```
/*src/application/Main.java*/
/*
 * JavaFx Main Class - The beginning of every thing
 */
package application;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import logic.InputUtility;
import logic.SoundManager;
import ui.GamePane;
import ui.IStoppable;
import ui.MainPane;

public class Main extends Application {

    private static final int SCREEN_WIDTH = 1200;
    private static final int SCREEN_HEIGHT = 600;
    private static Scene scene;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        MainPane mainPane = new MainPane(SCREEN_WIDTH, SCREEN_HEIGHT);
        scene = new Scene(mainPane, SCREEN_WIDTH, SCREEN_HEIGHT);
        scene.getStylesheets().add("bootstrapfx.css");
        InputUtility.instance.setEventHandler(scene);

        primaryStage.setTitle("Super Killing Wars");
        primaryStage.setResizable(true);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    @Override
    public void stop() {
        ((IStoppable) scene.getRoot()).stop();
    }

    public static void changeSceneToGame() {
        ((IStoppable) scene.getRoot()).stop();
        GamePane gamePane = new GamePane(SCREEN_WIDTH, SCREEN_HEIGHT);
        scene.setRoot(gamePane);
        SoundManager.start();
    }

    public static void changeSceneToMain() {
        ((IStoppable) scene.getRoot()).stop();
        SoundManager.stop();
        MainPane mainPane = new MainPane(SCREEN_WIDTH, SCREEN_HEIGHT);
        scene.setRoot(mainPane);
    }

    public static Scene getScene() {
        return scene;
    }
}

/*src/exception/HighscoreException.java*/
/*
 * throws when there is an error while getting high score (online)
 */
package exception;

public class HighscoreException extends Exception {
    private static final long serialVersionUID = 6882787462646581816L;

    public HighscoreException() {
        super("Error can't get high score");
    }
}

/*src/exception/InvalidNameException.java*/
/*
 * throws when trying to set name that contains spaces
 */
package exception;

public class InvalidNameException extends Exception {
    private static final long serialVersionUID = 4658717663341535289L;

    public InvalidNameException() {
        super("Name shouldn't contains spaces");
    }
}

/*src/logic/IMovable.java*/
/*
 * Object that can be moved (ex. MovingEntity)
 */
package logic;

public interface IMovable {
    public void move();
}

/*src/logic/SoundManager.java*/
/*
 * Managing sounds
 */
package logic;

import javafx.scene.media.AudioClip;

public class SoundManager {
    private static AudioClip bgm;
    private static AudioClip gunshot;
    private static AudioClip getRes(String path){
        return new AudioClip(ClassLoader.getSystemResource(path).toString());
    }
    static{
        bgm=getRes("sound/bgm.wav");
        bgm.setCycleCount(AudioClip.INDEFINITE);

        gunshot=getRes("sound/gunshot.wav");

        setVolume(0.2);
    }

    public static void start() {
        bgm.play();
    }

    public static void stop() {
        bgm.stop();
    }

    public static void setVolume(double value) {
        bgm.setVolume(value);
        gunshot.setVolume(value);
    }

    public static double getVolume() {
        return bgm.getVolume();
    }

    public static AudioClip getBgm() {
        return bgm;
    }

    public static AudioClip getGunshot() {
        return gunshot;
    }
}
/*src/logic/EnemyManager.java*/
/*
 * Managing enemy spawning (spawn in waves)
 */
package logic;

import model.Tile;
import model.TileSpawner;
import model.enemy.Enemy;
import model.enemy.EnemyBasic;
import model.enemy.EnemyBoss;
import java.util.ArrayList;
import java.util.Collections;

public class EnemyManager {
    public static EnemyManager instance;

    private int timer = 0;
    private int wave = 0;
    private static final int WAVE_DELAY = 1200; // 20 secs
    private static final int ROCKET_WAVE_DELAY = 120; // 2 secs

    public int getWaveNumber() {
        return wave;
    }

    public void update() {
        timer++;
        if (GameManager.instance.getRocketCount() > 0) {
            if (timer >= ROCKET_WAVE_DELAY) {
                timer = 0;
                spawn();
            }
        } else if (timer >= WAVE_DELAY) {
            timer = 0;
            wave++;
            spawn();
        }
    }

    public int getRemainingTime() {
        if (GameManager.instance.getRocketCount() > 0)
            return ROCKET_WAVE_DELAY - timer;
        return WAVE_DELAY - timer;
    }
    public String getNextWaveName() {
        if (GameManager.instance.getRocketCount() > 0)
            return "Rocket";
        else if (isBigWave(wave + 1))
            return "Big";
        else if (isBossWave(wave + 1))
            return "Boss";
        else
            return "Normal";
    }

    private boolean isBigWave(int wave) {
        return wave % 5 == 4;
    }

    private boolean isBossWave(int wave) {
        return wave % 5 == 0;
    }

    private void spawn() {
        // calculate number of each type of enemy
        int basicCount, bossCount;
        int level = wave;
        if (GameManager.instance.getRocketCount() > 0) { // rocket wave
            level = 85;
            basicCount = 2;
            bossCount = 1;
        } else if (isBigWave(wave)) { // big wave
            basicCount = GameManager.globalRNG.nextInt(5) + 5; // 5 - 9
            bossCount = 1;
        } else if (isBossWave(wave)) { // boss wave
            basicCount = GameManager.globalRNG.nextInt(3) + 1; // 1 - 3
            bossCount = GameManager.globalRNG.nextInt(3) + 3; // 3 - 5
        } else { // normal wave
            basicCount = GameManager.globalRNG.nextInt(3) + 3; // 3 - 5
            bossCount = 1;
        }
        spawnEnemiesOnSpawner(basicCount, bossCount, level);
    }

    private void spawnEnemiesOnSpawner(int basicCount, int bossCount, int level) {
        ArrayList<Tile> spawnableTile = new ArrayList<>();
        for (Tile tile : TileManager.instance.tileList) {
            if (tile instanceof TileSpawner) {
                spawnableTile.add(tile);
            }
        }
        Collections.shuffle(spawnableTile, GameManager.globalRNG);
        for (Tile tile : spawnableTile) {
            Enemy enemy;
            if (basicCount > 0) {
                enemy = new EnemyBasic(tile.getX(), tile.getY(), level);
                basicCount--;
            } else if (bossCount > 0) {
                enemy = new EnemyBoss(tile.getX(), tile.getY(), level);
                bossCount--;
            } else
                break;

            GameManager.addEntity(enemy);

            // if this enemy collide with other when created then destroy it
            if (CollisionUtility.isBlocked(enemy)) {
                enemy.destroy();
            }
        }
    }
}

/*src/logic/IRenderable.java*/
/*
 * Interface for renderable objects (can be drawn on canvas)

 */
package logic;

import javafx.scene.canvas.GraphicsContext;

public interface IRenderable {
    int getZ();

    void draw(GraphicsContext gc);

    boolean isDestroy();
}

/*src/logic/InputUtility.java*/
/*
 * Utility for managing input events
 */
package logic;

import java.util.ArrayList;

import javafx.scene.Scene;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
```

```java
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;

public class InputUtility {
    public static InputUtility instance = new InputUtility();

    private static ArrayList<KeyCode> keyPressed = new ArrayList<>();
    private static ArrayList<KeyCode> keyTriggered = new ArrayList<>();
    private double mouseX, mouseY;
    private boolean isMouseLeftTriggered, isMouseLeftDown;
    private boolean isMouseRightTriggered, isMouseRightDown;
    private boolean isMouseOnScreen;

    public InputUtility() {
        instance = this;
        isMouseLeftTriggered = isMouseLeftDown = false;
        isMouseRightTriggered = isMouseRightDown = false;
    }

    public void setEventHandler(Scene scene) {
        scene.setOnKeyPressed((KeyEvent event) -> {
            if (!keyPressed.contains(event.getCode()))
                keyPressed.add(event.getCode());
            if (!keyTriggered.contains(event.getCode()))
                keyTriggered.add(event.getCode());
        });
        scene.setOnKeyReleased((KeyEvent event) -> {

            keyPressed.remove(event.getCode());
        });

        scene.setOnMousePressed((MouseEvent event) -> {
            if (event.getButton() == MouseButton.PRIMARY) {
                isMouseLeftDown = true;
                isMouseLeftTriggered = true;
            } else if (event.getButton() == MouseButton.SECONDARY) {
                isMouseRightDown = true;
                isMouseRightTriggered = true;
            }
        });

        scene.setOnMouseReleased((MouseEvent event) -> {
            if (event.getButton() == MouseButton.PRIMARY) {
                isMouseLeftDown = false;
            } else if (event.getButton() == MouseButton.SECONDARY) {
                isMouseRightDown = false;
            }
        });

        scene.setOnMouseEntered((MouseEvent event) -> {
            isMouseOnScreen = true;
        });

        scene.setOnMouseExited((MouseEvent event) -> {
            isMouseOnScreen = false;
        });
        scene.setOnMouseDragged((MouseEvent event) -> {
            // Mouse drag = mouse moved while pressing
            mouseX = event.getX();
            mouseY = event.getY();
        });
        scene.setOnMouseMoved((MouseEvent event) -> {
            mouseX = event.getX();
            mouseY = event.getY();
        });
    }

    public void reset() {
        // clear triggered status
        isMouseLeftTriggered = false;
        isMouseRightTriggered = false;
        keyTriggered.clear();
    }

    public boolean isMouseLeftClicked() {
        return isMouseLeftTriggered;
    }

    public boolean isMouseLeftDown() {
        return isMouseLeftDown;
    }

    public boolean isMouseRightClicked() {
        return isMouseRightTriggered;
    }

    public boolean isMouseRightDown() {
        return isMouseRightDown;
    }

    public double getMouseX() {
        return mouseX;
    }

    public double getMouseY() {
        return mouseY;
    }
    public boolean isKeyDown(KeyCode a) {
        return keyPressed.contains(a);
    }

    public boolean isKeyTriggered(KeyCode a) {
        return keyTriggered.contains(a);
    }

    public boolean isMouseOnScreen() {
        return isMouseOnScreen;
    }
}

/*src/logic/GameManager.java*/
/*
 * Managing game states and update every entities
 */
package logic;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.util.Random;

import application.Main;
import javafx.application.Platform;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.input.KeyCode;
import model.*;
import thread.ThreadGameManager;
import thread.ThreadNewScore;
import ui.MainPane;

public class GameManager {
    public static GameManager instance;
    public static Random globalRNG = new Random();

    private ThreadGameManager thread;

    private int score = 0;
    private Player player;
    private int fps = 0;
    private boolean isGamePause = false;
    private boolean isRocketLaunched = false;
    private int rocketCount = 0;

    public GameManager() {
        // initialize singleton
        RenderableHolder.instance = new RenderableHolder();

        player = new Player(10, 10);
        addEntity(player);

        BuyManager.instance = new BuyManager();
        ResourceManager.instance = new ResourceManager();
        TileManager.instance = new TileManager();
        TileManager.instance.generateMap(globalRNG.nextInt(99999));
        EnemyManager.instance = new EnemyManager();
    }

    public static void addEntity(IRenderable entity) {
        RenderableHolder.instance.add(entity);
    }

    public void update() {
        synchronized (RenderableHolder.instance.getEntities()) {
            if (isGameEnded()) {
                return;
            }
            if (!isPause()) {
                updateOverlay();
                updateEntity();
                CollisionUtility.checkCollision();
                removeDestroyEntity();
                EnemyManager.instance.update();
            }
            if (isGameEnded()) {
                onGameEnded();
            }
            InputUtility.instance.reset();
        }
    }

    private boolean isPause() {
        if (InputUtility.instance.isKeyTriggered(KeyCode.P)) {
            isGamePause = !isGamePause;
        }
        return isGamePause;
    }

    private boolean isGameEnded() {
        return player.isDestroy() || isRocketLaunched;
