threatY > 30) {
        aiPlayer.setAim(Direction.UP);
    }else if(threatY < -30){
        aiPlayer.setAim(Direction.DOWN);
    }else{
        aiPlayer.setAim(Direction.NONE);
    }
    if(this.fireDelay > 1.0) {
        aiPlayer.fireWeapon();
        this.fireDelay = 0;
    }


    moveTo(threatX, threatY);
}

/**
 * moves to a spawn box to pickup weapons/ammo
 */
private void moveToSpawnBox() {
    int index = stateStack.size() - 1;
    if (index < 0) {
        return;
    }
    StateData stateData = stateStack.get(index);
    double timeDiff = this.gameTime - stateData.timeStamp;
    if (timeDiff > 5) {
        stateStack.remove(index);
        return;
    }
    if (target == null) {
        stateStack.remove(index);
        return;
    }
    moveTo(target.getX(), target.getY());
}

/**
 *
 * @return if it found a box sets it to the new target
 */
private boolean searchSpawnBox() {
    for (GameObject obj : gameObjects) {
        if (obj instanceof SpawnBox) {
            this.target = obj;
            return true;
        }
    }
    return false;
}

/**
 * search for a target to hunt/go too
 */
private void searchTarget() {
    double distance = 10000;
    int index = stateStack.size() - 1;
    if (index < 0) {
        return;
    }
    if (aiPlayer.currentAmmo() < 4) {
        if(searchSpawnBox()) {
            stateStack.set(index, new StateData(AiState.GETSPAWNBOX, gameTime));
            return;
        }

    }
    for (Player enemy : enemyList) {
        double threatX = aiPlayer.getX() - enemy.getX();
        double threatY = aiPlayer.getY() - enemy.getY();
        double tmpDistance = Math.sqrt(threatX * threatX + threatY * threatY);
        if (tmpDistance < distance) {
            this.target = enemy;
            distance = tmpDistance;
        }
    }
    if (this.target != null) {
        stateStack.remove(index);
    }

}

/**
 * avoid terain
 */
private void avoidTerrain() {
    int index = stateStack.size() - 1;
    if (index < 0) {
        return;
    }
    StateData stateData = stateStack.get(index);
    double timeDiff = this.gameTime - stateData.timeStamp;

    // timed out on this operation
    if (timeDiff > 0.5 ||  Math.abs(aiPlayer.currentDx()) > 15
            || Math.abs(aiPlayer.currentDy()) > 15) {
        stateStack.remove(index);
        return;
    }
    double diffX = aiPlayer.getX() - collisionPoint.getX();
    double diffY = aiPlayer.getY() - collisionPoint.getY();
    moveTo(aiPlayer.getX() + 2*diffX, aiPlayer.getY() + 2*diffY);

}

/**
 * collision point, where the ai collided with the terrain
 * @param x
 * @param y
 */
public void collisionWithTerrainAt(double x, double y) {
    this.collisionPoint = new Point2D(x, y);
    int index = stateStack.size() - 1;
    if (index >= 0 && stateStack.get(index).getState() != AiState.AVOID_TERRAIN) {
        StateData stateData = new StateData(AiState.AVOID_TERRAIN, gameTime);
        stateStack.add(stateData);
    }

}

/**
 * called if ai piced up a spawn box, removes this state if this was its mission
 */
public void pickedupSpawnBox() {
    int index = stateStack.size() - 1;
    if (index < 0) {
        return;
        }
        if(stateStack.get(index).state == AiState.GETSPAWNBOX){
            stateStack.remove(index);
        }
    }

    /**
     * update the ai state machine
     * @param frameDelta
     */
    public void updateAi(double frameDelta) {
        this.gameTime += frameDelta;
        this.fireDelay +=frameDelta;
        if (stateStack.isEmpty()) {
            stateStack.add(new StateData(AiState.HUNT, 0));
        }
        int index = stateStack.size() - 1;
        StateData stateData = stateStack.get(index);
        switch (stateData.state) {
            case HUNT:
                huntTarget();
                break;
            case AVOID_TERRAIN:
                avoidTerrain();
                break;
            case SEARCH:
                searchTarget();
                break;
            case GETSPAWNBOX: moveToSpawnBox(); break;
            case MOVETOPOINT: moveToPoint();break;
            default:
                huntTarget();
                break;
        }

    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameData;

import Collisions.Collisions;
import GameObjects.GameObject;
import GameObjects.Player;
import GameObjects.ProjectileType;
import GameObjects.SpawnBox;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

/**
 * This is the main Game Model has all game logic, and data, knows how to update
 * diffrent game objects
 * @author o_0
 */
public class GameModel {

    public static final double BILLION = 1000_000_000.0; //from Ball lab2b
    private long lastTime;
    private ArrayList<GameObject> gameObjects;
    private ArrayList<Ai> aiPlayers;
    private Collisions collisions;
    private Terrain terrain;
    private double heigth = 10;
    private double width = 10;
    private ArrayList<Player> currentPlayers;
    private Random rand;

    /**
     *
     * @param width the map width
     * @param heigth the map height
     * @param gameObj container for all gameObjects
     * @param aiPlayers ai players
     * @param collisions collision handler
     * @param terrain the game terrain
     */
    public GameModel(double width, double heigth, ArrayList<GameObject> gameObj,
ArrayList<Ai> aiPlayers, Collisions collisions, Terrain terrain) {
        super();
        this.width = width;
        this.heigth = heigth;
        this.gameObjects = gameObj;
        this.collisions = collisions;
        this.terrain = terrain;
        this.aiPlayers = aiPlayers;
        this.rand = new Random();
        this.currentPlayers = new ArrayList<Player>();
        this.findeplayers();
    }

    /**
     * Will remove all inactive objects from the gameModel, this will cause them
     * to not be rendered or get called in update
     * @return all objects that has been removed
     */
    public ArrayList<GameObject> reapInactiveObjects() {
        ArrayList<GameObject> removed = new ArrayList<GameObject>();
        // removes all inactive objects
        Iterator<GameObject> it = gameObjects.iterator();
        while (it.hasNext()) {
            GameObject obj = it.next();
            if (!obj.isActive()) {
                //removeObservers(obj);
                removed.add(obj);
                it.remove();
            }
        }
        return removed;
    }

    /**
     * this will add all players into currentPlayers that exists in gameObjects
     * Only call this once from constructor
     */
    private void findeplayers() {
        for (GameObject obj : this.gameObjects) {
            if (obj instanceof Player) {
                currentPlayers.add((Player) obj);
            }
        }
    }

    /**
     * Checks for inactive players
     * @return true if any player has died
     */
    public boolean deathCheck() {
        for (Player player : currentPlayers) {
            if (!player.isActive()) {
                return true;
            }
        }
        return false;
    }

    /**
     * Resets the players info, and returns a list of all reset players to be
     * added back into game
     * @return all players to be reset in ArrayList
     */
    public ArrayList<GameObject> reSpawnPlayers() {
        ArrayList<GameObject> respawned = new ArrayList<GameObject>();
        for (Player player : currentPlayers) {
            if (!player.isActive()) {
                double newX = rand.nextDouble() * 1000;
                double newY = rand.nextDouble() * 50;
                player.resetPlayer(newX, newY, 100);
                respawned.add(player);
            }
        }
        return respawned;
```

```
}

/**
 * creates a random spawnbox with ammo and weapons
 * @return a new game objects to be added to game
 */
public GameObject spawnBox() {
    double newX = rand.nextDouble() * 1000;
    double newY = rand.nextDouble() * 50;

    SpawnBox box;
    int type = rand.nextInt(3);
    switch (type) {
        case 0:
            box = new SpawnBox(ProjectileType.GRANADE, newX, newY);
            break;
        case 1:
            box = new SpawnBox(ProjectileType.BULLET, newX, newY);
            break;
        case 3:
            box = new SpawnBox(ProjectileType.MISSILE, newX, newY);
            break;
        default:
            box = new SpawnBox(ProjectileType.MISSILE, newX, newY); break;

    }
    return box;
}

/**
 * perfomrce all collision checks for the game
 */
public void checkCollisions() {
    collisions.checkAllCollisions();
    collisions.checkTerrainCollisions();
}

/**
 * Updates all ai in the game, and their logic
 * @param frameDelta fractions since last update
 */
public void updateAiPlayers(double frameDelta) {
    for (Ai ai : aiPlayers) {
        ai.updateAi(frameDelta);
    }
}

/**
 * updates all gameObjects, stats,positions, constraints
 * @param frameDelta fraction since last update
 * @return a list with all newly created objects
 */
```

```
    public ArrayList<GameObject> updateGameobjects(double frameDelta) {
        ArrayList<GameObject> spawnedObj = new ArrayList<GameObject>();
        for (GameObject obj : gameObjects) {
            obj.update(frameDelta, spawnedObj);
            obj.constrain(width, heigth);
        }
        return spawnedObj;
    }

    /**
     * Adds new objects to the models, after this they are active, and actively updated
     * @param newObjects all objects to be added
     */
    public void addObjects(ArrayList<GameObject> newObjects) {
        // adds all spawned objects
        if (!newObjects.isEmpty()) {
            gameObjects.addAll(newObjects);
        }
    }
}


package GameData;

import java.util.ArrayList;
import javafx.scene.image.Image;

/**
 * Stores all image/model data used by the rendering
 * also can preform animations, if added =)
 * @author mats
 */
public class GraphicModels {

    private ArrayList<Image> graphicModels;
    /**
     * Constructor
     */
    public GraphicModels() {
        graphicModels = new ArrayList<Image>();
    }

