# Tank Wars & KoalaBr8
# Manual

CSC 413 Summer 2017

Professor Anthony

Developer's Guide

August 13, 2017

**Team 4:**

Rong Tian Huang

&

Dilraj Singh

Table of Contents

## Link:

https://github.com/sdilraj/tankgame-irilakkuma-sdilraj

https://github.com/CSC413-SFSU/csc413-koala-team4

## Project Information and Overview:

Project Title: Tank and Koala Game

Class: CSC 413 (Summer 2017)

This is a developer's guide for the Tank Wars and KoalaBr8 single and multiplayer game. It will discuss and contain information on how the two programs are structured, how does it simply work, and problems we faced during the development to the ways how we handled the situation. Simply, if the problem is solved, we will discuss how the problem is solved. This information will be helpful for those who wants to maintain these programs or those who wants to improve these programs.

A little about the two games: The first game, called Tank Wars, is a two-player game where the opponent must try to destroy the other tank by shooting its missiles. One of the player can win by shooting the opponent for 4 times. While the Koala game is a single-player survival game in which the player must get all the koalas to the exit by avoiding the explosives and moving saws. There are other miscellaneous objects which will aid the koala to reach its destination.

## Resources:

All the resources for both games such as the images and audio were given to us by the instructor that provided us the project.

## Assumptions:

This development work was done on a Mac and using NetBeans IDE as a helper. Thus, if problems occurred during the process of running these two programs, the ideal environment would be using the two above.

We also assumed the following:

1) Since the animations of the explosion and koala movement images does not work if accessed from the folder directly, we uploaded the animations online. Thus, when the game runs, the java files will load some images from online.
2) Audio with .mp3 format did not work, so we converted it into a .wav
3) A game with a fixed map and non-spilt screen does not necessary need a mini-map.
4) Missiles shot, despite its target's position or angle, will deal the same damage to the Tank and breakable walls.
5) User already knows the buttons to move and shoot since instructions are on each of the Readme file or read instructions here.

****The Tank Game may randomly freeze and the tanks will not be able to move. We are unsure of why this is happening, but perhaps it is because of multiple key adapters or that we are not*

*using threads in the game. It seems to happen mostly when on player does not move for a certain period.*

## Compilation:

Both projects can either be run from a IDE or from command line.

Using the command line, prompt all the *java* files and compile the files into *class* files then turn those into a single *jar* file. This makes it easier to use the program in the future without having it to run the files in the command line or import it into a IDE every time. Also, *java* files can be use in any platform and is very portable, making it very convenient. First clone or download all the files from GitHub; then open *src* folder which includes *java* files and *resources* folder in NetBeans and run any of the files from there.

## Scope of Work:

The outline of the Tank Game has been made from complete scratch and majority of the Tank War's source code was reused for KoalaBr8. The Tank was simply our way of learning how to code a game, so a lot of the codes and checks in there were unnecessary. Similar games and their sources were provided in a form of examples, along with the external guidelines of what the game should contain were provided by the instructor. These were given to students so they can have an outline of how their game should look. The task is to make two games and implement all their function for it to work. How to implement each function was completely depended on each individual team.

## Features:

Tank and Koala Game can both be used as an entertainment in multiple ways.

1) Tank Game can be played by two players at the same time to verse one another.
2) Tank game players must hit their co-player four times with missiles to eliminate their co-player to win the game.
3) Koala Game is a single player game where one must take all the Koalas to the exit to win the game.
4) Koala Game is challenging because all the three Koalas move at the same time and follow the same command depending on which arrow key the user presses.
5) Both games have user friendly sounds, it's not too loud or too soft.
6) Tank and Koala game will both restart after they have won or lost the round.

## Tank Game vs. Koala Game:

The Tank Game option was given to us by the class instructor but everyone had the choice if they want to make both tanks to work together or verse each other. On the other hand, for the second game, we had an option and we decided to make the Koala Game. We chose this game because we wanted to implement a single player game since we already had experience of making a two player Tank Game, which is somewhat of a hassle. Furthermore, we chose this because it was more interesting compare to other games and we felt like Koala Game was also easier to implement. In the Tank Game, we had to make every single class from scratch while majority of the classes for Koala Game was reused from the Tank Game. Tank game was more

difficult to implement and we had more problems and issues we had to fix regarding the tank game. Perhaps this is because it is our first game and we had to learn most of the materials. On the other hand, we had minor issues with the Koala game and could figure out the problems easily. Classes such as Game Object, Explosion, and Sound were taken directly from the Tank Game and put in the Koala Game with only minor changes. Overall, both games are similar, but unique at the same time, in their own ways.