    }

    private void onGameEnded() {
        Platform.runLater(() -> {
            Alert alert = new Alert(AlertType.INFORMATION);
            if (isRocketLaunched) {
                alert.setContentText("You win");
                alert.setHeaderText("Congratulations!");
            } else {
                alert.setContentText("Game Over");
                alert.setHeaderText("Try again later.");
            }
            alert.show();
        });
        new ThreadNewScore(MainPane.getName(), score).start();
        Main.changeSceneToMain();
    }

    private void updateEntity() {
        for (int i = 0; i < RenderableHolder.instance.getEntities().size(); i++) {
            IRenderable ir = RenderableHolder.instance.getEntities().get(i);
            if (ir instanceof Entity) {
                ((Entity) ir).update();
            }
        }
    }

    private void updateOverlay() {
        if (!BuyManager.instance.isBuyMode)
            return;
        // Exit buyMode if right click
        if (InputUtility.instance.isMouseRightClicked()) {
            BuyManager.instance.isBuyMode = false;
            return;
        }
        // Try to place
        if (InputUtility.instance.isMouseLeftClicked()) {
            int x = (int) (InputUtility.instance.getMouseX() / TileManager.TILE_SIZE);
            int y = (int) (InputUtility.instance.getMouseY() / TileManager.TILE_SIZE);
            int[] resourceNeeded;
            try {
                Method getResourceNeeded = BuyManager.instance.currentObjectClass.getMethod("getResourceNeeded");
                resourceNeeded = (int[]) getResourceNeeded.invoke(null);
                if (BuyManager.instance.canBuy()) {
                    for (int i = 0; i < 5; i++) {
                        ResourceManager.instance.addResource(i, -resourceNeeded[i]);
                    }
                    Constructor con = BuyManager.instance.currentObjectClass.getDeclaredConstructor(Tile.class);
                    con.newInstance(TileManager.instance.tileArray[x][y]);

                    // keep buying if shift is down
                    if (!InputUtility.instance.isKeyDown(KeyCode.SHIFT))
                        BuyManager.instance.isBuyMode = false;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
    }

    private void removeDestroyEntity() {
        for (int i = RenderableHolder.instance.getEntities().size() - 1; i >= 0; i--) {
            if (RenderableHolder.instance.getEntities().get(i).isDestroy())
                RenderableHolder.instance.remove(i);
        }
    }

    public void setRocketLaunched(boolean launched) {
        this.isRocketLaunched = launched;
    }

    public int getRocketCount() {
        return rocketCount;
    }

    public void setRocketCount(int rocketCount) {
        this.rocketCount = rocketCount;
    }

    public boolean isGamePause() {
        return isGamePause;
    }

    public Player getPlayer() {
        return player;
    }

    public void increaseScore(int amount) {
        this.score += amount;
    }

    public int getScore() {
        return score;
    }

    public int getFps() {
        return fps;
    }

    public void setFps(int fps) {
        this.fps = fps;
    }

    public void startUpdateThread() {
        thread = new ThreadGameManager();
        thread.start();
    }

    public void stopUpdateThread() {
        thread.interrupt();
    }
}

/*src/logic/HighscoreManager.java*/
/*
 * Managing HTTP request to get highscore from the server.
 */
package logic;

import java.net.*;

import exception.HighscoreException;

import java.io.*;

public class HighscoreManager {
    // simple http request utility function
    private static String req(String path) throws HighscoreException {
        try {
            URL url = new URL("http://do.op01.tk/" + path);
            URLConnection con = url.openConnection();
            InputStreamReader isr = new InputStreamReader(con.getInputStream());
            BufferedReader in = new BufferedReader(isr);
            String inputLine, ret = "";
            while ((inputLine = in.readLine()) != null) {
                ret += inputLine + "\n";
            }
            in.close();
            return ret;
        } catch (Exception e) {
            throw new HighscoreException();
        }
    }

    public static String getScore() throws HighscoreException {
        return req("highscore.php");
    }

    public static void postScore(String name, int score) throws HighscoreException {
        req("newscore.php?name=" + name + "&score=" + score);
    }
}
/*src/logic/IBlockable.java*/
/*
 * Interface for blockable object (object that can t be collide with other blockable object)
 */
package logic;

// 2 Entity of IBlockable can't collide with each other
public interface IBlockable extends ICollidable {
    public void undoMove(); // call when objects collided to undo move
}

/*src/logic/BuyManager.java*/
/*
 * Manage items buying
 */
package logic;

import java.lang.reflect.Method;

import javafx.scene.image.Image;
import model.Tile;

public class BuyManager {
    public static BuyManager instance;
    public boolean isBuyMode = false;
    public Image currentObjectImage = null;
    public Class currentObjectClass = null;

    public boolean canBuy() {
        int x = TileManager.getMouseTileX();
        int y = TileManager.getMouseTileY();
        if (TileManager.isOutOfBound(x, y))
            return false;
        boolean ret = true;
        try {
            // call methods on currentObjectClass class (static)
            Method canPlace = BuyManager.instance.currentObjectClass.getMethod("canPlace", Tile.class);
            ret = (boolean) canPlace.invoke(null, TileManager.instance.tileArray[x][y]);
            Method getResourceNeeded = BuyManager.instance.currentObjectClass.getMethod("getResourceNeeded");
            int[] resourceNeeded = (int[]) getResourceNeeded.invoke(null);
            for (int i = 0; i < 5; i++) {
                if (ResourceManager.instance.getResource(i) < resourceNeeded[i])
                    ret = false;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return ret;
    }
}

/*src/logic/ICollidable.java*/
/*
 * Interface for collidable object (Can check collision with other Collidable Object)
 */
package logic;

/* ICollidable entity will be called by CollisionManager */

public interface ICollidable {
    public void onCollision(ICollidable entity);

    public double getX();

    public double getY();

    public double getWidth();

    public double getHeight();
}

/*src/logic/CollisionUtility.java*/
/*
 * Methods for calculating collisions
 */
package logic;

import model.RenderableHolder;

public class CollisionUtility {

    public static boolean isCollide(ICollidable o1, ICollidable o2) {
        // rectangle collision detection
        if (o1.getX() >= o2.getX() + o2.getWidth())
            return false;
        if (o1.getX() + o1.getWidth() <= o2.getX())
            return false;
        if (o1.getY() >= o2.getY() + o2.getHeight())
            return false;
        if (o1.getY() + o1.getHeight() <= o2.getY())
            return false;
        return true;
    }

    public static boolean isBlocked(ICollidable object) {
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof IBlockable && ir != object) {
                if (isCollide((ICollidable) ir, object)) {
                    return true;
                }
            }
        }
        return false;
    }

    public static void checkCollision() {
        int n = RenderableHolder.instance.getEntities().size();
        for (int i = 0; i < n; i++) {
            IRenderable e1 = RenderableHolder.instance.getEntities().get(i);
            if (!(e1 instanceof ICollidable))
                continue;
            for (int j = i + 1; j < n; j++) {
                IRenderable e2 = RenderableHolder.instance.getEntities().get(j);
                if (e2 instanceof ICollidable) {
                    ICollidable o1 = (ICollidable) e1;
                    ICollidable o2 = (ICollidable) e2;
                    // check hit
                    if (isCollide(o1, o2)) {
                        o1.onCollision(o2);
                        o2.onCollision(o1);
                    }
                }
            }
        }
    }

    public static double findDistance(ICollidable ic1, ICollidable ic2) {
        double x1 = ic1.getX() + ic1.getWidth() / 2;
        double x2 = ic2.getX() + ic2.getWidth() / 2;
        double y1 = ic1.getY() + ic1.getHeight() / 2;
        double y2 = ic2.getY() + ic2.getHeight() / 2;
        return Math.hypot(x1 - x2, y1 - y2);
    }
}

/*src/logic/TileManager.java*/
/*
 * Managing tiles
 */
package logic;
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import model.RenderableHolder;
import model.Tile;
import model.TileGround;
import model.TileVoid;
import model.tileObject.TileObjectStone;
import model.tileObject.TileObjectTree;
import model.tileObject.TileObjectVoid;
import model.TileSpawner;

public class TileManager {
    public static TileManager instance;

    public static final int TILE_COUNT_X = 30;
    public static final int TILE_COUNT_Y = 20;
    public static final double TILE_SIZE = 30;

    public List<Tile> tileList;
    public Tile[][] tileArray; // Use X,Y coordinate system

    public TileManager() {
        tileList = new ArrayList<Tile>();
        tileArray = new Tile[TILE_COUNT_X][TILE_COUNT_Y];
    }

    private ICollidable createCollidableFromTile(Tile tile, int sizeX, int sizeY) {
        // create temporary ICollidable object to check collision with entities
        ICollidable ic = new ICollidable() {
            @Override
            public double getY() {
                return tile.getY();
            }

            @Override
            public double getX() {
                return tile.getX();
            }

            @Override
            public double getWidth() {
                return sizeX * TileManager.TILE_SIZE;
            }

            @Override
            public double getHeight() {
                return sizeY * TileManager.TILE_SIZE;
            }

            @Override
            public void onCollision(ICollidable entity) {
            }
        };

        return ic;
    }

    public boolean canPlace(Tile tile, int sizeX, int sizeY) {
        // can't place on tile that has Blocking entity
        ICollidable ic = createCollidableFromTile(tile, sizeX, sizeY);
        if (CollisionUtility.isBlocked(ic))
            return false;

        for (int dx = 0; dx < sizeX; dx++) {
            for (int dy = 0; dy < sizeY; dy++) {
                int x = tile.getTileX() + dx;
                int y = tile.getTileY() + dy;
                if (x < 0 || y < 0 || x >= TileManager.TILE_COUNT_X || y >= TileManager.TILE_COUNT_Y) {
                    return false;
                }
                if (tileArray[x][y].getTileObject() != null) {
                    return false; // already have object
                }
                if (tileArray[x][y] instanceof TileSpawner)
                    return false; // can't place on spawner
            }
        }
        return true;
    }

    public void generateMap(int seed) {
        Random random = new Random(seed);
        for (int x = -1; x <= TILE_COUNT_X; x++) {
            for (int y = -1; y <= TILE_COUNT_Y; y++) {
                if (x == -1 || y == -1 || x == TILE_COUNT_X || y == TILE_COUNT_Y) {
                    TileVoid vt = new TileVoid(x, y);
                    new TileObjectVoid(vt);
                    tileList.add(vt);
                } else {
                    if (x >= TILE_COUNT_X - 2) {
                        tileArray[x][y] = new TileSpawner(x, y);
                    } else {
                        tileArray[x][y] = new TileGround(x, y);
                    }
                    tileList.add(tileArray[x][y]);
                    RenderableHolder.instance.add(tileArray[x][y]);
                }
            }
        }
        // add some tree & stone
        for (Tile t : tileList) {
            if (random.nextInt(100) < 10 && TileObjectTree.canPlace(t)) {
                new TileObjectTree(t);
            } else if (random.nextInt(100) < 5 && TileObjectStone.canPlace(t)) {
                new TileObjectStone(t);
            }
        }
    }

    public static int getMouseTileX() {
        return (int) (InputUtility.instance.getMouseX() / TILE_SIZE);
    }

    public static int getMouseTileY() {
        return (int) (InputUtility.instance.getMouseY() / TILE_SIZE);
    }