    /**
     * Loads all modelNames into this class, for later use
     * @param modelNames String array of all models
     */
    public void loadmodel(String[] modelNames){
        for(String name: modelNames) {
            graphicModels.add(new Image(name));
        }
    }
    /**
     * Gets the image for with the modelId, and state
     * @param modelID what model to use
     * @param state can later be used to select key frames for a model
     * @return
     */
    public Image getModel(int modelID, int state){
        return graphicModels.get(modelID);
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameData;

import Controller.ExplosionObserver;
import GameObjects.Physics;
import GameObjects.Projectile;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;
import javafx.embed.swing.SwingFXUtils;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import javafx.scene.image.PixelReader;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javax.imageio.ImageIO;

/**
 * This is the class used for terrain for the game,
 * it can load and save the current terrain, for later playability
 * it has destructible terrain, and listens for explosions
 * @author o_0
 */
public class Terrain implements Observer {

    private Canvas mapCanvas;
    private Image background;
    private ArrayList<Circle> craters;
    double mapWidth;
    double mapHeigth;
    private boolean needsUpdate = false;

    /**
     *
     * @param width the size of the terrain, width
     * @param heigth size for height
     */
    public Terrain(double width, double heigth) {
        this.craters = new ArrayList<Circle>();
        this.background = new Image("Resource/battleTerrain.png");  //defualt terrain
        this.mapHeigth = heigth;
        this.mapWidth = width;
        this.mapCanvas = new Canvas(mapWidth, mapHeigth);
        //this.background = Image();
        GraphicsContext gc = this.mapCanvas.getGraphicsContext2D();
        gc.drawImage(background, 0, 0, mapCanvas.getWidth(), mapCanvas.getHeight());

    }

    /**
     * This will load a map, it will only load from the game directory
     * @param file the file info picked by fileChooser
     * @throws IOException if we fail to load our terrain
     */
    public void loadTerrain(File file) throws IOException {
        if (file == null) {
            return;
        }
        //Image tmpImg = background;
        String path = file.toURI().toString();
        this.background = new Image(path);
        this.needsUpdate = true;
        craters.clear();
    }

    /**
     * Saves the terrain for later loading
     * @param file the file to be saved
     * @throws IOException if it fails to save the file
     */
    public void saveTerrain(File file) throws IOException {
        if (file == null) {
            return;
        }

        BufferedImage buffer = SwingFXUtils.fromFXImage(this.background, null);
        ImageIO.write(buffer, "png", file);

    }

    /**
     * Checks if a point is inside the background
     * @param x x pos
     * @param y y pos
     * @return true if its valid point
     */
    private boolean isValidPoint(int x, int y) {
        if (x < 0 || background.getWidth() <= x || y < 0 || background.getHeight() <= y) {
            return false;
        }
        return true;
    }

    /**
     * This checks if a object is in contact with the terrain,
     * it does this by checking the pixels in the background if it is a skypixel
     * the pixels is based on the objects x,y and radius
     * @param obj object to check if it collides
     * @return true if it did collide
     */
    public boolean checkCollision(Physics obj) {
        PixelReader pr = background.getPixelReader();
        double scalingX = background.getWidth() / this.mapWidth;
        double scalingY = background.getHeight() / this.mapHeigth;
        Color sky = Color.LIGHTSKYBLUE;
        int centerX = (int) (obj.getX() * scalingX);
        int centerY = (int) (obj.getY() * scalingY);
        int radius = (int) (obj.getBodyRadius() * scalingX);

        int hitCount = 0;
        double avgX = 0;
        double avgY = 0;

        int tmpX = centerX;
        int tmpY = centerY - radius;
        if (isValidPoint(tmpX, tmpY) && !pr.getColor(tmpX, tmpY).equals(sky)) {
            avgX += tmpX;
            avgY += tmpY;
            hitCount++;
        }

        tmpX = centerX - radius;
        tmpY = centerY;
        if (isValidPoint(tmpX, tmpY) && !pr.getColor(tmpX, tmpY).equals(sky)) {
            avgX += tmpX;
            avgY += tmpY;
            hitCount++;
        }

        tmpX = centerX + radius;
        tmpY = centerY;
        if (isValidPoint(tmpX, tmpY) && !pr.getColor(tmpX, tmpY).equals(sky)) {
            avgX += tmpX;
            avgY += tmpY;
            hitCount++;
        }
        tmpX = centerX;
        tmpY = centerY + radius;
        if (isValidPoint(tmpX, tmpY) && !pr.getColor(tmpX, tmpY).equals(sky)) {
            avgX += tmpX;
            avgY += tmpY;
            hitCount++;
        }
        if (hitCount <= 0) {
            return false;
        }
        avgX = avgX / hitCount;
        avgY = avgY / hitCount;

        // tells the obj where the collision point is, scaled to game coordinates
        obj.collisionWithTerrainAt(avgX / scalingX, avgY / scalingY);

        return true;
    }

    /**
     * This returns the terrainImage, it only renders it into a new image
     * if something has changed, else it reuses the same image
     * @return returns the terrainImage
     */
    public Image getTerrainImage() {
        if (!this.needsUpdate) {
            return background;
        }
        this.needsUpdate = false;
        // Draw a cleen background
        GraphicsContext gc = this.mapCanvas.getGraphicsContext2D();
        gc.clearRect(0, 0, mapWidth, mapHeigth);
        gc.drawImage(background, 0,0,this.mapWidth, this.mapHeigth);

        // for all new creaters, paint sky color with crater radius,
        gc.setFill(Color.LIGHTSKYBLUE);   //
        for (Circle crater : craters) {
            double radius = crater.getRadius();
            double x = crater.getCenterX() - radius;
            double y = crater.getCenterY() - radius;
            gc.fillOval(x, y, 2 * radius, 2 * radius);
            //gc.strokeOval(x,y, 2*radius, 2*radius);

        }
        craters.clear();
        // save canvase to a new background image
        background = mapCanvas.snapshot(null, null);
        return background;
    }

    /**
     * Is notified if a explosion has happen
```

```
     * Adds all explosions to a crater
     * @param o the observable
     * @param arg not used
     */
    @Override
    public void update(Observable o, Object arg) {
        Projectile exploded = ((ExplosionObserver) o).getProjectile();
        Circle crater = new Circle(exploded.getX(),
                exploded.getY(),
                exploded.getDamageRadius());
        craters.add(crater);
        this.needsUpdate = true;
    }

}


package GameObjects;

/**
 * all directions, that is used
 * @author o_0
 */
public enum Direction {LEFT,RIGHT,UP,DOWN,NONE};


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import java.util.ArrayList;

/**
 * this handels the explosion object,
 * It also changes its size and grows with time
 * @author o_0
 */
public class Explosion extends GameObject{

    private double timeToLive = 0.5;
    private double kaboomSize = 1.0;
    private double tickScaling = 1;
    /**
     *
     * @param x position
     * @param y position
     * @param kaboomSize how big it going to be
     */
    public Explosion(double x, double y,double kaboomSize) {
        super(2);
        this.setX(x);
        this.setY(y);
        this.kaboomSize = kaboomSize;
        this.tickScaling = (0.5*kaboomSize)/timeToLive;
        this.setBodySize(kaboomSize/2);
    }

    /**
     * Updates the Explosion and makes it grow, and then deactivate it self
     * @param frameDelta fraction of seconde since last update
     * @param spawnedObj not used
     * @return
     */
    @Override
    public boolean update(double frameDelta, ArrayList<GameObject> spawnedObj) {
        timeToLive -= frameDelta;
        this.setBodySize(this.getBodySize() + tickScaling*frameDelta);
        if(timeToLive < 0) {
            this.deactivate();
        }
        return true;
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import java.util.ArrayList;

/**
 * The main gameObject in the game, this absractclass is the base for all objects in game
 * @author o_0
 */
public abstract class GameObject {
    private int modelID;
    private boolean active;
    //private boolean physicsEnable = false;
    private double x = 0.0;
    private double y = 0.0;
    private double bodySize = 0.0;
    public double getX(){return this.x;}
    public double getY(){return this.y;}
    protected void setX(double x){this.x = x;}
    protected void setY(double y){this.y = y;}
    public double getBodySize() {return this.bodySize;}
    protected void setBodySize(double bodySize) {this.bodySize = bodySize;}
```

```java
/**
 *
 * @param modelId what model it has
 */
protected GameObject(int modelId) {
    this.modelID = modelId;
    this.active = true;
}
/**
 * This is called every frame by gameTimer
 * @param frameDelta fraction of a seconde since last update
 * @param spawnedObj list to store all newly created objects
 * @return not used
 */
public abstract boolean update(double frameDelta, ArrayList<GameObject> spawnedObj);
/**
 *
 * @return true if this unit is active
 */
public boolean isActive() {
    return this.active;
}

/**
 * set this object to inactive
 */
protected void deactivate() {
    this.active = false;
}
/**
 * reactivate object
 */
protected void reActivate(){
    this.active = true;
}

/**
 *
 * @return if this is a physics object, so the more expencive instanceof do not
 * need to be used
 */
public boolean physicsEnable() { return false;};

/**
 *
 * @return the modelid
 */
public int getModelID() {
    return this.modelID;
}
/**
 * Sets a new modelId
```

```
     * @param modelId the new modelId
     */
    public void setModelID(int modelId) {
        this.modelID = modelId;
    }

    /**
     * Constrain the GameObject insde the height and width
     * @param w width
     * @param h height
     */
    public void constrain(double w, double h) {
        if(x < 0) {
            this.setX(1);
        }else if(w < x) {
            this.setX(w - 1);
        }
        if(y < 0) {
            this.setY(1);
        }else if(h < y) {
            this.setY(h - 1);
        }
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import Controller.ExplosionObserver;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