    public static boolean isOutOfBound(int x, int y) {
        return x >= TILE_COUNT_X || x < 0 || y >= TILE_COUNT_Y || y < 0;
    }

}

/*src/logic/ResourceManager.java*/
/*
 * Managing resources
 */
package logic;

public class ResourceManager {
    public static final int WOOD = 0, STONE = 1, IRON = 2, DIAMOND = 3, ARTIFACT = 4;
    public static final String[] NAME = { "Wood", "Stone", "Iron", "Diamond", "Alien Artifact" };

    // x artifacts = 1 target unit
    public static final int[] EXCHANGE_RATE = { 1, 10, 100, 250 };

    public static ResourceManager instance;

    private int[] capacity = new int[] { 0, 0, 0, 0, 9999 };
    private int[] resource = new int[] { 0, 0, 0, 0, 0 };

    public int getCapacity(int index) {
        return capacity[index];
    }

    public int getResource(int index) {
        return resource[index];
    }

    public void addResource(int index, int amount) {
        resource[index] += amount;
        normalize(index);
    }

    public void addCapacity(int index, int amount) {
        capacity[index] += amount;
        normalize(index);
    }

    // check resource overflow
    private void normalize(int index) {
        if (resource[index] > capacity[index]) {
            resource[index] = capacity[index];
        }
    }
}

/*src/model/TileVoid.java*/
/*
 * Tile void (outside of game screen for placing tile object void)
 */
package model;

import javafx.scene.canvas.GraphicsContext;

public class TileVoid extends Tile {

    public TileVoid(int tileX, int tileY) {
        super(tileX, tileY);
    }

    @Override
    public void draw(GraphicsContext gc) {

    }

}

/*src/model/Entity.java*/
/*
 * All object in game
 */
package model;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import logic.ICollidable;
import logic.IRenderable;

/* Every entity is :
 * - Destroyable (has hp)
 * - collide (Collision detection)
 * - Renderable
 */

public abstract class Entity implements IRenderable, ICollidable {
    protected double x, y;
    protected double width, height;
    protected boolean isDestroyed;
    protected int hp;
    protected int maxHp;

    private static final double HEALTHBAR_WIDTH = 20;
    private static final double HEALTHBAR_HEIGHT = 3;

    public Entity(double x, double y, double width, double height, int hp) {
        this.isDestroyed = false;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.hp = this.maxHp = hp;
    }

    public void destroy() {
        if (this.isDestroyed)
            return;
        this.isDestroyed = true;
        this.onDestroy();
    }

    protected void onDestroy() {

    }

    public void update() {
        if (this.hp <= 0 && !this.isDestroyed) {
            this.isDestroyed = true;
            this.onDestroy();
        }
    }

    public void onCollision(ICollidable collider) {

    }

    @Override
    public boolean isDestroy() {
        return isDestroyed;
    }

    public void setHp(int hp) {
        this.hp = hp;
    }

    public int getHp() {
        return this.hp;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getHeight() {
        return height;
    }
    public void setHeight(double height) {
        this.height = height;
    }

    public void drawHealthBar(GraphicsContext gc) {
        if(this.hp == this.maxHp) return; // don't draw when at full hp
        gc.setFill(Color.RED);
        gc.fillRect(getCenterX()-HEALTHBAR_WIDTH/2, y-5, HEALTHBAR_WIDTH, HEALTHBAR_HEIGHT);
        gc.setFill(Color.LIME);
        gc.fillRect(getCenterX()-HEALTHBAR_WIDTH/2, y-5, HEALTHBAR_WIDTH * (double)hp/maxHp , HEALTHBAR_HEIGHT);
    }

    // default draw function for all entity
    protected void draw(GraphicsContext gc, Image img) {
        gc.drawImage(img, x, y, width, height);
    }

    public void reduceHP(int damage) {
        this.hp -= damage;
    }

    public double getCenterX() {
        return x+width/2;
    }

    public double getCenterY() {
        return y+height/2;
    }
}

/*src/model/Tile.java*/
/*
 * A tile (background)
 */
package model;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import logic.IRenderable;
import logic.TileManager;
import model.tileObject.TileObject;

public abstract class Tile implements IRenderable {
    private TileObject tileObject = null; // object on this tile
    protected double x, y;
    protected int tileX, tileY; // x,y in tile grid

    public Tile(int tileX, int tileY) {
        this.tileX = tileX;
        this.tileY = tileY;
        this.x = tileX * TileManager.TILE_SIZE;
        this.y = tileY * TileManager.TILE_SIZE;
    }

    public int getTileX() {
        return this.tileX;
    }

    public int getTileY() {
        return this.tileY;
    }

    public double getX() {
        return this.x;
    }

    public double getY() {
        return this.y;
    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    @Override
    public int getZ() {
        return -10;
    }

    @Override
    public boolean isDestroy() {
        return false;
    }

    public void setTileObject(TileObject tileObject) {
        this.tileObject = tileObject;
    }

    public void draw(GraphicsContext gc, Image img) {
        gc.drawImage(img, x, y, TileManager.TILE_SIZE, TileManager.TILE_SIZE);
    }

    public TileObject getTileObject() {
        return tileObject;
    }
}

/*src/model/RenderableHolder.java*/
/*
 * Holder for IRenderable objects
 */
package model;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import javafx.scene.image.Image;
import logic.IRenderable;

public class RenderableHolder {
    private List<IRenderable> entities;
    private Comparator<IRenderable> comparator;

    public static Image volume_img;
    public static Image tile_ground_img;
    public static Image tile_spawner_img;

    public static Image tileObject_tree_img;
    public static Image tileObject_stone_img;

    public static Image[] resource_img;

    public static Image projectile_arrow_img;
    public static Image projectile_laser_img;
    public static Image projectile_rock_img;
    public static Image projectile_bullet_img;
    public static Image projectile_bomb_img;

    public static Image player_img;
    public static Image enemy_basic_img;
    public static Image enemy_boss_img;

    public static Image tower_arrow_img;
    public static Image tower_laser_img;
    public static Image tower_catapult_img;
    public static Image tower_turret_img;
    public static Image tower_sniper_img;
    public static Image tower_bomb_img;

    public static Image tileObject_wall_wood_img;
    public static Image tileObject_wall_stone_img;
    public static Image tileObject_wall_iron_img;

    public static Image generator_wood_img;
    public static Image generator_stone_img;
    public static Image generator_iron_img;
    public static Image generator_diamond_img;

    public static Image storage_wood_img;
    public static Image storage_stone_img;
    public static Image storage_iron_img;
    public static Image storage_diamond_img;

    public static Image research_building_img;
    public static Image research_iron_img;
    public static Image research_diamond_img;
    public static Image research_gun_img;
    public static Image research_supergun_img;
    public static Image research_regen_img;
    public static Image research_smallgun_img;

    public static Image tileObject_rocket_img;

    public static RenderableHolder instance;

    public RenderableHolder() {
        entities = new ArrayList<>();
        comparator = new Comparator<IRenderable>() {
            public int compare(IRenderable a, IRenderable b) {
                return Integer.compare(a.getZ(), b.getZ());
            }
        };
    }

    static {
        loadResource();
    }

    public void add(IRenderable entity) {
        instance.entities.add(entity);
        instance.entities.sort(comparator);
    }

    private static Image getRes(String path) {
        return new Image(ClassLoader.getSystemResource(path).toString());
    }

    private static void loadResource() {
        volume_img = getRes("img/ui/volume.png");
        tile_ground_img = getRes("img/tile/ground.png");
        tile_spawner_img = getRes("img/tile/spawner.png");

        resource_img = new Image[5];
        resource_img[0] = getRes("img/ui/res_wood.png");
        resource_img[1] = getRes("img/ui/res_stone.png");
        resource_img[2] = getRes("img/ui/res_iron.png");
        resource_img[3] = getRes("img/ui/res_diamond.png");
        resource_img[4] = getRes("img/ui/res_alienArtifact.png");

        tileObject_tree_img = getRes("img/tileObject/tree.png");
        tileObject_stone_img = getRes("img/tileObject/stone.png");

        player_img = getRes("img/entity/player.png");
        enemy_basic_img = getRes("img/entity/enemy.png");
        enemy_boss_img = getRes("img/entity/enemy_boss.png");

        tower_arrow_img = getRes("img/tileObject/towerArrow.png");
        tower_laser_img = getRes("img/tileObject/towerLaser.png");
        tower_catapult_img = getRes("img/tileObject/towerCatapult.png");
        tower_turret_img = getRes("img/tileObject/towerTurret.png");
        tower_bomb_img = getRes("img/tileObject/towerBomb.png");
        tower_sniper_img = getRes("img/tileObject/towerSniper.png");

        projectile_arrow_img = getRes("img/projectile/arrow.png");
        projectile_rock_img = getRes("img/projectile/rock.png");
        projectile_bomb_img = getRes("img/projectile/bomb.png");
        projectile_laser_img = getRes("img/projectile/laser.png");
        projectile_bullet_img = getRes("img/projectile/bullet.png");

        tileObject_wall_wood_img = getRes("img/tileObject/wallWood.png");
        tileObject_wall_stone_img = getRes("img/tileObject/wallStone.png");
        tileObject_wall_iron_img = getRes("img/tileObject/wallIron.png");

        generator_wood_img = getRes("img/tileObject/generatorWood.png");
        generator_stone_img = getRes("img/tileObject/generatorStone.png");
        generator_iron_img = getRes("img/tileObject/generatorIron.png");
        generator_diamond_img = getRes("img/tileObject/generatorDiamond.png");

        storage_wood_img = getRes("img/tileObject/storageWood.png");
        storage_stone_img = getRes("img/tileObject/storageStone.png");
        storage_iron_img = getRes("img/tileObject/storageIron.png");
        storage_diamond_img = getRes("img/tileObject/storageDiamond.png");

        research_building_img = getRes("img/ui/research_building.png");
        research_iron_img = getRes("img/ui/research_iron.png");
        research_diamond_img = getRes("img/ui/research_diamond.png");
        research_gun_img = getRes("img/ui/research_gun.png");
        research_supergun_img = getRes("img/ui/research_supergun.png");
        research_regen_img = getRes("img/ui/research_regen.png");
        research_smallgun_img = getRes("img/ui/research_smallGun.png");

        tileObject_rocket_img = getRes("img/tileObject/rocketSilo.png");

    }

    public void remove(int index) {
        instance.entities.remove(index);
    }

    public List<IRenderable> getEntities() {
        return entities;
    }
}

/*src/model/TileGround.java*/
/*
 * Tile ground (grass)
 */
package model;

import javafx.scene.canvas.GraphicsContext;

public class TileGround extends Tile {
    public TileGround(int tileX, int tileY) {
        super(tileX, tileY);
    }

    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tile_ground_img);
    }
}

/*src/model/BlockingEntity.java*/
/*
 * MovableEntity that is also blockable
 */
package model;

import logic.CollisionUtility;
import logic.IBlockable;

public abstract class BlockingEntity extends MovingEntity implements IBlockable {
    protected double lastX, lastY;

    public BlockingEntity(double x, double y, double width, double height, double speed, int hp) {
        super(x, y, width, height, speed, hp);
    }

    public void move() {
        // move with blocking behavior
        lastX = x;
        lastY = y;
        x += velX * speed;
        if (CollisionUtility.isBlocked(this))
            x = lastX;
        y += velY * speed;
        if (CollisionUtility.isBlocked(this))
            y = lastY;
    }

}

/*src/model/MovingEntity.java*/
/*
 * Entity that can move
 */
package model;

import logic.IMovable;

public abstract class MovingEntity extends Entity implements IMovable {
    protected double speed;
    protected double velX, velY; // velocity (vector)

    public MovingEntity(double x, double y, double width, double height, double speed, int hp) {
        super(x, y, width, height, hp);
        this.speed = speed;
        velX = velY = 0;
    }

    @Override
    public void update() {
        super.update();
        if (!this.isDestroyed)
            move();
    }

    @Override
    public void move() {
        // move
        x += velX * speed;
        y += velY * speed;
    }

    public double getSpeed() {
        return speed;
    }

    public void setSpeed(double speed) {
        this.speed = speed;
    }

    public double getVelX() {
        return velX;
    }

    public void setVelX(double velX) {
        this.velX = velX;
    }

    public double getVelY() {
        return velY;
    }

    public void setVelY(double velY) {
        this.velY = velY;
    }

}

/*src/model/ScoreRecord.java*/
/*
 * Store score record for sort and display in high score
 */
package model;

public class ScoreRecord implements Comparable<ScoreRecord> {
    private int score;
    private String name;

    public ScoreRecord(int score, String name) {
        this.score = score;
        this.name = name;
    }

    @Override
    public int compareTo(ScoreRecord o) {
        return -Integer.compare(score, o.score);
    }

    @Override
    public String toString() {
        return name + " : " + score;
    }

}

/*src/model/TileSpawner.java*/
/*
 * Tile spawner (enemy spawn on this tile)
 */
package model;

import javafx.scene.canvas.GraphicsContext;

public class TileSpawner extends Tile {
    public TileSpawner(int tileX, int tileY) {
        super(tileX, tileY);
    }

    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tile_spawner_img);
    }
}

/*src/model/Player.java*/
/*
 * Player entity
 */
package model;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.input.KeyCode;
import logic.BuyManager;
import logic.GameManager;
import logic.InputUtility;
import logic.SoundManager;
import logic.TileManager;
import model.projectile.Projectile;
import model.projectile.ProjectilePlayerBullet;
import model.tileObject.TileObject;

public class Player extends BlockingEntity {

    private static final double SPEED = 5;
    private static final double WIDTH = 20;
    private static final double HEIGHT = 20;
    private static final int START_HP = 750;

    private int healthRegenerationTimer = 0;
    private static final int HEALTH_REGEN_DELAY = 10;
    private static int healthRegenerationRate = 0;

    private int shootingTimer = 0;
    private static final int SHOOTING_DELAY = 15;
    private static final int HARVEST_POWER = 1;

    public Player(double x, double y) {
        super(x, y, WIDTH, HEIGHT, SPEED, START_HP);
        shootingTimer = SHOOTING_DELAY;
    }

    @Override
    public void move() {
        super.move();
        velX = velY = 0;
    }

    @Override
    public void undoMove() {
        x = lastX;
        y = lastY;
    }

    @Override
    public int getZ() {
        return 0;
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.player_img);
    }

    private void updateVelocity() {
        if (InputUtility.instance.isKeyDown(KeyCode.A))
            setVelX(-1);
        if (InputUtility.instance.isKeyDown(KeyCode.D))
            setVelX(1);
        if (InputUtility.instance.isKeyDown(KeyCode.W))
            setVelY(-1);
        if (InputUtility.instance.isKeyDown(KeyCode.S))
            setVelY(1);
    }

    private void updateShoot() {
        if (shootingTimer < SHOOTING_DELAY) {
            shootingTimer++;
        } else {
            if (!BuyManager.instance.isBuyMode && InputUtility.instance.isMouseLeftDown()) {
                SoundManager.getGunshot().play();
                Projectile bullet = new ProjectilePlayerBullet(getCenterX(), getCenterY(),
                        InputUtility.instance.getMouseX(), InputUtility.instance.getMouseY());
                GameManager.addEntity(bullet);
                shootingTimer = 0;
            }
        }
    }

    private void updateHealthRegeneration() {
        if (healthRegenerationTimer < HEALTH_REGEN_DELAY) {
            healthRegenerationTimer++;
        } else {
            hp += healthRegenerationRate;
            if (hp > maxHp) {
                hp = maxHp;
            }
            healthRegenerationTimer = 0;
        }
    }

    private void updateHarvest() {
        if (InputUtility.instance.isMouseRightDown()) {
            int x = TileManager.getMouseTileX();
            int y = TileManager.getMouseTileY();
            if (!TileManager.isOutOfBound(x, y)) {
                TileObject object = TileManager.instance.tileArray[x][y].getTileObject();
                if (object != null) {
                    object.reduceHP(HARVEST_POWER);
                }
            }
        }
    }

    @Override
    public void update() {
        updateVelocity();
        super.update();
        updateShoot();
        updateHealthRegeneration();
        updateHarvest();
    }

    public static void setHealthRegenerationRate(int rate) {
        healthRegenerationRate = rate;
    }
}

/*src/model/enemy/EnemyBasic.java*/
/*
 * Basic enemy (weak)
 */
package model.enemy;

import javafx.scene.canvas.GraphicsContext;
import model.RenderableHolder;

public class EnemyBasic extends Enemy {

    private static final double SPEED = 2;

    public EnemyBasic(double x, double y, int level) {
        super(x, y, SPEED, 20 + 20 * level, (int) (3 + 0.7 * level), 3 * level);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.enemy_basic_img);
    }
}

/*src/model/enemy/Enemy.java*/
/*
 * Base class for enemy
 */
package model.enemy;

import logic.CollisionUtility;
import logic.GameManager;
import logic.ICollidable;
import logic.IRenderable;
import logic.ResourceManager;
import model.BlockingEntity;
import model.Entity;
import model.RenderableHolder;
import model.tileObject.TileObjectRocket;

public abstract class Enemy extends BlockingEntity {

    private static final double WIDTH = 20;
    private static final double HEIGHT = 20;

    private int damage;
    private int attackTimer;
    private int reward;
    private static final int ATTACK_DELAY = 20;
    private static final double ATTACK_RANGE = 5;

    private Entity target;

    public Enemy(double x, double y, double speed, int startHp, int damage, int reward) {
        super(x, y, WIDTH, HEIGHT, speed, startHp);
        this.damage = damage;
        this.reward = reward;
    }

    private Entity findTarget(Class targetClass) {
        double minDist = 0;
        Entity target = null;
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof Entity && !((Entity) ir).isDestroy()) {
                if (targetClass.isAssignableFrom(ir.getClass())) {
                    double dist = CollisionUtility.findDistance(this, (ICollidable) ir);
                    if (target == null || dist < minDist) {
                        target = (Entity) ir;
                        minDist = dist;
                    }
                }
            }
        }
        return target;
    }

    @Override
    public void update() {
        super.update();

        if (target != null && target.isDestroy())
            target = null;

        if (target == null)
            target = findTarget(TileObjectRocket.class);
        if (target == null)
            target = GameManager.instance.getPlayer();

        if (target != null) {
            double dx = target.getX() - x;
            double dy = target.getY() - y;
            double angle = Math.atan2(dy, dx);
            this.velX = Math.cos(angle);
            this.velY = Math.sin(angle);
        }

        attackTimer++;
        if (attackTimer >= ATTACK_DELAY) {
            attack();
            attackTimer = 0;
        }

    }

    private void attack() {
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof Entity) {
                if (!(ir instanceof Enemy)) {
                    double dx = Math.abs(((Entity) ir).getCenterX() - getCenterX());
                    double dy = Math.abs(((Entity) ir).getCenterY() - getCenterY());
                    dx -= getWidth() / 2 + ((Entity) ir).getWidth() / 2;
                    dy -= getHeight() / 2 + ((Entity) ir).getHeight() / 2;
                    if (dx <= ATTACK_RANGE && dy <= ATTACK_RANGE) {
                        ((Entity) ir).reduceHP(damage);
                    }
                }
            }
        }
    }

    @Override
    public void undoMove() {
        x = lastX;
        y = lastY;
    }

    @Override
    public int getZ() {
        return 15;
    }

    public void setTarget(Entity target) {
        this.target = target;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        ResourceManager.instance.addResource(ResourceManager.ARTIFACT, reward);
        GameManager.instance.increaseScore(reward);
    }

}

/*src/model/enemy/EnemyBoss.java*/
/*
```

```
 * Enemy boss
 */
package model.enemy;

import javafx.scene.canvas.GraphicsContext;
import model.RenderableHolder;

public class EnemyBoss extends Enemy {

    private static final double SPEED = 3;

    public EnemyBoss(double x, double y, int level) {
        super(x, y, SPEED, 50 + 40 * level, (int) (4 + 0.8 * level), 10 * level);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.enemy_boss_img);
    }
}

/*src/model/projectile/ProjectileBullet.java*/
/*
 * Bullet (from Turret tower)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import model.Entity;
import model.RenderableHolder;

public class ProjectileBullet extends Projectile {

    private static final double WIDTH = 20;
    private static final double HEIGHT = 8;
    private static final double SPEED = 12;
    private static final int DAMAGE = 40;

    public ProjectileBullet(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, targetX, targetY);
    }

    public ProjectileBullet(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_bullet_img);
    }

}

/*src/model/projectile/ProjectileBomb.java*/
/*
 * Bomb (From bomb tower)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import logic.CollisionUtility;
import logic.ICollidable;
import logic.IRenderable;
import model.Entity;
import model.RenderableHolder;
import model.enemy.Enemy;

public class ProjectileBomb extends Projectile {
    private static final double WIDTH = 30;
    private static final double HEIGHT = 30;
    private static final double SPEED = 3;
    private static final int DAMAGE = 30;

    private static final double EXPLOSIVE_RANGE = 120;
    private static final int EXPLOSIVE_DAMAGE = 60;

    public ProjectileBomb(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, targetX, targetY);
    }

    public ProjectileBomb(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, target);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof Entity) {
                if (ir instanceof Enemy && CollisionUtility.findDistance(this, (ICollidable) ir) <= EXPLOSIVE_RANGE) {
                    ((Entity) ir).reduceHP(EXPLOSIVE_DAMAGE);
                }
            }
        }
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_bomb_img);
    }

}

/*src/model/projectile/ProjectileArrow.java*/
/*
 * Arrow (From arrow tower)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import model.Entity;
import model.RenderableHolder;

public class ProjectileArrow extends Projectile {
    private static final double WIDTH = 20;
    private static final double HEIGHT = 8;
    private static final double SPEED = 8;
    private static final int DAMAGE = 5;

    public ProjectileArrow(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, targetX, targetY);
    }

    public ProjectileArrow(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_arrow_img);
    }

}

/*src/model/projectile/ProjectileBulletSniper.java*/
/*
 * Sniper bullet (from Sniper tower)
 */
package model.projectile;

import model.Entity;

public class ProjectileBulletSniper extends ProjectileBullet {

    private static final double WIDTH = 40;
    private static final double HEIGHT = 8;
    private static final double SPEED = 20;
    private static final int DAMAGE = 200;

    public ProjectileBulletSniper(double x, double y, double targetX, double targetY) {
        super(x, y, targetX, targetY);
        this.width = WIDTH;
        this.height = HEIGHT;
        this.speed = SPEED;
        this.damage = DAMAGE;
    }

    public ProjectileBulletSniper(double x, double y, Entity target) {
        super(x, y, target);
        this.width = WIDTH;
        this.height = HEIGHT;
        this.speed = SPEED;
        this.damage = DAMAGE;
    }

}

/*src/model/projectile/Projectile.java*/
/*
 * Base class for projectile
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import javafx.scene.transform.Affine;
import javafx.scene.transform.Rotate;
import logic.ICollidable;