/**
 * All objects that has a physics property, movement, gravity and so on subclass this class
 * @author o_0
 */
public abstract class Physics extends GameObject implements Observer {

    private double dx;
    private double dy;
    private boolean gravityAcitve = true;
    private double bodyRadius = 10;
    /**
     * getter for dx
     * @return dx
     */
    protected double getDx() {
```

```
    return dx;
}

/**
 * getter for dy
 * @return
 */
protected double getDy() {
    return dy;
}

/**
 * add value to the dx component
 * @param dx added speed
 */
protected void addToDx(double dx) {
    this.dx += dx;
}

/**
 * add value to the dy component
 * @param dy speed to add
 */
protected void addToDy(double dy) {
    this.dy += dy;
}

/**
 *
 * @param dx start movement x direction
 * @param dy start movement y direction
 * @param bodyRadius the bodyRadius for this object
 * @param modelId the model to be used
 */
protected Physics(double dx, double dy, double bodyRadius, int modelId) {
    super(modelId);
    this.dx = dx;
    this.dy = dy;
    this.bodyRadius = bodyRadius;
    this.setBodySize(bodyRadius*2);
}

/**
 * that this object has physicsEnable
 * @return true
 */
@Override
public boolean physicsEnable() { return true;};

/**
 * set if it uses gravity
 * @param isOn true == on
 */
protected void setGravity(boolean isOn) {
    this.gravityAcitve = isOn;
}

/**
 * updates the gravity effect
 * @param frameDelta diff between frames
 */
protected void updateGravity(double frameDelta) {
    this.dy += 80.0 * frameDelta;
}

/**
 * Sets a new bodyRadius
 * @param bodyRadius new value
 */
protected void setBodyRadius(double bodyRadius) {
    this.bodyRadius = bodyRadius;
    this.setBodySize(bodyRadius);
}

/**
 * getter for bodyRadius
 * @return bodyRadius
 */
public double getBodyRadius() {
    return this.bodyRadius;
}

/**
 * Called if this object collides with anything
 * @param gameObj what it collided with
 */
public abstract void collisionWith(Physics gameObj);

/**
 * The point it hit the terrain at
 * @param x point it hit the terrain at
 * @param y point it hit the terrain at
 */
public abstract void collisionWithTerrainAt(double x, double y);

/**
 * Updates all physics on this object
 * @param frameDelta see super
 * @param spawnedObj see super
 * @return true
 */
@Override
public boolean update(double frameDelta, ArrayList<GameObject> spawnedObj) {
    if (this.gravityAcitve) {
        updateGravity(frameDelta);
    }
    this.setX(this.getX() + dx * frameDelta);
    this.setY(this.getY() + dy * frameDelta);
    return true;
}

/**
 * This is called if there is an explosion
 * @param o what projectile that exploded
 * @param arg not in use
 */
@Override
public void update(Observable o, Object arg) {

    Projectile exploded = ((ExplosionObserver) o).getProjectile();

    double diffX = this.getX() - exploded.getX();
    double diffY = this.getY() - exploded.getY();
    double effectRadius = exploded.getDamageRadius();

    double distance = Math.sqrt(diffX*diffX + diffY*diffY);
    if(distance > effectRadius) {
        return;
    }

    double power = 1 - distance/effectRadius;
    double strength = power * exploded.getDamage();
    this.dx += Math.signum(diffX) * strength;
    this.dy += Math.signum(diffY) * strength;

    if(this instanceof Player) {
        ((Player)this).takeDamage(strength);
    }
}

/**
 * Constrain the GameObject insde the height and width
 * @param w width
 * @param h height
 */
@Override
public void constrain(double w, double h) {
    double x = this.getX();
    double y = this.getY();
    if(x < bodyRadius) {
        this.setX(bodyRadius);
        this.dx = 0;//-dx/2;
    }else if(w - bodyRadius < x) {
        this.setX(w - bodyRadius);
        this.dx = 0;//-dx/2;;
    }
```

```
        if(y < bodyRadius) {
            this.setY(bodyRadius);
            this.dy = 0;//-dy/2;
        }else if(h - bodyRadius < y) {
            this.setY(h - bodyRadius);
            this.dy = 0;//-dy/2;
        }
    }

}

package GameObjects;

import Controller.GameStatsObservable;
import Controller.KeyboardController;
import GameData.Ai;
import java.util.ArrayList;

/**
 * the player that is controlled by an ai or a human
 * @author o_0
 */
public class Player extends Physics {

    private boolean jetpackState = false;
    private String name;
    private Weapon weapon;
    private boolean didFire = false;
    private double health = 100;
    private GameStatsObservable gameStatsObservable = null;
    private int numberofdeaths = 0;

    private Direction dir;
    private Direction aim;

    private Ai aiControl = null;

    /**
     *
     * @param name_ name of player
     * @param x starting x position
     * @param y starting y position
     * @param modelId what model to use
     */
    public Player(String name_, double x, double y, int modelId) {
        super(0, 0, 7, modelId); // should be stationary
        this.setX(x);
        this.setY(y);
        this.dir = Direction.NONE;
        this.aim = Direction.NONE;
        this.name = name_;
        this.weapon = new Weapon(this, ProjectileType.GRANADE, 3);
```

```
}

/**
 * The Observable to be called if some player stats was changed
 * @param o the GameStatsObservable
 */
public void setGameStatsObservable(GameStatsObservable o) {
    this.gameStatsObservable = o;
}

/**
 * set if an ai controls this player
 * @param aiControl the ai
 */
public void setAiControl(Ai aiControl) {
    this.aiControl = aiControl;
}

/**
 * called to take damage on player, if player health < 0 it deactivets
 * @param damage how much
 */
public void takeDamage(double damage) {
    health -= damage;
    System.out.println("player: " + this.name + " Health: " + this.health);
    if (health < 0) {
        this.deactivate();
        numberofdeaths++;
    }
    if (gameStatsObservable != null) {
        this.gameStatsObservable.checkUIInfo();
    }
}

/**
 * Rests the player and activates it with health
 * @param x x coordinate
 * @param y y coordinate
 * @param health what health to start witg
 */
public void resetPlayer(double x, double y, double health) {
    this.setX(x);
    this.setY(y);
    this.weapon = new Weapon(this, ProjectileType.GRANADE, 3);
    this.health = health;
    this.reActivate();
    if (gameStatsObservable != null) {
        this.gameStatsObservable.checkUIInfo();
    }
}

/**
```

```
     *
     * @return how manny times this player has died
     */
    public int getDeaths(){
        return this.numberofdeaths;
    }

    /**
     * Used by ai to get info
     *
     * @return the players dx
     */
    public double currentDx() {
        return this.getDx();
    }

    /**
     * Used by ai to get info
     *
     * @return the players dy
     */
    public double currentDy() {
        return this.getDy();
    }

    /**
     * getter
     * @return currentHealth
     */
    public double currentHealth() {
        return this.health;
    }

    /**
     * getter
     * @return currentAmmo
     */
    public int currentAmmo() {
        return this.weapon.getAmmo();
    }

    /**
     * getter
     * @return currentAimAngle
     */
    public double currentAimAngle() {
        return weapon.getAimAngle();
    }
/**
     * This is called every frame by gameTimer thru the controller
     * @param frameDelta fraction of a seconde since last update
     * @param spawnedObj list to store all newly created objects
```

```
 * @return not used
 */
@Override
public boolean update(double frameDelta, ArrayList<GameObject> spawnedObj) {
    switch (this.dir) {
        case LEFT:
            super.addToDx(-5);
            weapon.setAimX(dir);
            break;
        case RIGHT:
            super.addToDx(5);
            weapon.setAimX(dir);
            break;
    }
    weapon.setAimY(aim);
    weapon.update(frameDelta, spawnedObj);
    if (this.jetpackState == true) {
        super.addToDy(-5);
    }
    if (didFire && gameStatsObservable != null) {
        this.gameStatsObservable.checkUIInfo();
        didFire = false;
    }
    super.update(frameDelta, spawnedObj);
    return true;
}

/**
 * fires weapon
 */
public void fireWeapon() {
    this.didFire = this.weapon.fire();
    /*System.out.println("player: " + getName()
     + " Ammo: " + weapon.getAmmo()
     + " cooldown: " + weapon.getCooldown()
     );*/
}

/**
 * if jetpack should be on or off
 * @param b on/off
 */
public void setJetpackState(boolean b) {
    this.jetpackState = b;
}

/**
 * what direction the player moves
 * @param direction
 */
public void setDirection(Direction direction) {
    this.dir = direction;
}

/**
 * if he aims up or down
 * @param direction aim
 */
public void setAim(Direction direction) {
    this.aim = direction;
}

/**
 * The player has been in a collision, with gameObj
 * @param gameObj gameObj
 */
@Override
public void collisionWith(Physics gameObj) {
    if (gameObj instanceof SpawnBox) {
        SpawnBox box = (SpawnBox) gameObj;
        this.weapon = box.consumeBox(this);

        if (this.aiControl != null) {
            this.aiControl.pickedupSpawnBox();
        }
    }
}

/**
 * players name
 * @return name in string
 */
public String getName() {
    return this.name;
}
/**
 * The point it hit the terrain at
 * @param x point it hit the terrain at
 * @param y point it hit the terrain at
 */
public void collisionWithTerrainAt(double x, double y) {
    // stop movement
    this.addToDx(-this.getDx());
    this.addToDy(-this.getDy());
    this.addToDx((this.getX() - x) / 1);
    this.addToDy((this.getY() - y) / 1);
    if (this.aiControl != null) {
        this.aiControl.collisionWithTerrainAt(x, y);
    }
}

/**
 * a class used for storage of player info from the lobby,
 */
```

```
    public static class Playerinfo {

        private String playername;
        private KeyboardController keyInputs;

        /**
         *
         * @param name player name
         * @param playerinput input controller for this player
         */
        public Playerinfo(String name, KeyboardController playerinput) {
            this.playername = name;
            this.keyInputs = playerinput;
        }

        /**
         *
         * @param name_ name
         */
        public void setPlayerName(String name_) {
            this.playername = name_;
        }

        /**
         *
         * @return name of player
         */
        public String getPlayerName() {

            return this.playername;
        }

        /**
         *
         * @return this players KeyboardController
         */
        public KeyboardController getKeyboard() {
            return this.keyInputs;
        }

        /**
         *
         * @param playerinput_ set new KeyboardController for player
         */
        public void setKeyboard(KeyboardController playerinput_) {
            this.keyInputs = playerinput_;
        }

    }
}

/*
```

```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import java.util.ArrayList;

/**
 *Class containing information about the projectiles used by weapons in the game
 * @author o_0
 */
public class Projectile extends Physics {

    private double damage;
    private double speed;
    private double damageRadius;
    private Player owner;
    private Player target;
    private double timeToLive = 2.0;

    private ProjectileType type;

    /**
     * Private constructor, using the builder pattern
     * @param aimSide what start dx
     * @param aimUp what start dy
     * @param builder a builder for a projectile
     */
    private Projectile(double aimSide, double aimUp, ProjectileBuilder builder) {
        super(0, 0, 10,builder.modelId);
        this.damage = builder.damage;
        this.damageRadius = builder.radius;
        this.speed = builder.speed;
        this.owner = builder.owner;
        this.target = builder.target;
        this.type = builder.type;
        this.setX(this.owner.getX());
        this.setY(this.owner.getY());
        super.addToDx(aimSide * speed);
        super.addToDy(aimUp * speed);
        if (type == ProjectileType.BULLET) {
            super.setGravity(false);
        }
    }

    /**
     * getter
     * @return damage
     */
    public double getDamage() {
        return this.damage;
    }

    /**
     *
     * @return damageRaduys
     */
    public double getDamageRadius() {
        return this.damageRadius;
    }

    /**
     * what player shot this
     * @return
     */
    public Player getOwner() {
        return this.owner;
    }
/**
     * This is called every frame by gameTimer thru the controller
     * @param frameDelta fraction of a seconde since last update
     * @param spawnedObj list to store all newly created objects
     * @return not used
     */
    @Override
    public boolean update(double frameDelta, ArrayList<GameObject> spawnedObj) {
        super.update(frameDelta, spawnedObj);
        timeToLive -= frameDelta;
        if (timeToLive < 0) {
            spawnedObj.add(new Explosion(getX(), getY(), getDamageRadius()));
            super.deactivate();
        }
        return true;
    }

    /**
     * The player has been in a collision, with gameObj
     * @param gameObj gameObj
     */
    @Override
    public void collisionWith(Physics gameObj) {
        if (this.owner != gameObj) {
            timeToLive = -1;
        }
    }
  /**
     * The point it hit the terrain at
     * @param x point it hit the terrain at
     * @param y point it hit the terrain at
     */
    public void collisionWithTerrainAt(double x, double y) {
        timeToLive = -1;    // explodes when < 0
    }
```

```java
/**
 * builder for projectile
 */
public static class ProjectileBuilder {

    private double damage = 10;
    private double speed = 20;
    private ProjectileType type;
    private double radius = 50;
    private Player owner = null;
    private Player target = null;

    private int modelId = 0;

    /**
     *
     * @param type what projectile
     */
    public ProjectileBuilder(ProjectileType type) {
        this.type = type;
    }

    /**
     *
     * @param modelId modelid
     * @return builder
     */
    public ProjectileBuilder withModel(int modelId) {
        this.modelId = modelId;
        return this;
    }

    /**
     *
     * @param damage
     * @return
     */
    public ProjectileBuilder withDamage(double damage) {
        this.damage = damage;
        return this;
    }

    /**
     *
     * @param radius
     * @return builder
     */
    public ProjectileBuilder withRadius(double radius) {
        this.radius = radius;
        return this;
    }
```

```
        /**
         *
         * @param speed
         * @return builder
         */
        public ProjectileBuilder withSpeed(double speed) {
            this.speed = speed;
            return this;
        }

        /**
         *
         * @param owner
         * @return builder
         */
        public ProjectileBuilder withOwner(Player owner) {
            this.owner = owner;
            return this;
        }

        /**
         *
         * @param target
         * @return builder
         */
        public ProjectileBuilder withTarget(Player target) {
            this.target = target;
            return this;
        }

        /**
         *
         * @param aimX
         * @param aimY
         * @return a new projectile for use by game
         */
        public Projectile build(double aimX, double aimY) {
            double aimSide = aimX / Math.sqrt(aimX * aimX + aimY * aimY);
            double aimUp = aimY / Math.sqrt(aimX * aimX + aimY * aimY);
            return new Projectile(aimSide, aimUp, this);
        }
    }
}



/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package GameObjects;

/**
 * different ProjectileType
 * @author o_0
 */
public enum ProjectileType {GRANADE, BULLET, MISSILE};


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import GameObjects.Projectile.ProjectileBuilder;
import java.util.Random;

/**
 *Boxes containing weapons that are spawned during the game
 * @author o_0
 */
public class SpawnBox extends Physics {
    private Weapon weapon;
    private ProjectileBuilder projectile;
    private int ammo;
    private double cooldown;
    /**
     * Constructor for spawnbox
     * @param type what type of projectile the box will contain,
     * the stats of the projectile is randomised during creation
     * @param x the x value the box will have when spawned
     * @param y the y value the box will have when spawned
     */
    public SpawnBox(ProjectileType type,double x, double y) {
        super(0, 0,10, 0);
        this.setX(x);
        this.setY(y);
        Random rand = new Random();
        this.projectile = new ProjectileBuilder(type);
        this.projectile = this.projectile.withDamage(1 + rand.nextDouble() * 50)
                .withSpeed(20 + rand.nextDouble() * 500)
                .withRadius(5 + rand.nextDouble() * 200);
        switch(type) {
            case GRANADE: projectile.withModel(3); break;
            case BULLET: projectile.withModel(4); break;
            case MISSILE: projectile.withModel(1); break;
            default: projectile.withModel(0); break;
        }
        this.ammo = rand.nextInt(100) + 10;
        //System.out.println("Create Spawnbox with ammo: " + ammo);
        this.cooldown = 0.01 + rand.nextDouble() * 2 ;
        //this.setGravity(false);
    }

    /**
     * when a box is consumed by a player
     * @param player the player that took the box
     * @return the randomised weapon the got
     */
    public Weapon consumeBox(Player player) {
        this.deactivate();
        //System.out.println("player: " + player.getName() + " created weapon: ammo" + ammo);
        return new Weapon(player,projectile,cooldown,ammo,0);
    }

    /**
     * Checks if the box collides with the terrain
     * @param x box x value
     * @param y box y value
     */
    public void collisionWithTerrainAt(double x, double y) {
        // stop movement
        this.addToDx(-this.getDx());
        this.addToDy(-this.getDy());

        // move the objected so it do not collide
        double diffX = (getX() - x);
        double diffY = (getY() - y);
        this.setX(getX() + 1*Math.signum(diffX));
        this.setY(getY() + 1*Math.signum(diffY));
        if(getBodyRadius()!=0) {
            diffX = diffX - diffX*(1 - diffX/getBodyRadius());
            diffY = diffY- diffY*(1 - diffY/getBodyRadius());
        }
        this.addToDx(diffX);
        this.addToDy(diffY);
    }

    @Override
    /**
     * spawnbox collided with the object taken as parameter.
     * The box will however do noting in this case.
     */
    public void collisionWith(Physics gameObj) {

    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameObjects;

import GameObjects.Projectile.ProjectileBuilder;

import java.util.ArrayList;

/**
 *Weapon class containing information needed for weapon functionality
 * @author o_0
 */
public class Weapon extends GameObject {
    Player owner;
    private Direction dirX = Direction.LEFT;
    private Direction dirY = Direction.NONE;
    private double angle = 0;
    private double cooldown = 0.1;
    private double fireTimer = 0;
    private boolean didFire = false;
    private ProjectileBuilder projectileBuilder;
    private int ammo = 0;

    /**
     *
     * @param owner who has this weapon
     * @param projectile hhe ProjectileBuilder
     * @param cooldown what cooldown this weapon should have
     * @param ammo how much ammo it has
     * @param modelId the modelid, no used
     */
    public Weapon(Player owner, ProjectileBuilder projectile,double cooldown, int ammo, int
modelId) {
        super(modelId);
        this.owner = owner;
        this.setX(50);
        this.setY(50);
        this.ammo = ammo;
        this.cooldown = cooldown;
        this.fireTimer = 0;
        this.projectileBuilder = projectile.withOwner(owner);

    }

    /**
     * Default value, basic weapon
     * @param owner
     * @param type
     * @param modelId
     */
    public Weapon(Player owner, ProjectileType type,int modelId) {
        super(modelId);
        this.setX(50);
        this.setY(50);
        this.ammo = 50;
        this.owner = owner;
        this.projectileBuilder = new ProjectileBuilder(type)
                .withModel(1)
                .withDamage(20)
                .withSpeed(200)
                .withRadius(50)
                .withOwner(owner);
    }