import model.Entity;
import model.MovingEntity;
import model.enemy.Enemy;
import model.tileObject.TileObjectVoid;

public abstract class Projectile extends MovingEntity {

    protected int damage;
    private double angle;

    private double originalWidth, originalHeight;

    private Entity target = null;

    // Every projectile should be going to the target
    public Projectile(double x, double y, double width, double height, double speed, int damage, double targetX,
            double targetY) {
        super(x - width / 2, y - height / 2, width, height, speed, 100);
        this.damage = damage;
        this.originalWidth = width;
        this.originalHeight = height;
        setTarget(targetX, targetY);
    }

    public Projectile(double x, double y, double width, double height, double speed, int damage, Entity target) {
        this(x, y, width, height, speed, damage, target.getX(), target.getY());
        this.target = target;
    }

    private void setTarget(double tx, double ty) {
        double dx = tx - x;
        double dy = ty - y;
        this.angle = Math.atan2(dy, dx);
        this.velX = Math.cos(angle);
        this.velY = Math.sin(angle);
        // modify my hitbox
        this.width = Math.max(Math.abs(Math.sin(angle) * this.originalHeight),
                Math.abs(Math.cos(angle) * this.originalWidth));

        this.height = Math.max(Math.abs(Math.cos(angle) * this.originalHeight),
                Math.abs(Math.sin(angle) * this.originalWidth));
    }

    @Override
    public void update() {
        super.update();
        if (target != null && target.isDestroy())
            target = null;
        if (target != null) {
            setTarget(target.getCenterX(), target.getCenterY());
        }
    }

    @Override
    public int getZ() {
        return 20;
    }

    @Override
    public void onCollision(ICollidable collider) {
        if (this.isDestroyed)
            return;
        else if (collider instanceof Projectile)
            return; // projectiles are not suppose to hit each other
        else if (collider instanceof TileObjectVoid)
            this.destroy();
        else if (collider instanceof Enemy) {
            this.destroy();
            ((Entity) collider).reduceHP(this.damage);
        }
    }

    public void draw(GraphicsContext gc, Image img) {
        // draw with rotation
        Affine old = gc.getTransform().clone();
        Affine rotated = old.clone();
        rotated.append(new Rotate(angle / Math.PI * 180, x + originalWidth / 2, y + originalHeight / 2));
        gc.setTransform(rotated);
        gc.drawImage(img, x, y, originalWidth, originalHeight);
        gc.setTransform(old);
    }

    @Override
    public void reduceHP(int damage) {

    }

}

/*src/model/projectile/ProjectilePlayerBullet.java*/
/*
 * Player s bullet (Damage can be modified from research)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import model.Entity;
import model.RenderableHolder;

public class ProjectilePlayerBullet extends Projectile {

    private static final double WIDTH = 20;
    private static final double HEIGHT = 8;
    private static final double SPEED = 12;
    private static int damage = 15;

    public ProjectilePlayerBullet(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, damage, targetX, targetY);
    }

    public ProjectilePlayerBullet(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, damage, target);
    }
    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_bullet_img);
    }

    public static void addDamage(int d) {
        damage += d;
    }

    public static int getDamage() {
        return damage;
    }
}

/*src/model/projectile/ProjectileLaser.java*/
/*
 * Laser (from laser tower)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import model.Entity;
import model.RenderableHolder;

public class ProjectileLaser extends Projectile {

    private static final double WIDTH = 60;
    private static final double HEIGHT = 7;
    private static final double SPEED = 20;
    private static final int DAMAGE = 6;

    public ProjectileLaser(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, targetX, targetY);
    }

    public ProjectileLaser(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_laser_img);
    }

}

/*src/model/projectile/ProjectileRock.java*/
/*
 * Rock (From catapult)
 */
package model.projectile;

import javafx.scene.canvas.GraphicsContext;
import model.Entity;
import model.RenderableHolder;

public class ProjectileRock extends Projectile {
    private static final double WIDTH = 30;
    private static final double HEIGHT = 30;
    private static final double SPEED = 5;
    private static final int DAMAGE = 30;

    public ProjectileRock(double x, double y, double targetX, double targetY) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, targetX, targetY);
    }

    public ProjectileRock(double x, double y, Entity target) {
        super(x, y, WIDTH, HEIGHT, SPEED, DAMAGE, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.projectile_rock_img);
    }

}

/*src/model/tileObject/TileObjectWallStone.java*/
/*
 * Stone wall
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectWallStone extends TileObject {

    private static final int START_HP = 700;
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;

    public TileObjectWallStone(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_wall_stone_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 0, 2, 0, 0, 0 };
    }

}

/*src/model/tileObject/TileObjectTree.java*/
/*
 * Tree (Give 2 woods when destroyed)
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectTree extends TileObject {
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 100;

    public TileObjectTree(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP); // 1x2
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        ResourceManager.instance.addResource(ResourceManager.WOOD, 2);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_tree_img);
    }

}

/*src/model/tileObject/TileObjectVoid.java*/
/*
 * TileObjectVoid is place around the outside of the screen so entity won t go out
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.GameManager;
import model.Tile;

public class TileObjectVoid extends TileObject {
    private static final int SIZE_X = 1;
    private static final int SIZE_Y = 1;
    private static final int START_HP = -1;

    public TileObjectVoid(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP); // 1x1
    }

    @Override
    public void draw(GraphicsContext gc) {

    }

    @Override
    public void drawHealthBar(GraphicsContext gc) {

    }

    @Override
    public void update() {
        // TileObjectVoid don't die
    }

    @Override
    public void place(Tile tile) {
        GameManager.addEntity(this);
    }

}

/*src/model/tileObject/TileObjectWallIron.java*/
```

```
/*
 * Iron wall
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectWallIron extends TileObject {

    private static final int START_HP = 1000;
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;

    public TileObjectWallIron(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_wall_iron_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 0, 0, 2, 0, 0 };
    }

}

/*src/model/tileObject/TileObjectWallWood.java*/
/*
 * Wood wall
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectWallWood extends TileObject {

    private static final int START_HP = 300;
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;

    public TileObjectWallWood(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_wall_wood_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 2, 0, 0, 0, 0 };
    }

}

/*src/model/tileObject/TileObjectStone.java*/
/*
 * Stone (Give 3 stones when destroy)
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectStone extends TileObject {
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 250;

    public TileObjectStone(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP); // 2x2
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        ResourceManager.instance.addResource(ResourceManager.STONE, 3);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_stone_img);
    }

}

/*src/model/tileObject/TileObjectRocket.java*/
/*
 * Rocket silo (When placed, it will countdown from 30 sec to 0 sec and then win)
 */
package model.tileObject;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import logic.GameManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class TileObjectRocket extends TileObject {

    private static final int START_HP = 1500;
    public static final int SIZE_X = 3;
    public static final int SIZE_Y = 3;

    private static final int LAUNCH_DELAY = 1800;
    private int launchTimer = 0;

    public TileObjectRocket(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP);
        GameManager.instance.setRocketCount(GameManager.instance.getRocketCount() + 1);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        GameManager.instance.setRocketCount(GameManager.instance.getRocketCount() - 1);
    }

    @Override
    public void update() {
        super.update();
        if (launchTimer < LAUNCH_DELAY) {
            launchTimer++;
        } else {
            GameManager.instance.setRocketLaunched(true);
        }
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tileObject_rocket_img);
        gc.setFill(Color.GREEN);
        gc.setFont(Font.font(20));
        gc.fillText("" + (int) ((LAUNCH_DELAY - launchTimer) / 60), this.getCenterX() - 10, this.getCenterY());
    }

    public static int[] getResourceNeeded() {
        return new int[] { 100, 100, 100, 100, 9999 };
    }

}

/*src/model/tileObject/TileObject.java*/
/*
 * An object that is fix to a tile
 */
package model.tileObject;

import java.util.ArrayList;
import java.util.List;

import logic.GameManager;
import logic.IBlockable;
import logic.TileManager;
import model.Entity;
import model.Tile;

/* Tile Object is an object that is fix to a tile (object which can't be moved) */
public abstract class TileObject extends Entity implements IBlockable {

    protected List<Tile> tile; // can take multiple tile
    public int sizeX;
    public int sizeY;

    public TileObject(Tile tile, int sizeX, int sizeY, int hp) {
        super(tile.getX(), tile.getY(), TileManager.TILE_SIZE * sizeX, TileManager.TILE_SIZE * sizeY, hp);
        this.tile = new ArrayList<Tile>();
        this.sizeX = sizeX;
        this.sizeY = sizeY;
        this.place(tile);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // remove link from tile
        for (Tile t : tile) {
            t.setTileObject(null);
        }
    }

    @Override
    public int getZ() {
        return 10;
    }

    @Override
    public void undoMove() {
        //
    }

    public void place(Tile tile) {
        for (int dx = 0; dx < this.sizeX; dx++) {
            for (int dy = 0; dy < this.sizeY; dy++) {
                int x = tile.getTileX() + dx;
                int y = tile.getTileY() + dy;
                TileManager.instance.tileArray[x][y].setTileObject(this);
                this.tile.add(TileManager.instance.tileArray[x][y]);
            }
        }
        GameManager.addEntity(this);
    }
}

/*src/model/tileObject/generator/Generator.java*/
/*
 * Base class for generator
 */
package model.tileObject.generator;

import model.Tile;
import model.tileObject.TileObject;

public abstract class Generator extends TileObject {

    private int resource;
    private int amount;
    private int resourceGenerateTimer = 0;
    private int resourceGenerateDelay;

    public Generator(Tile tile, int sizeX, int sizeY, int hp, int resource, int delay, int amount) {
        super(tile, sizeX, sizeY, hp);
        this.resource = resource;
        this.resourceGenerateDelay = delay;
        this.amount = amount;
    }

    public void update() {
        super.update();
        if (resourceGenerateTimer < resourceGenerateDelay) {
            resourceGenerateTimer++;
        } else {
            logic.ResourceManager.instance.addResource(resource, amount);
            resourceGenerateTimer = 0;
        }
    }

}

/*src/model/tileObject/generator/GeneratorWood.java*/
/*
 * Wood generator
 */
package model.tileObject.generator;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class GeneratorWood extends Generator {
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 250;

    private static final int RESOURCE = ResourceManager.WOOD;
    private static final int DELAY = 60;
    private static final int AMOUNT = 1;