    /**
     * sets aim
     * @param dir
     */
    public void setAimX(Direction dir) {
        this.dirX = dir;
    }

    /**
     *
     * @param dir
     */
    public void setAimY(Direction dir) {
        this.dirY = dir;
    }

    /**
     * updates cooldown
     * @param frameDelta
     */
    private void updateCooldown(double frameDelta) {
        if(fireTimer > 0) {
            fireTimer -= frameDelta;
        }
    }

    /**
     * fires a weapon
     * @return if it sucseeded
     */
    public boolean fire() {
        if(fireTimer > 0) {
            //System.out.println("player: " + owner.getName() + " Ammo: " + ammo);
            return false;
        }
        fireTimer = cooldown;
        didFire = true;
        return true;
    }

    /**
```

```
 *
 * @return ammo left
 */
public int getAmmo() {
    return this.ammo;
}

/**
 * cooldown left
 * @return
 */
public double getCooldown() {
    return this.cooldown;
}

/**
 *
 * @return the current aim angle
 */
protected double getAimAngle() {
    return this.angle;
}

/**
 * This is called every frame by gameTimer thru the controller
 * @param frameDelta fraction of a seconde since last update
 * @param spawnedObj list to store all newly created objects
 * @return not used
 */
@Override
public boolean update(double frameDelta, ArrayList<GameObject> spawnedObj) {

    switch(dirY) {
        case UP: angle += -1.0*frameDelta; break;
        case DOWN: angle += 1.0*frameDelta; break;
    }
    double halfPi = Math.PI/2;
    angle = (angle > halfPi) ? halfPi : angle;
    angle = (angle < -halfPi) ? -halfPi : angle;

    updateCooldown(frameDelta);

    if(didFire && ammo > 0) {
        double tmpAngle = angle;
        if(dirX == Direction.LEFT && angle < halfPi) {
            tmpAngle = -tmpAngle;
            tmpAngle += Math.PI;
        }
        if(dirX == Direction.RIGHT && angle > halfPi) {
            tmpAngle -= Math.PI;
        }
        double aimX = Math.cos(tmpAngle);
```

```
            double aimY = Math.sin(tmpAngle);
            System.out.println("owner: " + owner.getName() +" total Angle: " + tmpAngle +" sin angle: 
" + angle);
            Projectile projectile = projectileBuilder.build(aimX, aimY);
            spawnedObj.add(projectile);
            didFire = false;
            ammo--;
        }
        return true;
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameTimers;

import Controller.GameController;
import GameObjects.GameObject;
import java.util.ArrayList;
import javafx.animation.AnimationTimer;

/**
 *Gametimer keeps track of time during the game and calls
 * functions reliant on timed actions.
 * @author o_0
 */
public class GameTimer extends AnimationTimer {

    public static final double BILLION = 1000_000_000.0; //from Ball lab2b
    private long lastTime;
    GameController gameController;
    private double gametime = 0;
    private double savetime = 0;
    private double spawnTime = 0;

    /**
     * Constructor
     * @param gameController takes a gameController as input
     * since the methods are needed
     */
    public GameTimer(GameController gameController) {
        super();
        this.gameController = gameController;
    }

    @Override
    /**
     * handle keeps track of time and calls methods from gameController.
     * handle removes and adds objects, spawn players.
     */
    public void handle(long now) {

        double frameDelta = (now - lastTime) / BILLION;
        frameDelta = (frameDelta < 1) ? frameDelta : 0;
        lastTime = now;
        this.gametime += frameDelta;
        spawnTime += frameDelta;
        ArrayList<GameObject> inActiveObj;
        inActiveObj = gameController.removeInactiveObjects();
        gameController.removeObservers(inActiveObj);

        gameController.gameCollisionUpdate();

        ArrayList<GameObject> newObjects;
        newObjects = gameController.updateGame(frameDelta);
        if(spawnTime > 10) {
            newObjects.add(gameController.makeSpawnBox());
            spawnTime = 0;
        }
        if (gameController.deathcheck()) {
            savetime += frameDelta;
            if (savetime > 2) {
                gameController.playerRespawn();
                savetime = 0;
            }
        }
        gameController.addObjects(newObjects);
        gameController.addObservers(newObjects);

    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package GameTimers;

import Controller.RenderController;
import GameData.Terrain;
import GameObjects.GameObject;
import static GameTimers.GameTimer.BILLION;
import java.util.ArrayList;
import javafx.animation.AnimationTimer;

/**
 *
 * @author o_0
 */
public class RenderTimer extends AnimationTimer {

    private ArrayList<GameObject> gameObjects;
    RenderController renderController;
    private Terrain terrain;
    long lastTime = 0;
    double gameTime = 0;
    /**
     *
     * @param renderController the controller for render
     * @param gameObj all gameObjects
     * @param terrain the game terrain
     */
    public RenderTimer(RenderController renderController, ArrayList<GameObject> gameObj,
Terrain terrain) {
        super();
        this.gameObjects = gameObj;
        this.terrain = terrain;
        this.renderController = renderController;
    }

    /**
     * updates all game movements
     * @param now
     */
    @Override
    public void handle(long now) {
        renderController.displayTerrain(terrain);
        double frameDelta = (now - lastTime) / BILLION;
        frameDelta = (frameDelta < 1) ? frameDelta : 0;
        lastTime = now;
        gameTime += frameDelta;
// due to GameUpdate controller ability to add/remmove from gameObjects, we need a clone
        ArrayList<GameObject> localbuffer = (ArrayList<GameObject>) gameObjects.clone();
        for (GameObject obj : localbuffer) {
            RenderingState state = new RenderingState(obj,gameTime);
            renderController.displayModel(obj,state);
        }
    }

    /**
     * used by the controller to determin animation and scaling
     */
    public class RenderingState {
        private double timePassed;
        private double radius;
        /**
         *
         * @param obj gameObject
         * @param time time it was done
         */
        public RenderingState(GameObject obj,double time) {
            this.timePassed = time;
            this.radius = obj.getBodySize();
            radius = (radius >= 0) ? radius : 0;
        }

        /**
         *
         * @return
         */
        public double getTimeCount() {
            return this.timePassed;
        }
        /**
         *
         * @param width
         * @param heigth
         * @return the scaling factor
         */
        public double getScaleFactor(double width,double heigth) {
            double maxValue = Math.max(width, width);
            if(this.radius <= 0 || maxValue <= 0) {
                return 1;
            }
            return this.radius/maxValue;
        }
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package UIGraphics;

import Controller.KeyboardController;
import GameObjects.Player.Playerinfo;
import java.util.ArrayList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import lab5game.GameSetup;

/**
 *The Lobby class creates a number of buttons and textfields made to collect
 * inputs from users. It is also where a user can choose to play by themselfe
 * or against a ai opponent.
 * the buttons are used as keybindings and the textfield allowes user to
 * enter their name.
 * @author mats
 */
public class Lobby {

    private int numOfAi;
    private ArrayList<Playerinfo> playerInfo;
    private Stage Lobbystage;
    private Group root;
    private CheckBox aiCheck;
    private GameSetup gameSetup;
    private String player1name;
    private String player2name;

    private boolean enterNewKey;
    private int player;
    private int whatkey;

    /**
    * constructor for the Lobby class
    * @param stage takes a stage used to draw on
    * @param gameSetup_ takes gameSetup to gain accses to methods to start game
    * Also creates the standard keyboard controllers in case the users chooses not
    * to change them.
    *
    */
    public Lobby(Stage stage, GameSetup gameSetup_) {
        KeyboardController input = new KeyboardController(KeyCode.A, KeyCode.D,
                KeyCode.W,KeyCode.S,
                KeyCode.SHIFT, KeyCode.Q);

        playerInfo = new ArrayList<Playerinfo>();
        playerInfo.add(new Playerinfo("bob", input));
        input = new KeyboardController(KeyCode.LEFT,
KeyCode.RIGHT,KeyCode.UP,KeyCode.DOWN,
                KeyCode.PERIOD, KeyCode.SPACE);
        playerInfo.add(new Playerinfo("tod", input));
        numOfAi = 0;
        this.Lobbystage = stage;
        this.gameSetup = gameSetup_;
    }
```

```
/**
 * lobbysetup method does a lot of things, firstly it creates a number of
 * buttons used to change keyboardcontrollers for the users.
 * lobbysetup also creates the checkbox where a user can choose to face a
 * ai opponen and the textfields used to get user names.
 * lastly the lobbysetup puts all these things on the screen and uses
 * the startbutton it created to start the game.
 */
public void lobbysetup() {
    Stage stage = this.Lobbystage;
    enterNewKey = false;
    player = 0;
    whatkey = 0;

    EventHandler handler;
    handler = new EventHandler<KeyEvent>() {
        @Override
        public void handle(KeyEvent event) {
            if (enterNewKey) {
                setKey(event.getCode());
                enterNewKey = false;
                event.consume();
            } else {
                event.consume();
            }
        }
    };

    Label jumpp1 = new Label("Jump");
    Button jumpbuttonp1 = new Button();
    jumpbuttonp1.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            player = 1;
            enterNewKey = true;
            whatkey = 1;
        }
    });
    HBox jumpcon = new HBox(10);

    Label leftp1 = new Label("Left");
    Button leftbuttonp1 = new Button();
    leftbuttonp1.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            player = 1;
            enterNewKey = true;
            whatkey = 2;
        }
    });
    HBox leftcon = new HBox(10);

    Label rightp1 = new Label("Right");
    Button rightbuttonp1 = new Button();
```

```
rightbuttonp1.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        player = 1;
        enterNewKey = true;
        whatkey = 3;
    }
});
HBox rightcon = new HBox(10);

Label jumpp2 = new Label("Jump");
Button jumpbuttonp2 = new Button();
jumpbuttonp2.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        player = 2;
        enterNewKey = true;
        whatkey = 1;
    }
});
HBox jumpcon2 = new HBox(10);

Label leftp2 = new Label("Left");
Button leftbuttonp2 = new Button();
leftbuttonp2.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        player = 2;
        enterNewKey = true;
        whatkey = 2;
    }
});
HBox leftcon2 = new HBox(10);

Label rightp2 = new Label("Right");
Button rightbuttonp2 = new Button();
rightbuttonp2.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        player = 2;
        enterNewKey = true;
        whatkey = 3;
    }
});
HBox rightcon2 = new HBox(10);

Label player1 = new Label("Player 1 Name:");
TextField player1Name = new TextField();
HBox namecon = new HBox(10);

Label player2 = new Label("Player 2 Name:");
TextField player2Name = new TextField();
HBox namecon2 = new HBox(10);

aiCheck = new CheckBox();
Button start = new Button("Start");
```

```
start.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        player1name = player1Name.getText();
        playerInfo.get(0).setPlayerName(player1name);
        player2name = player2Name.getText();
        playerInfo.get(1).setPlayerName(player2name);
        if (aiCheck.isSelected() == true) {
            playerInfo.remove(1);
            numOfAi++;
        }
        stage.removeEventHandler(KeyEvent.ANY, handler);
        startMap();
    }
});

root = new Group();
Scene scene = new Scene(root, 1024, 720, Color.AZURE);

Canvas canvas = new Canvas(scene.getWidth(), scene.getHeight());

start.setLayoutX(canvas.getWidth() / 2);
start.setLayoutY(canvas.getHeight() / 6 * 5);

aiCheck.setText("Ai");

namecon.getChildren().addAll(player1, player1Name);

namecon2.getChildren().addAll(player2, player2Name, aiCheck);

jumpcon.getChildren().addAll(jumpp1, jumpbuttonp1);

leftcon.getChildren().addAll(leftp1, leftbuttonp1);

rightcon.getChildren().addAll(rightp1, rightbuttonp1);

jumpcon2.getChildren().addAll(jumpp2, jumpbuttonp2);

leftcon2.getChildren().addAll(leftp2, leftbuttonp2);

rightcon2.getChildren().addAll(rightp2, rightbuttonp2);

Label wichplayer = new Label("Player1:");
VBox keyconp1 = new VBox(10);
keyconp1.getChildren().addAll(wichplayer, jumpcon, leftcon, rightcon);

Label wichplayer2 = new Label("Player2:");
VBox keyconp2 = new VBox(10);
keyconp2.getChildren().addAll(wichplayer2, jumpcon2, leftcon2, rightcon2);

HBox players12 = new HBox(10);
```

```
    players12.getChildren().addAll(keyconp1, keyconp2);

    VBox keyconinfo = new VBox(10);
    Label info = new Label("Choose buttons to controll your caracter with:");
    keyconinfo.getChildren().addAll(info, players12);

    /**
     * HBox namecon4 = new HBox(); namecon4.setSpacing(10);
     * namecon4.getChildren().addAll(namecon, namecon2, keyconinfo);
    namecon4.setPrefWidth(scene.getWidth());
     */
    FlowPane namecon4 = new FlowPane();
    namecon4.setVgap(10);
    namecon4.setHgap(10);
    namecon4.getChildren().addAll(namecon, namecon2, keyconinfo);
    namecon4.setPrefWidth(scene.getWidth());

    stage.addEventHandler(KeyEvent.KEY_PRESSED, handler);

    /**
     * stage.addEventHandler(KeyEvent.KEY_PRESSED, new
     * EventHandler<KeyEvent>() {
     *
     * @Override public void handle(KeyEvent event) { if(enterNewKey) {
     * setKey(event.getCode()); enterNewKey = false; } }
    });
     */
    root.getChildren().add(canvas);
    root.getChildren().add(namecon4);
    root.getChildren().add(start);

    stage.setTitle("Lobby");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.sizeToScene();
    stage.show();
}

/**
 * Simply a method that calls the startMap method in gameSetup so that
 * the game starts
 */
public void startMap() {

    gameSetup.startMap();
}

/**
 * A getter method used to see if the users choose to play against ai
 * opponents
 * @return int containing number of ai opponents
 */
```

```
    public int getnumOfAi() {

        return numOfAi;
    }

    /**
     * A getter method that returns information about the player in a arraylist
     * @return Arraylist of Playerinfo typ contains name and keybordcontroller
     */
    public ArrayList<Playerinfo> getPlayerInfo() {

        return playerInfo;
    }

    /**
     * Method that sets the keyboard buttons the users want to use to controll
     * their players
     * @param key is a KeyCode used to determine which key users pressed
     */
    private void setKey(KeyCode key) {
        int playerIdx = player - 1;
        if (playerIdx < 0 || playerIdx >= playerInfo.size()) {
            return;
        }
        KeyboardController input = playerInfo.get(playerIdx).getKeyboard();
        switch (whatkey) {
            case 1:
                input.setKeyJetpackOn(key);
                break;
            case 2:
                input.setKeyMoveLeft(key);
                break;
            case 3:
                input.setKeyMoveRight(key);
                break;
            default:
                break;
        }

    }

}


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package UIGraphics;

import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;
import lab5game.GameSetup;

/**
 *
 * @author mats
 * Creates the first screen the application shows, used only as a startscreen where
 * the user can choose to play or close the application.
 */
public class LoginScreen {
    GameSetup gameSetup;
    private Group root;
    private Image image;
    private Image imagebut;
    private Image imagebut2;
    private Button but;
    private Button but2;
    private Stage loginStage;

    /**
     * constructor for the LoginScreen class
     * @param stage Gets a Stage used for canvas and Boxes
     * @param gameSetup takes a gameSetup as parameter to gett accses to methods
     * Also loads the background and button images
     */
    public LoginScreen(Stage stage, GameSetup gameSetup) {
        this.gameSetup = gameSetup;
        this.loginStage = stage;
        this.image = new Image("Resource/battleTerrain.png",false);
        this.imagebut = new Image("Resource/Playknapp.png",false);
        //this.imagebut2 = new Image("Resource/higknapp.png",false);

    }
    /**
     * Setup method that creates the buttons and stages, also
     * creates the canvas
     */
    public void setup(){
        Stage stage = this.loginStage;
            but = new Button();
            but.setGraphic(new ImageView(imagebut));

            but.setOnAction(new EventHandler<ActionEvent>() {
```

```
            @Override
            public void handle(ActionEvent event) {
                play();
            }
        });

        /**but2 = new Button();
            but2.setGraphic(new ImageView(imagebut2));

            but2.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                loadgame();
            }
        });*/

            root = new Group();
            Scene scene = new Scene(root, 1024, 720, Color.GREEN);
            Canvas canvas = new Canvas(scene.getWidth(), scene.getHeight());

            but.setLayoutX(canvas.getWidth()/2 - (imagebut.getWidth()/2));
        but.setLayoutY(canvas.getHeight()/2 - (imagebut.getHeight()/2));

        /**but2.setLayoutX(canvas.getWidth()/2 - (imagebut.getWidth()/2));
        but2.setLayoutY(canvas.getHeight()/2 - (imagebut.getHeight()/2));*/

            GraphicsContext gc = canvas.getGraphicsContext2D();
            gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
            gc.drawImage(image, 0, 0, canvas.getWidth(), canvas.getHeight());

            root.getChildren().add(canvas);
            root.getChildren().add(but);
        //root.getChildren().add(but2);


            stage.setTitle("Start");
            stage.setScene(scene);
            stage.setResizable(false);
            stage.sizeToScene();
            stage.show();

    }

    /**
     * Calls method to load next window in gameSetup
     */
    public void play(){
        gameSetup.lobbyStart(this.gameSetup);
    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package UIGraphics;

import GameData.Terrain;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.Menu;
import lab5game.BattleArena;
import lab5game.GameSetup;

/**
 *Class creating the menu at the top of the scene during the game
 * @author mats
 */
public class TopMenu {

    private BattleArena battlearena;
    private MenuBar menubar;
    private GameSetup GS;

    /**
     * constructor for TopMenu
     * @param ba takes a BattleArena as input to get accsess to methods
     * @param GS_ takes gameSetup to get accsess to methods
     */
    public TopMenu(BattleArena ba, GameSetup GS_){
        this.battlearena = ba;
        this.GS = GS_;
    }

    /**
     * Method that creates the menubar and all it's options and returns it
     * @return returns a MenuBar to be put on the game scene
     */
    public MenuBar getMenu(){

        menubar = new MenuBar();

        Menu menuState = new Menu("State");

            MenuItem stopgame = new MenuItem("Stop");
            stopgame.setOnAction(stop());
            MenuItem startgame = new MenuItem("start");
        startgame.setOnAction(start());
    menuState.getItems().addAll(stopgame, startgame);

    Menu menuOptions = new Menu("Options");

    MenuItem savegame = new MenuItem("Save");
       savegame.setOnAction(save());
    MenuItem loadgame = new MenuItem("Load");
       loadgame.setOnAction(load());
    MenuItem infogame = new MenuItem("Information");
       infogame.setOnAction(info());

       //MenuItem scoregame = new MenuItem("Score board");
    //startgame.setOnAction(score());

    menuOptions.getItems().addAll(savegame, loadgame,  infogame);