    public GeneratorWood(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, DELAY, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.generator_wood_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 10, 0, 0, 0, 1 };
    }

}

/*src/model/tileObject/generator/GeneratorStone.java*/
/*
 * Stone generator
 */
package model.tileObject.generator;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class GeneratorStone extends Generator {
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 500;

    private static final int RESOURCE = ResourceManager.STONE;
    private static final int DELAY = 60;
    private static final int AMOUNT = 1;

    public GeneratorStone(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, DELAY, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.generator_stone_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 10, 10, 0, 0, 10 };
    }

}

/*src/model/tileObject/generator/GeneratorDiamond.java*/
/*
 * Diamond generator
 */
package model.tileObject.generator;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class GeneratorDiamond extends Generator {
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 1000;

    private static final int RESOURCE = ResourceManager.DIAMOND;
    private static final int DELAY = 180;
    private static final int AMOUNT = 1;

    public GeneratorDiamond(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, DELAY, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.generator_diamond_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 10, 10, 10, 10, 500 };
    }

}
```

```
/*src/model/tileObject/generator/GeneratorIron.java*/
/*
 * Iron generator
 */
package model.tileObject.generator;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class GeneratorIron extends Generator {
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    private static final int START_HP = 750;

    private static final int RESOURCE = ResourceManager.IRON;
    private static final int DELAY = 120;
    private static final int AMOUNT = 1;

    public GeneratorIron(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, DELAY, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.generator_iron_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 10, 10, 10, 0, 100 };
    }

}

/*src/model/tileObject/storage/StorageWood.java*/
/*
 * Storage for wood
 */
package model.tileObject.storage;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class StorageWood extends Storage {
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;
    private static final int START_HP = 100;

    private static final int RESOURCE = ResourceManager.WOOD;
    private static final int AMOUNT = 15;

    public StorageWood(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.storage_wood_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 5, 0, 0, 0, 0 };
    }

}

/*src/model/tileObject/storage/StorageIron.java*/
/*
 * Storage for iron
 */
package model.tileObject.storage;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class StorageIron extends Storage {
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;
    private static final int START_HP = 300;

    private static final int RESOURCE = ResourceManager.IRON;
    private static final int AMOUNT = 15;

    public StorageIron(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.storage_iron_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 5, 0, 5, 0, 0 };
    }

}

/*src/model/tileObject/storage/StorageStone.java*/
/*
 * Storage for stone
 */
package model.tileObject.storage;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class StorageStone extends Storage {
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;
    private static final int START_HP = 200;

    private static final int RESOURCE = ResourceManager.STONE;
    private static final int AMOUNT = 15;

    public StorageStone(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.storage_stone_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 5, 5, 0, 0, 0 };
    }

}

/*src/model/tileObject/storage/Storage.java*/
/*
 * Base class for storage
 */
package model.tileObject.storage;

import logic.ResourceManager;
import model.Tile;
import model.tileObject.TileObject;

public abstract class Storage extends TileObject {

    private int resource;
    private int amount;

    public Storage(Tile tile, int sizeX, int sizeY, int hp, int resource, int amount) {
        super(tile, sizeX, sizeY, hp);
        this.resource = resource;
        this.amount = amount;
        ResourceManager.instance.addCapacity(resource, amount);
    }

    public void onDestroy() {
        super.onDestroy();
        ResourceManager.instance.addCapacity(resource, -amount);
    }

}

/*src/model/tileObject/storage/StorageDiamond.java*/
/*
 * Storage for diamond
 */
package model.tileObject.storage;

import javafx.scene.canvas.GraphicsContext;
import logic.ResourceManager;
import logic.TileManager;
import model.RenderableHolder;
import model.Tile;

public class StorageDiamond extends Storage {
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;
    private static final int START_HP = 500;

    private static final int RESOURCE = ResourceManager.DIAMOND;

    private static final int AMOUNT = 15;

    public StorageDiamond(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, RESOURCE, AMOUNT);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.storage_diamond_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 5, 0, 0, 5, 0 };
    }

}

/*src/model/tileObject/tower/TowerTurret.java*/
/*
 * Turret tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileBullet;

public class TowerTurret extends Tower {

    private static final int START_HP = 500;
    private static final int SHOOTING_DELAY = 30;
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 2;
    public static final int SHOOTING_RANGE = 250;

    public TowerTurret(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileBullet(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_turret_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 0, 5, 3, 0, 0 };
    }
}

/*src/model/tileObject/tower/TowerBomb.java*/
/*
 * Bomb tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileBomb;

public class TowerBomb extends Tower {

    private static final int START_HP = 800;
    private static final int SHOOTING_DELAY = 60;
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    public static final int SHOOTING_RANGE = 300;


    public TowerBomb(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileBomb(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_bomb_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 0, 2, 8, 0, 0 };
    }
}

/*src/model/tileObject/tower/TowerSniper.java*/
/*
 * Sniper tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileBulletSniper;

public class TowerSniper extends Tower {

    private static final int START_HP = 500;
    private static final int SHOOTING_DELAY = 180;
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    public static final int SHOOTING_RANGE = 700;

    public TowerSniper(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileBulletSniper(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_sniper_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 2, 2, 8, 2, 0 };
    }
}

/*src/model/tileObject/tower/TowerLaser.java*/
/*
 * Laser tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileLaser;

public class TowerLaser extends Tower {

    private static final int START_HP = 400;
    private static final int SHOOTING_DELAY = 3;
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 2;
    public static final int SHOOTING_RANGE = 300;

    public TowerLaser(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }
    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileLaser(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_laser_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 0, 0, 6, 3, 0 };
    }
}

/*src/model/tileObject/tower/TowerCatapult.java*/
/*
 * Catapult tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileRock;

public class TowerCatapult extends Tower {

    private static final int START_HP = 250;
    private static final int SHOOTING_DELAY = 60;
    public static final int SIZE_X = 2;
    public static final int SIZE_Y = 1;
    public static final int SHOOTING_RANGE = 400;

    public TowerCatapult(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileRock(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_catapult_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 5, 5, 0, 0, 0 };
    }
}

/*src/model/tileObject/tower/Tower.java*/
/*
 * Base class for tower
 */
package model.tileObject.tower;

import logic.CollisionUtility;
import logic.ICollidable;
import logic.IRenderable;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.enemy.Enemy;
import model.projectile.Projectile;
import model.tileObject.TileObject;

public abstract class Tower extends TileObject {

    private int shootingTimer = 0;
    private int shootingDelay = 0;
    private double shootingRange;

    public Tower(Tile tile, int sizeX, int sizeY, int hp, int shootingDelay, double shootingRange) {
        super(tile, sizeX, sizeY, hp);
        this.shootingDelay = shootingDelay;
        this.shootingRange = shootingRange;
    }

    @Override
    public void update() {
        super.update();

        shootingTimer++;
        if (shootingTimer >= shootingDelay) {
            shoot();
            shootingTimer = 0;
        }
    }

    protected abstract Projectile createProjectile(double x, double y, Entity target);

    private void shoot() {
        Enemy target = null;
        double targetDist = Double.MAX_VALUE;
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof Enemy) {
                double dist = CollisionUtility.findDistance(this, (ICollidable) ir);
                if (dist < targetDist) {
                    targetDist = dist;
                    target = (Enemy) ir;
                }
            }
        }

        if (target != null && targetDist <= shootingRange) {
            Projectile projectile = createProjectile(this.x + this.width / 2, this.y + this.height / 2, target);
            RenderableHolder.instance.add(projectile);
        }
    }

}

/*src/model/tileObject/tower/TowerArrow.java*/
/*
 * Arrow tower
 */
package model.tileObject.tower;

import javafx.scene.canvas.GraphicsContext;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.projectile.Projectile;
import model.projectile.ProjectileArrow;

public class TowerArrow extends Tower {

    private static final int START_HP = 200;
    private static final int SHOOTING_DELAY = 20;
    public static final int SIZE_X = 1;
    public static final int SIZE_Y = 1;
    public static final int SHOOTING_RANGE = 150;


    public TowerArrow(Tile tile) {
        super(tile, SIZE_X, SIZE_Y, START_HP, SHOOTING_DELAY, SHOOTING_RANGE);
    }

    public static boolean canPlace(Tile tile) {
        return TileManager.instance.canPlace(tile, SIZE_X, SIZE_Y);
    }

    @Override
    protected Projectile createProjectile(double x, double y, Entity target) {
        return new ProjectileArrow(x, y, target);
    }

    @Override
    public void draw(GraphicsContext gc) {
        super.draw(gc, RenderableHolder.tower_arrow_img);
    }

    public static int[] getResourceNeeded() {
        return new int[] { 3, 0, 0, 0, 0 };
    }
}

/*src/thread/ThreadShowHighscore.java*/
/*
 * Get high score from server (blocking) and display it
 */
package thread;

import java.util.ArrayList;
import java.util.Collections;

import exception.HighscoreException;
import javafx.application.Platform;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import logic.HighscoreManager;
import model.ScoreRecord;

public class ThreadShowHighscore extends Thread {
    @Override
    public void run() {
        String rawscore;
        try {
            rawscore = HighscoreManager.getScore();
            ArrayList<ScoreRecord> scores = new ArrayList<>();
            for (String scoreline : rawscore.split("\n")) {
                String[] result = scoreline.split(" ");
                int score = Integer.parseInt(result[0]);
                String name = result[1];
                scores.add(new ScoreRecord(score, name));
            }
            Collections.sort(scores);
            String scoreText = "";
            for (int i = 0; i < Math.min(10, scores.size()); i++) {
                scoreText += scores.get(i).toString();
                scoreText += "\n";
            }
            final String copyOfScoreText = scoreText;
            Platform.runLater(() -> {
                Alert alert = new Alert(AlertType.INFORMATION);
                alert.setContentText(copyOfScoreText);
                alert.show();
            });
        } catch (HighscoreException e) {
            Platform.runLater(() -> {
                Alert alert = new Alert(AlertType.ERROR);
                alert.setHeaderText("showHighscore Error");
                alert.setContentText(e.getMessage());
                alert.show();
            });
        }
    }

}

/*src/thread/ThreadGameManager.java*/
/*
 * Thread for updating logic / game state (call GameManager update)
 */
package thread;

import application.Main;
import logic.GameManager;
import ui.GamePane;

public class ThreadGameManager extends Thread {

    public ThreadGameManager() {
        super("Game Manager Thread");
    }

    public void run() {
        try {
            while (!interrupted() && Main.getScene().getRoot() instanceof GamePane) {
                Thread.sleep(16);
                GameManager.instance.update();
            }
        } catch (InterruptedException e) {

        }
    }
}

/*src/thread/ThreadNewScore.java*/
/*
 * Put name and score to the high score server (blocking)
 */
package thread;

import exception.HighscoreException;
import javafx.application.Platform;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import logic.HighscoreManager;

public class ThreadNewScore extends Thread {
    private int score;
    private String name;

    public ThreadNewScore(String name, int score) {
        this.score = score;
        this.name = name;
    }

    public void run() {
        try {
            HighscoreManager.postScore(name, score);
        } catch (HighscoreException e) {
            Platform.runLater(() -> {
                Alert alert = new Alert(AlertType.ERROR);
                alert.setHeaderText("newScore Error");
                alert.setContentText(e.getMessage());
                alert.show();
            });
        }
    }
}

/*src/ui/IStoppable.java*/
/*
 * Stop unfinished job before get destroyed
 */
package ui;

public interface IStoppable {
    public void stop();
}

/*src/ui/StatsBox.java*/
/*
 * Display game stats
 */
package ui;

import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import logic.EnemyManager;
import logic.GameManager;
import logic.TileManager;
import model.RenderableHolder;
import model.projectile.ProjectilePlayerBullet;

public class StatsBox extends VBox {
    private Label score, fps, entityCount, playerDamage, wave;

    public StatsBox() {
        score = new Label();
        fps = new Label();
        entityCount = new Label();
        playerDamage = new Label();
        wave = new Label();
        getChildren().addAll(score, fps, wave);
    }

    public void update() {
        score.setText("Score " + GameManager.instance.getScore());
        fps.setText(GameManager.instance.getFps() + "fps");
        playerDamage.setText("Damage " + ProjectilePlayerBullet.getDamage());
        wave.setText("Wave " + EnemyManager.instance.getWaveNumber() + " , Next in "
                + (EnemyManager.instance.getRemainingTime() / 60) + " sec " + "("
                + EnemyManager.instance.getNextWaveName() + ")");
        int tileCount = (TileManager.TILE_COUNT_X + 2) * (TileManager.TILE_COUNT_Y + 2);
        entityCount.setText("Entity Count : " + (RenderableHolder.instance.getEntities().size() - tileCount));
    }
}

/*src/ui/ResearchBox.java*/
/*
 * Unlock ability/new resource
 */
package ui;

import javafx.scene.layout.GridPane;
import ui.research.BuildingResearch;
import ui.research.DiamondResearch;
import ui.research.GunResearch;
import ui.research.HealthRegenerationResearch;
import ui.research.IronResearch;
import ui.research.SmallGunResearch;
import ui.research.SuperGunResearch;

public class ResearchBox extends GridPane {

    public ResearchBox() {
        add(new BuildingResearch(), 0, 0);
        add(new IronResearch(), 1, 0);
        add(new DiamondResearch(), 2, 0);
        add(new HealthRegenerationResearch(), 3, 0);
        add(new SmallGunResearch(), 0, 1);
        add(new GunResearch(), 1, 1);
        add(new SuperGunResearch(), 2, 1);
    }

}

/*src/ui/VolumePane.java*/
/*
 * Control game volume
 */
package ui;

import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Slider;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import logic.SoundManager;
import model.RenderableHolder;

public class VolumePane extends HBox {
    Slider sl;

    public VolumePane() {
        setAlignment(Pos.CENTER_RIGHT);
        this.setPadding(new Insets(0, 100, 0, 0));

        ImageView iv = new ImageView(RenderableHolder.volume_img);
        sl = new Slider(0, 1, SoundManager.getVolume());

        iv.setFitWidth(64);
        iv.setFitHeight(64);

        getChildren().addAll(iv, sl);
    }

    public double getVolume() {
        return sl.getValue();
    }
}

/*src/ui/BuyBox.java*/
/*
 * Buy and place item on the map
 */
package ui;

import javafx.scene.control.ListView;
import model.*;
import model.tileObject.TileObjectRocket;
import model.tileObject.TileObjectWallIron;
import model.tileObject.TileObjectWallStone;
import model.tileObject.TileObjectWallWood;
import model.tileObject.generator.*;
import model.tileObject.storage.*;
import model.tileObject.tower.TowerArrow;
import model.tileObject.tower.TowerBomb;
import model.tileObject.tower.TowerCatapult;
import model.tileObject.tower.TowerLaser;
import model.tileObject.tower.TowerSniper;
import model.tileObject.tower.TowerTurret;

public class BuyBox extends ListView<BuyItem> {
    public BuyBox() {
        getItems().add(new BuyItem("Arrow", RenderableHolder.tower_arrow_img, TowerArrow.class));
        getItems().add(new BuyItem("Catapult", RenderableHolder.tower_catapult_img, TowerCatapult.class));
        getItems().add(new BuyItem("Turret", RenderableHolder.tower_turret_img, TowerTurret.class));
        getItems().add(new BuyItem("Bomb", RenderableHolder.tower_bomb_img, TowerBomb.class));
        getItems().add(new BuyItem("Laser", RenderableHolder.tower_laser_img, TowerLaser.class));
        getItems().add(new BuyItem("Sniper", RenderableHolder.tower_sniper_img, TowerSniper.class));

        getItems().add(new BuyItem("Wood Wall", RenderableHolder.tileObject_wall_wood_img, TileObjectWallWood.class));
        getItems()
                .add(new BuyItem("Stone Wall", RenderableHolder.tileObject_wall_stone_img, TileObjectWallStone.class));
        getItems().add(new BuyItem("Iron Wall", RenderableHolder.tileObject_wall_iron_img, TileObjectWallIron.class));

        getItems().add(new BuyItem("Wood Generator", RenderableHolder.generator_wood_img, GeneratorWood.class));
        getItems().add(new BuyItem("Stone Generator", RenderableHolder.generator_stone_img, GeneratorStone.class));
        getItems().add(new BuyItem("Iron Generator", RenderableHolder.generator_iron_img, GeneratorIron.class));
        getItems()
                .add(new BuyItem("Diamond Generator", RenderableHolder.generator_diamond_img, GeneratorDiamond.class));

        getItems().add(new BuyItem("Wood Storage", RenderableHolder.storage_wood_img, StorageWood.class));
        getItems().add(new BuyItem("Stone Storage", RenderableHolder.storage_stone_img, StorageStone.class));
        getItems().add(new BuyItem("Iron Storage", RenderableHolder.storage_iron_img, StorageIron.class));
        getItems().add(new BuyItem("Diamond Storage", RenderableHolder.storage_diamond_img, StorageDiamond.class));

        getItems().add(new BuyItem("Rocket", RenderableHolder.tileObject_rocket_img, TileObjectRocket.class));
    }
}

/*src/ui/BuyItem.java*/
/*
 * Buy item T (Class) - Tower,Generator,..etc
 */
package ui;

import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import logic.BuyManager;
import model.RenderableHolder;

public class BuyItem extends HBox {
    public BuyItem(String _name, Image img, Class T) {
        ImageView iv = new ImageView(img);
        VBox vb = new VBox();
        Label name = new Label();
        HBox cost = new HBox();

        iv.setFitWidth(32);
        iv.setFitHeight(32);
        name.setText(_name);

        int[] resourceNeeded = null;
        try {
            resourceNeeded = (int[]) T.getMethod("getResourceNeeded").invoke(null);
        } catch (Exception e) {
            e.printStackTrace();
        }
        for (int i = 0; i < 5; i++) {
            Label needed = new Label("  " + resourceNeeded[i]);
            ImageView pic = new ImageView(RenderableHolder.resource_img[i]);
            pic.setFitWidth(16);
            pic.setFitHeight(16);

            cost.getChildren().addAll(needed, pic);
        }

        vb.getChildren().addAll(name, cost);
        getChildren().addAll(iv, vb);

        setOnMouseClicked(e -> {
            BuyManager.instance.isBuyMode = true;
            BuyManager.instance.currentObjectImage = img;
            BuyManager.instance.currentObjectClass = T;
        });
    }

}

/*src/ui/GamePane.java*/
/*
 * Hold GameScreen,GameMenu and GameManager
 */
package ui;

import javafx.animation.AnimationTimer;
import javafx.scene.layout.HBox;
import logic.GameManager;

public class GamePane extends HBox implements IStoppable {
    AnimationTimer at;

    public GamePane(int width, int height) {
        // 300 px for game menu
        GameScreen gs = new GameScreen(width - 300, height);
        GameMenu menu = new GameMenu();
        getChildren().add(gs);
        getChildren().add(menu);

        GameManager.instance = new GameManager();

        at = new AnimationTimer() {
            Long start = 0l;

            @Override
            public void handle(long now) {
                if (start == 0l)
                    start = now;
                long diff = now - start;
                if (diff > 0)
                    GameManager.instance.setFps((int) (1000000000l / (diff)));

                if (diff >= 10000000l) {
                    //GameManager.instance.update();
                    gs.paintComponents();
                    menu.update();
                    start = now;
                }

            }
        };
        at.start();
        GameManager.instance.startUpdateThread();
    }

    public void stop() {
        at.stop();
        GameManager.instance.stopUpdateThread();
    }
}

/*src/ui/ResourceItem.java*/
/*
 *  ResourceItem index corresponding to ResourceManager index
 */
package ui;

import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressBar;
import javafx.scene.layout.HBox;
import logic.ResourceManager;

public class ResourceItem extends HBox {
    private ProgressBar pb;
    private Label lb;
    private int index;
    public ResourceItem(int index) {
        pb=new ProgressBar(-1);
        lb=new Label();
        this.index=index;
        Button addResource=new Button("+");
        addResource.setOnAction(e->{
            if(ResourceManager.instance.getResource(ResourceManager.ARTIFACT) >= ResourceManager.EXCHANGE_RATE[index]){
                ResourceManager.instance.addResource(index,1);
                ResourceManager.instance.addResource(ResourceManager.ARTIFACT,-ResourceManager.EXCHANGE_RATE[index]);
            }
        });
        getChildren().add(pb);
        if(index!=4){
            // no self exchange for artifact
            getChildren().add(addResource);
        }
        getChildren().add(lb);
    }
    public void update(){
        int res=ResourceManager.instance.getResource(index);
        int cap=ResourceManager.instance.getCapacity(index);
        lb.setText(ResourceManager.NAME[index]+" "+res+"/"+cap);
        if(cap==0){
            pb.setDisable(true);
            pb.setProgress(-1);
        }
        else{
            pb.setDisable(false);
            pb.setProgress((double)res/cap);
        }
    }
}

/*src/ui/MainPane.java*/
/*
 * Main menu for this application
 */
package ui;

import application.Main;
import exception.InvalidNameException;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundImage;
import javafx.scene.layout.VBox;
import logic.SoundManager;
import thread.ThreadShowHighscore;

public class MainPane extends VBox implements IStoppable {
    private static String playerName = "ProgMeth";
    Thread joiner;

    public MainPane(int screenWidth, int screenHeight) {
        Button start = new Button("Start");
        Label nameLabel = new Label("Your name");
        TextField name = new TextField(playerName);
        Button exit = new Button("Exit");
        Button highscore = new Button("High score");
        VolumePane volume = new VolumePane();
        getChildren().addAll(nameLabel, name, start, exit, highscore, volume);