    Menu menuEnd = new Menu("End game");

    MenuItem maingame = new MenuItem("Main menu");
    maingame.setOnAction(reset());
    MenuItem exitgame = new MenuItem("Exit Game");
    exitgame.setOnAction(quit());

    menuEnd.getItems().addAll(maingame, exitgame);


    menubar.getMenus().addAll(menuState, menuOptions, menuEnd);


    return menubar;
}

/**
 * Action event containing a call to a method
 * @return a Eventhandler of actionevent type
 */
private EventHandler<ActionEvent> start(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent y) {
                battlearena.play();
        }
    };
}

/**
 * Action event containing a call to a method
 * @return Eventhandler of actionevent type
 */
public EventHandler<ActionEvent> stop(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent t) {
                battlearena.paus();
                }
            };

}

/**
 * Action event containing a call to a method
 * @return Eventhandler of actionevent type
 */
private EventHandler<ActionEvent> save(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent y) {
            battlearena.saveGame();
        }
    };
}

/**
 * Action event containing a call to a method
 * @return Eventhandler of actionevent type
 */
private EventHandler<ActionEvent> load(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent y) {
            battlearena.loadGame();
        }
    };
}

/**
 * Action event containing a call to a method
 * @return Eventhandler of actionevent type
 */
public EventHandler<ActionEvent> reset(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent z) {
                GS.resetGame();
                }
            };
}

/**
 * Action event containing a call to a method
 * @return Eventhandler of actionevent type
 */
public EventHandler<ActionEvent> quit(){
    return new EventHandler<ActionEvent>() {
        public void handle(ActionEvent z) {
                GS.exitGame();
                }
            };
}
```

```
    public void showscoreboard(){}

    /**
     * Action event containing a call to a method
     * @return Eventhandler of actionevent type
     */
    public EventHandler<ActionEvent> info(){
        return new EventHandler<ActionEvent>() {
            public void handle(ActionEvent z) {
                        battlearena.info();
                        }
                };
    }


}


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package UIGraphics;

import java.util.Observable;
import java.util.Observer;
import GameObjects.Player;
import java.util.ArrayList;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import java.text.DecimalFormat;

/**
 *Observer that looks on the basic stats of a player like health and ammo.
 * If changes happen to the stats this class will update the numbers
 * shown on the screen in the game.
 * @author mats
 */
public class UIStatObserver implements Observer {

    //private StackPane root;
    private Label player1name;
    private Label player2name;
    private Label player1health;
    private Label player2health;
    private Label player1ammo;
    private Label player2ammo;
    private Label player1death;
    private Label player2death;
    private ArrayList<Player> players;

    /**
     * constructor for the observer
     * @param players takes a arraylist of Player type containing players
     */
    public UIStatObserver(ArrayList<Player> players) {
        super();
        this.players = players;
        //createUI(root);

    }

    /**
     * Method that creates the graphic view for the stats in a HBox that is
     * returned
     * @param root takes a StackPane  a parameter since this is were the
     * information will be added
     * @return returns HBox containing the information
     */
    public HBox createUI(StackPane root) {
        player1name = new Label(players.get(0).getName());
        player2name = new Label(players.get(1).getName());
        player1name.setTextFill(Color.RED);
        player2name.setTextFill(Color.RED);
        player1health = new Label("Health:" + players.get(0).currentHealth());
        player2health = new Label("Health:" + players.get(1).currentHealth());
        player1health.setTextFill(Color.RED);
        player2health.setTextFill(Color.RED);
        player1ammo = new Label("Ammo:" + players.get(0).currentAmmo());
        player2ammo = new Label("Ammo:" + players.get(1).currentAmmo());
        player1ammo.setTextFill(Color.RED);
        player2ammo.setTextFill(Color.RED);
        player1death = new Label("Deaths:" + players.get(0).getDeaths());
        player2death = new Label("Deaths:" + players.get(1).getDeaths());
        player1death.setTextFill(Color.RED);
        player2death.setTextFill(Color.RED);
        HBox playerinfocon1 = new HBox(5);
        playerinfocon1.getChildren().addAll(player1name, player1health, player1ammo,
player1death);
        HBox playerinfocon2 = new HBox(5);
        playerinfocon2.getChildren().addAll(player2name, player2health, player2ammo,
player2death);
        HBox namecon = new HBox(500);
        namecon.setAlignment(Pos.BOTTOM_CENTER);
        namecon.getChildren().addAll(playerinfocon1, playerinfocon2);
        //root.getChildren().addAll(playerinfocon1, playerinfocon2);
        return namecon;
    }
    /**
     * sets the players of whom the stats are shown
     * @param players_ ArrayList of Player type
     */
    public void setPlayers(ArrayList<Player> players_) {
        this.players = players_;

    }

    @Override
    /**
     * Method that updates the labels containing player information
     */
    public void update(Observable o, Object arg) {
        DecimalFormat form = new DecimalFormat("##0.0");
        String p1h = form.format(players.get(0).currentHealth());
        String p2h = form.format(players.get(1).currentHealth());

        this.player1health.setText("Health:" + p1h);
        this.player2health.setText("Health:" + p2h);
        //this.player1health.setText("Health:" + players.get(0).currentHealth());
        //this.player2health.setText("Health:" + players.get(1).currentHealth());
        this.player1ammo.setText("Ammo:" + players.get(0).currentAmmo());
        this.player2ammo.setText("Ammo:" + players.get(1).currentAmmo());
        this.player1death.setText("Deaths:" + players.get(0).getDeaths());
        this.player2death.setText("Deaths:" + players.get(1).getDeaths());
    }

}


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package View;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;

/**
 *
 * @author mats
 */
public class GameView {

    private Canvas canvas;
    //private Canvas background;
    public GameView(Canvas canvas) {
        this.canvas = canvas;
    }

    /**
     * renders backgournd
     * @param background
     */
    public void drawbackground(Image background){
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.drawImage(background, 0, 0, canvas.getWidth(), canvas.getHeight());
    }

    /**
     *
     * @param model model image to be renders
     * @param x coord
     * @param y coord
     * @param scalingFactor scaling to be done
     */
    public void drawmodel(Image model, double x, double y, double scalingFactor){
        GraphicsContext gc = canvas.getGraphicsContext2D();
        double width = model.getWidth() * scalingFactor;
        double height = model.getHeight() * scalingFactor;
        gc.drawImage(model, x - width/2, y - height/2, width, height);
    }

   /**
    * the terrain to be drawn
    * @param terrain
    */
    public void drawterrain(Image terrain){
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.drawImage(terrain, 0, 0, canvas.getWidth(), canvas.getHeight());
    }

}


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package lab5game;

import Collisions.Collisions;
import Controller.ExplosionObserver;
import Controller.GameController;
import Controller.GameStatsObservable;
import GameData.GameModel;
import GameTimers.RenderTimer;
import GameTimers.GameTimer;
import Controller.KeyboardController;
import Controller.RenderController;
import GameData.Ai;
import GameData.GraphicModels;
import GameData.Terrain;
import GameObjects.GameObject;
import GameObjects.Physics;
import GameObjects.Player;
import UIGraphics.TopMenu;
import UIGraphics.UIStatObserver;
import View.GameView;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.MenuBar;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

/**
 *Class responsible for the construction of the game.
 * contains everything from players and terrain to alerts.
 * @author o_0
 */
public class BattleArena {

    private ArrayList<GameObject> gameObjects;
    private Terrain terrain;
    //private Group root;
    private StackPane root;

    private RenderTimer renderTimer;
    private GameTimer gameUpdate;

    private ArrayList<EventHandler<KeyEvent>> inputs;
    private Stage gameStage;
    //private UIStatObserver uIStatObserver;
    private GameStatsObservable gameStatsObservable;

    /**
     * Constructor takes a stage as input
     * @param stage stage taken from start of program GameSetup
     */
    public BattleArena(Stage stage) {
        this.gameStage = stage;
    }



    /**
     * Creates players, adds all infromation regarding name and keybord
     * to players
     * @param playerInfo ArrayList of type Player internal class playerinfo
     * @return returns ArrayList of Player type
     */
    private ArrayList<Player> createPlayers(ArrayList<Player.Playerinfo> playerInfo) {
        inputs = new ArrayList<EventHandler<KeyEvent>>();
        ArrayList<Player> newPlayers = new ArrayList<Player>();
        EventHandler<KeyEvent> pressed;
        EventHandler<KeyEvent> released;
        KeyboardController keyboard;
        for (Player.Playerinfo info : playerInfo) {
            Player player = new Player(info.getPlayerName(), 500, 20, 0);
            newPlayers.add(player);
            System.out.println(info.getPlayerName());
            keyboard = info.getKeyboard();
            pressed = keyboard.getPlayerKeyPressedHandler(player);
            released = keyboard.getPlayerKeyReleasedHandler(player);
            gameStage.addEventHandler(KeyEvent.KEY_PRESSED, pressed);
            gameStage.addEventHandler(KeyEvent.KEY_RELEASED, released);
            inputs.add(pressed);
            inputs.add(released);

        }
        if(!newPlayers.isEmpty()) {
            gameObjects.addAll(newPlayers);
        }
        return newPlayers;
    }

    /**
     * removes eventhandler from gameStage
     */
    private void removeKeyEvents() {
        for (EventHandler<KeyEvent> input : inputs) {
            gameStage.removeEventHandler(KeyEvent.ANY, input);
        }
        inputs.clear();
    }