        setAlignment(Pos.BOTTOM_RIGHT);
        setPrefSize(screenWidth, screenHeight);

        Image img = new Image(ClassLoader.getSystemResource("img/ui/background.png").toString());
        BackgroundImage bgi = new BackgroundImage(img, null, null, null, null);
        Background bg = new Background(bgi);
        setBackground(bg);

        nameLabel.getStyleClass().setAll("h1");
        nameLabel.setPadding(new Insets(20, 130, 10, 0));

        name.setMaxWidth(200);
        setMargin(name, new Insets(0, 100, 20, 0));

        start.setOnAction(event -> {
            SoundManager.setVolume(volume.getVolume());
            try {
                setName(name.getText());
            } catch (InvalidNameException e) {
                Alert alert = new Alert(AlertType.ERROR);
                alert.setContentText(e.getMessage());
                alert.show();
                return;
            }
            Main.changeSceneToGame();
        });
        start.getStyleClass().setAll("btn", "btn-lg", "btn-success");
        start.setAlignment(Pos.CENTER);
        start.setPrefSize(200, 80);
        start.setStyle("-fx-cursor: hand;");
        setMargin(start, new Insets(20, 100, 20, 0));

        exit.setOnAction(e -> {
            Platform.exit();
        });
        exit.getStyleClass().setAll("btn", "btn-lg", "btn-danger");
        exit.setAlignment(Pos.CENTER);
        exit.setPrefSize(200, 80);
        exit.setStyle("-fx-cursor: hand;");
        setMargin(exit, new Insets(20, 100, 20, 0));

        highscore.setOnAction(e -> {
            highscore.setText("Loading");
            Thread t = new ThreadShowHighscore();
            t.start();
            joiner = new Thread(() -> {
                try {
                    t.join();
                    Platform.runLater(() -> {
                        highscore.setText("High score");
                    });
                } catch (InterruptedException e1) {

                }
            });
            joiner.start();
        });
        highscore.getStyleClass().setAll("btn", "btn-lg", "btn-info");
        highscore.setAlignment(Pos.CENTER);
        highscore.setPrefSize(200, 80);
        highscore.setStyle("-fx-cursor: hand;");
        setMargin(highscore, new Insets(20, 100, 20, 0));
    }

    public void stop() {
        if (joiner != null) {
            joiner.interrupt();
        }
    }

    public static String getName() {
        return playerName;
    }

    public static void setName(String name) throws InvalidNameException {
        if (name.contains(" ")) {
            throw new InvalidNameException();
        }
    }
}

/*src/ui/ResourceBox.java*/
/*
 * Display resource status
 */
package ui;

import javafx.scene.Node;
import javafx.scene.layout.VBox;

public class ResourceBox extends VBox {
    public ResourceBox() {
        for (int i = 0; i < 5; i++) {
            ResourceItem item = new ResourceItem(i);
            getChildren().add(item);
        }
    }

    public void update() {
        for (Node item : getChildren()) {
            ((ResourceItem) item).update();
        }
    }
}

/*src/ui/GameScreen.java*/
/*
 * Canvas for drawing
 */
package ui;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import logic.BuyManager;
import logic.GameManager;
import logic.IRenderable;
import logic.TileManager;
import model.Entity;
import model.RenderableHolder;
import model.Tile;
import model.tileObject.tower.Tower;

public class GameScreen extends Canvas {
    private int screen_width, screen_height;

    public GameScreen(int width, int height) {
        super(width, height);
        screen_width = width;
        screen_height = height;
    }

    public void paintComponents() {
        synchronized (RenderableHolder.instance.getEntities()) {
            GraphicsContext gc = this.getGraphicsContext2D();
            drawBackground(gc);
            drawEntities(gc);
            if (BuyManager.instance.isBuyMode) {
                drawOverlay(gc);
                drawBuyingItem(gc);
            }
            if (GameManager.instance.isGamePause()) {
                drawPause(gc);
            }
        }
    }

    private void drawPause(GraphicsContext gc) {
        gc.setGlobalAlpha(0.5);
        gc.setFill(Color.BLACK);
        gc.fillRect(0, 0, screen_width, screen_height);
        gc.setFill(Color.WHITE);
        gc.setFont(Font.font(40));
        gc.fillText("PAUSE", screen_width / 2 - 50, screen_height / 2 - 15);
        gc.setGlobalAlpha(1);
    }

    public void drawBuyingItem(GraphicsContext gc) {
        int x = TileManager.getMouseTileX();
        int y = TileManager.getMouseTileY();
        int sizeX;
        int sizeY;
        try {
            sizeX = BuyManager.instance.currentObjectClass.getDeclaredField("SIZE_X").getInt(null);
            sizeY = BuyManager.instance.currentObjectClass.getDeclaredField("SIZE_Y").getInt(null);
            // ie currentObjectClass Object is instance of Tower
            if (Tower.class.isAssignableFrom(BuyManager.instance.currentObjectClass)) {
                gc.setGlobalAlpha(0.3);
                gc.setFill(Color.BLACK);
                double r = BuyManager.instance.currentObjectClass.getDeclaredField("SHOOTING_RANGE").getDouble(null);
                gc.fillOval((x + (double) (sizeX) / 2) * TileManager.TILE_SIZE - r,
                        (y + (double) (sizeY) / 2) * TileManager.TILE_SIZE - r, 2 * r, 2 * r);
                gc.setGlobalAlpha(1);
            }
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
        boolean canPlace = BuyManager.instance.canBuy();
        if (!canPlace) {
            gc.setGlobalAlpha(0.2);
        }
        gc.drawImage(BuyManager.instance.currentObjectImage, x * TileManager.TILE_SIZE, y * TileManager.TILE_SIZE,
                sizeX * TileManager.TILE_SIZE, sizeY * TileManager.TILE_SIZE);
        gc.setGlobalAlpha(1.0);
    }

    public void drawOverlay(GraphicsContext gc) {
        gc.setGlobalAlpha(0.5);
        for (Tile tile : TileManager.instance.tileList) {
            if (tile.getTileObject() == null) {
                gc.setFill(Color.GREEN);
                gc.fillRect(tile.getX(), tile.getY(), TileManager.TILE_SIZE, TileManager.TILE_SIZE);
            } else {
                gc.setFill(Color.DARKRED);
                gc.fillRect(tile.getX(), tile.getY(), TileManager.TILE_SIZE, TileManager.TILE_SIZE);
            }
        }
        gc.setGlobalAlpha(1);
    }

    public void drawBackground(GraphicsContext gc) {
        gc.setFill(Color.BLACK);
        gc.fillRect(0, 0, screen_width, screen_height);
    }

    public void drawEntities(GraphicsContext gc) {
        for (IRenderable o : RenderableHolder.instance.getEntities()) {
            o.draw(gc);
        }
        // draw healthbar
        for (IRenderable ir : RenderableHolder.instance.getEntities()) {
            if (ir instanceof Entity) {
                ((Entity) ir).drawHealthBar(gc);
            }
        }

    }
}

/*src/ui/GameMenu.java*/
/*
 * Right game menu
 */
package ui;

import javafx.scene.layout.VBox;

public class GameMenu extends VBox {
    private StatsBox statsBox;
    private BuyBox buyBox;
    private ResourceBox resourceBox;
    private ResearchBox researchBox;

    public GameMenu() {
        this.setPrefSize(300, 600);
        buyBox = new BuyBox();
        statsBox = new StatsBox();
        resourceBox = new ResourceBox();
        researchBox = new ResearchBox();
        getChildren().addAll(statsBox,buyBox,resourceBox,researchBox);
    }

    public void update() {
        statsBox.update();
        resourceBox.update();
    }
}

/*src/ui/research/GunResearch.java*/
/*
 * Research Gun (Gun level 2)
 */
package ui.research;

import model.RenderableHolder;
import model.projectile.ProjectilePlayerBullet;

public class GunResearch extends ResearchItem {
    public GunResearch() {
        super(RenderableHolder.research_gun_img, "Make your gun even more powerful", new int[] { 0, 0, 1, 0, 0 });
    }

    @Override
    public void onResearchSuccess() {
        ProjectilePlayerBullet.addDamage(20);
    }
}

/*src/ui/research/SmallGunResearch.java*/
/*
 * Research SmallGun (Gun level 1)
 */
package ui.research;

import model.RenderableHolder;
import model.projectile.ProjectilePlayerBullet;

public class SmallGunResearch extends ResearchItem {
    public SmallGunResearch() {
        super(RenderableHolder.research_smallgun_img, "Make your gun more powerful", new int[] { 2, 2, 0, 0, 0 });
    }

    @Override
    public void onResearchSuccess() {
        ProjectilePlayerBullet.addDamage(10);
    }
}

/*src/ui/research/ResearchItem.java*/
/*
 * Base class for research item
 */
package ui.research;

import javafx.scene.control.Tooltip;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import logic.ResourceManager;

public abstract class ResearchItem extends ImageView {
    private boolean isResearched;
    private int[] resourceNeeded;

    public ResearchItem(Image img, String tooltip, int[] resourceNeeded) {
        isResearched = false;
        this.resourceNeeded = resourceNeeded;
        tooltip += "\nWood:" + resourceNeeded[0] + " Stone:" + resourceNeeded[1] + " Iron:" + resourceNeeded[2]
                + " Diamond:" + resourceNeeded[3] + " Artifact:" + resourceNeeded[4];
        setImage(img);
        Tooltip.install(this, new Tooltip(tooltip));
        setOnMouseClicked(e -> {
            research();
        });
    }

    public boolean isResearched() {
        return isResearched;
    }

    private void research() {
        if (isResearched)
            return;
        for (int i = 0; i < 5; i++) {
            if (resourceNeeded[i] > ResourceManager.instance.getResource(i)) {
                return;
            }
        }
        for (int i = 0; i < 5; i++) {
            ResourceManager.instance.addResource(i, -resourceNeeded[i]);
        }
        isResearched = true;
        this.setEffect(new ColorAdjust(0, -0.5, -0.5, 0));
        onResearchSuccess();
    }

    public abstract void onResearchSuccess();
}

/*src/ui/research/HealthRegenerationResearch.java*/
/*
 * Increase 10 HP every tick
 */
package ui.research;

import model.Player;
import model.RenderableHolder;

public class HealthRegenerationResearch extends ResearchItem {
    public HealthRegenerationResearch() {
        super(RenderableHolder.research_regen_img, "Health Regeneration", new int[] { 0, 0, 0, 0, 500 });
    }

    @Override
    public void onResearchSuccess() {
        Player.setHealthRegenerationRate(10);
    }
}

/*src/ui/research/DiamondResearch.java*/
/*
 * Unlock diamond storage
 */
package ui.research;

import logic.ResourceManager;
import model.RenderableHolder;

public class DiamondResearch extends ResearchItem {
    public DiamondResearch() {
        super(RenderableHolder.research_diamond_img, "Unlock diamond", new int[] { 50, 50, 30, 0, 2500 });
    }

    @Override
    public void onResearchSuccess() {
        ResourceManager.instance.addCapacity(ResourceManager.DIAMOND, 5);
    }
}

/*src/ui/research/SuperGunResearch.java*/
/*
 * Research SuperGun (Gun level 3)
 */
package ui.research;

import model.RenderableHolder;
import model.projectile.ProjectilePlayerBullet;

public class SuperGunResearch extends ResearchItem {
    public SuperGunResearch() {
        super(RenderableHolder.research_supergun_img, "Make your gun even a lot more powerful",
                new int[] { 0, 0, 0, 1, 0 });
    }

    @Override
    public void onResearchSuccess() {
        ProjectilePlayerBullet.addDamage(40);
    }
}

/*src/ui/research/BuildingResearch.java*/
/*
 * Unlock wood and stone storage
 */
package ui.research;

import logic.ResourceManager;
import model.RenderableHolder;

public class BuildingResearch extends ResearchItem {
    public BuildingResearch() {
        super(RenderableHolder.research_building_img, "Unlock wood & stone", new int[] { 0, 0, 0, 0, 25 });
    }

    @Override
    public void onResearchSuccess() {
        ResourceManager.instance.addCapacity(ResourceManager.WOOD, 5);
        ResourceManager.instance.addCapacity(ResourceManager.STONE, 5);
    }
}

/*src/ui/research/IronResearch.java*/
/*
 * Unlock iron storage
 */
package ui.research;

import logic.ResourceManager;
import model.RenderableHolder;

public class IronResearch extends ResearchItem {
    public IronResearch() {
        super(RenderableHolder.research_iron_img, "Unlock iron", new int[] { 10, 10, 0, 0, 1000 });
    }

    @Override
    public void onResearchSuccess() {
        ResourceManager.instance.addCapacity(ResourceManager.IRON, 5);
    }
}
```