    /**
     * Creates new explosion observer for objects that can detonate
     * @return a explosionObserver
     */
    public ExplosionObserver observerSetup() {
        ExplosionObserver explosionObserver = new ExplosionObserver();
        explosionObserver.addObserver(this.terrain);
        for (GameObject obj : gameObjects) {
            if (obj.physicsEnable()) {
                explosionObserver.addObserver((Physics) obj);
            }
        }
        return explosionObserver;
    }

    /**
     * findes players in a gameobjects array
     * @return ArrayList of type player containing players
     */
     public ArrayList<Player> playerFinde() {
        ArrayList<Player> players = new ArrayList<Player>();
        for(GameObject obj : this.gameObjects) {
            if(obj instanceof Player){
                players.add((Player) obj);
            }
        }
        return players;
    }

    /**
     * Sets up gamunits, controllers and modles for the game
     * @param playerInfo ArrayList of Player subclass Playerinfo type
     * @param numOfAi a int number of ai
     * @param gameSetup a GameSetup
     */
    public void setup(ArrayList<Player.Playerinfo> playerInfo, int numOfAi, GameSetup
gameSetup) {
        Stage stage = this.gameStage;

        //root = new Group();

        root = new StackPane();




        //root.getChildren().
        double width = 1024;
        double height = 720;


        Canvas canvas = new Canvas(width, height);


        gameObjects = new ArrayList<GameObject>();
        ArrayList<Player> aiEnemys = createPlayers(playerInfo);
        ArrayList<Ai> aiPlayers = new ArrayList<Ai>();
        for(int i = 0; i < numOfAi; i++) {
            Player aiUnit = new Player("Ai monster",200 , 40, 0);
            gameObjects.add(aiUnit);
            Ai ai = new Ai(aiUnit,aiEnemys,gameObjects);
            aiUnit.setAiControl(ai);
            aiPlayers.add(ai);
        }


        terrain = new Terrain(width, height);


        ExplosionObserver explosionObserver = observerSetup();
        Collisions collisions = new Collisions(terrain,gameObjects);
        GameModel gameModel = new GameModel(width, height, gameObjects, aiPlayers, collisions,
terrain);
        GameController gameController = new GameController(gameModel, explosionObserver);
        gameUpdate = new GameTimer(gameController);

        // render timer, controller, and GraphicModels
        String[] imgNames = {"buss.png", "Resource/Misil.png",
            "Resource/explosion.png",
            "Resource/Granade.png",
            "Resource/Bullet.png"};
        GraphicModels graphicModels = new GraphicModels();
        graphicModels.loadmodel(imgNames);

        GameView gameView = new GameView(canvas);
        RenderController render = new RenderController(gameView, graphicModels);
        renderTimer = new RenderTimer(render, gameObjects, terrain);



        //MenuBar menubar = new MenuBar();
        TopMenu menu = new TopMenu(this, gameSetup);
        MenuBar menubar = menu.getMenu();
        menubar.prefWidthProperty().bind(gameStage.widthProperty());

        VBox hbox = new VBox();
        HBox menuBox = new HBox();
        menuBox.getChildren().add(menubar);

        HBox gameBox = new HBox();
        gameBox.getChildren().add(canvas);
        gameBox.prefHeight(720);
        StackPane.setMargin(hbox, Insets.EMPTY);

        ArrayList<Player> allPlayers = playerFinde();
        UIStatObserver uIStatObserver = new UIStatObserver(allPlayers);
        HBox uigamestat = uIStatObserver.createUI(root);
        //uigamestat.getChildren().add(gameBox);
        //gameBox.getChildren().add();
        hbox.getChildren().addAll(menuBox,uigamestat,gameBox);
        root.getChildren().addAll(hbox);

        gameStatsObservable = new GameStatsObservable();
        gameStatsObservable.addObserver(uIStatObserver);

        for(Player player : allPlayers) {
            player.setGameStatsObservable(gameStatsObservable);
        }

        Scene scene = new Scene(root, width, height + 50, Color.GREEN);

        this.play();
        stage.setTitle("Lab5Game");
        stage.setScene(scene);
        stage.setResizable(false);
        stage.sizeToScene();
        stage.show();
    }

    /**
     * kills game
     */
    public void killGame() {
        renderTimer.stop();
        gameUpdate.stop();
        removeKeyEvents();

    }

    /**
     * freses game uppdates and renders
     */
    public void paus() {
        gameUpdate.stop();
        renderTimer.stop();
    }

    /**
     * starts game again after paus
     */
    public void play() {
        gameUpdate.start();
        renderTimer.start();
    }
    /**
     * contains info for alert
     */
    public void info(){
        String titel = ("Keybindings");
        String header = ("Standard keybindings");
        //String msg =("Shoot at enemyKeybindings can be changed in lobby");
        String msg =("Shooot and kill your enmey, and take ammoboxes\n"
                +"Unless you have changed the keybinding in lobby\n"
                + "the standardsare:\n"
                + "Player1: Jump Q, Shoot SHIFT,\n"
                + "Aim up/down W/S, Move left/right A/D\n"
                + "\n"
                + "Player2: Jump SPACE, Shoot PERIOD\n,"
                + "Aim up/down UPARROW/DOWNARROW,\n "
                + "Move left/right LEFTARROW/RIGHTARROW\n");
        this.infoMessage(titel, header, msg);
    }

    /**
     * Shows the information alert msg
     * @param title the title for the alert
     * @param header the header
     * @param msg  and the main msg
     */
    public void infoMessage(String title,String header,String msg){
        informationPopup.setTitle(title);
        informationPopup.setHeaderText(header);
        informationPopup.setContentText(msg);
        informationPopup.showAndWait();
    }

    /**
     * The alert for game information
     */
    private final Alert informationPopup = new Alert(AlertType.INFORMATION);

    /**
     * This will open a fileChooser and save the current terrain
     */
    public void saveGame() {
        paus();
        System.out.println("savegame selected");
        System.out.println("loadgame selected");
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save game map");
        File file = fileChooser.showSaveDialog(gameStage);
        try {
            this.terrain.saveTerrain(file);
        } catch (IOException ex) {
            //System.out.println("save error");
            showErrorMsg("Failed to save","Path error","Make sure you save it with a valid name");
        }
    }

    /**
     * loads a saved game file, uses fileChooser, this also handels
     * errors and displays a error msg if you failed to load file
     */
```

```
    public void loadGame() {
        paus();
        System.out.println("loadgame selected");
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Load game map");
        File file = fileChooser.showOpenDialog(gameStage);
        try {
            this.terrain.loadTerrain(file);
        } catch (IOException ex) {
            //System.out.println("loadgame error");
            showErrorMsg("Failed to load","Could not load","File need to be of typ png");

        }catch(IllegalArgumentException ex) {
            showErrorMsg("Failed to load","Path error","File can only be loaded from game directory\n
or valid path");
        }
        renderTimer.start();
    }

    /**
     * Used tp show an erraor alert msg,
     * @param title the title of the alert
     * @param header the header of alert
     * @param msg the msg to be displayed
     */
    public void showErrorMsg(String title,String header,String msg) {

        errorMsg.setTitle(title);
        errorMsg.setHeaderText(header);
        errorMsg.setContentText(msg);
        errorMsg.show();

    }
    /**
     * the alert error msg thing
     */
    private final Alert errorMsg = new Alert(Alert.AlertType.ERROR);
}



/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package lab5game;

import UIGraphics.Lobby;
import UIGraphics.LoginScreen;
import javafx.scene.control.Alert;
import javafx.stage.Stage;
```

```
/**
 *
 * @author o_0
 */
public class GameSetup {
    private LoginScreen loginScreen = null;
    private BattleArena battleArena = null;
    private Lobby lobby = null;

    private Stage mainStage;
    /**
     * This sets the main stage
     * @param stage
     */
    public GameSetup(Stage stage) {
        this.mainStage = stage;
    }

    /**
     * starts the mainmenu
     */
    public void startMainMenu() {
        if(loginScreen == null) {
            loginScreen = new LoginScreen(mainStage, this);
        }
        loginScreen.setup();
    }
    /**
     * starts the lobby
     * @param gameSetup_ this
     */
    public void lobbyStart(GameSetup gameSetup_){
        if(lobby == null){
            lobby = new Lobby(mainStage, gameSetup_);
        }
        lobby.lobbysetup();
    }

    /**
     * starts the game play
     */
    public void startMap() {

        battleArena = new BattleArena(mainStage);
        /**KeyboardController input;
        input = new KeyboardController(KeyCode.A, KeyCode.D, KeyCode.SPACE);
        ArrayList<KeyboardController> keyInputs = new ArrayList<KeyboardController>();
        keyInputs.add(input);
        int numOfAi = 0;*/
        //Ovanstående ska in i en Lobby
        battleArena.setup(lobby.getPlayerInfo(), lobby.getnumOfAi(), this);
        //ovanstående ska ha getters från lobby istället för keyinputs och ai
        //Stoppa allt i en try catch så exception kastas om ai eller keyinpus inte finns
    }

    /**
     * exit program
     */
    public void exitGame() {
        mainStage.close();
    }

    /**
     * resets game
     */
    public void resetGame() {
        if(battleArena != null) {
            battleArena.killGame();
        }

        startMainMenu();
    }

}


package lab5game;

import javafx.application.Application;
import javafx.stage.Stage;

/**
 *Simply a small class containing main and the startup method
 * @author o_0
 */
public class Lab5game extends Application{
    private GameSetup game;
    @Override
    /**
     * Method creating the GameSetup and uses the startMainMenu method
     */
    public void start(Stage stage) throws Exception {
        game = new GameSetup(stage);
        game.startMainMenu();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
```

```
}
```