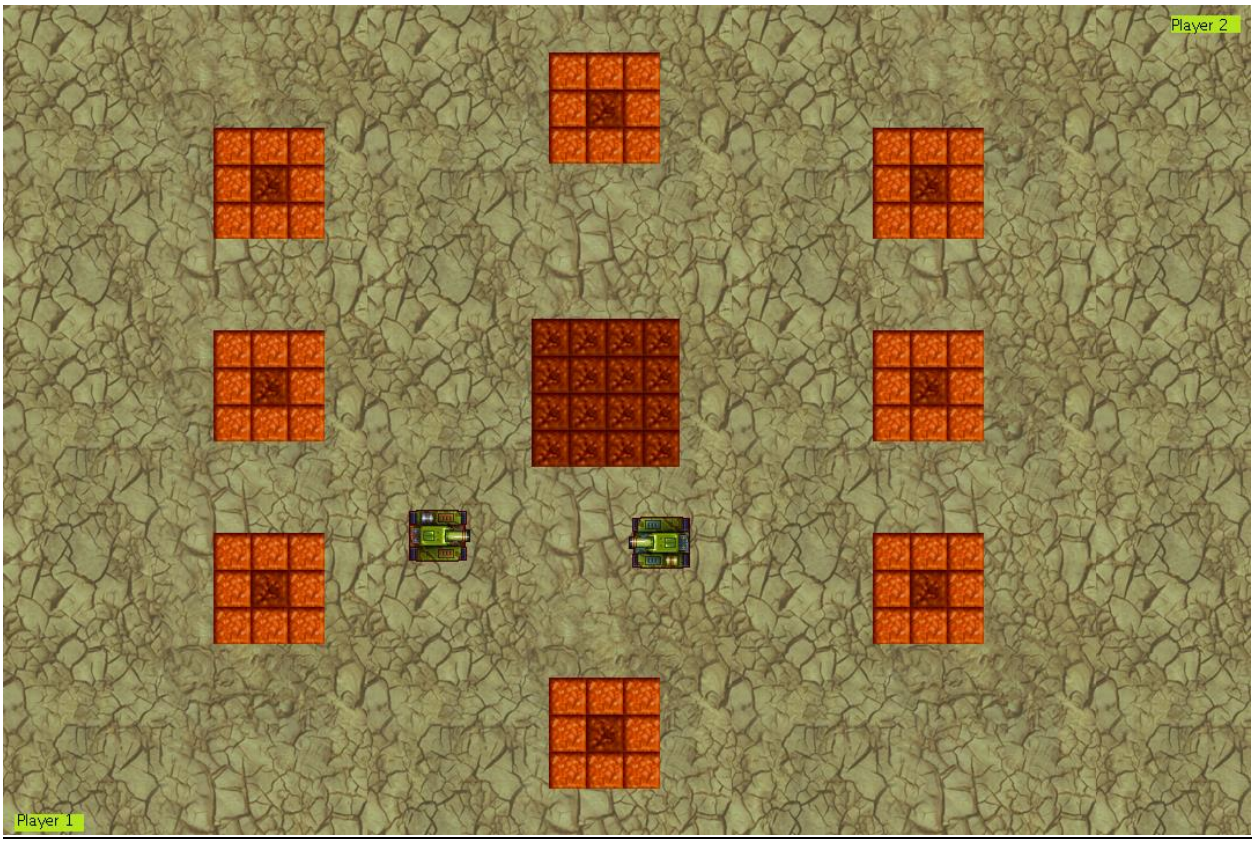
## Tank Controls & Explanation:

This guide covers 15 java files of the Tank game, all of them are used to make the tanks have wars between one another. The Tank game looks very alike any 2Dgame, where two players can verse one another. Player 1 uses the arrow keys to move around and spacebar to shoot missiles. While Player 2 uses WASD keyword keys to move around and Z to shoot missiles. The tank game has eight destroyable blocks and indestructible block of squares in the middle, which can be used as a shield while crossing firing their opponent. Both tanks start off with opposite ends in the beginning of the game before they can move around to engage in the battle. Tanks can fire multiple shots at one another but to win, one must hit the other tank four times. Each time a tank is hit by the missile, it's health drops by 25% and after four hits, the screen displays a message saying player 1 or player 2 won, depending on which tank got hit with missiles four times first.

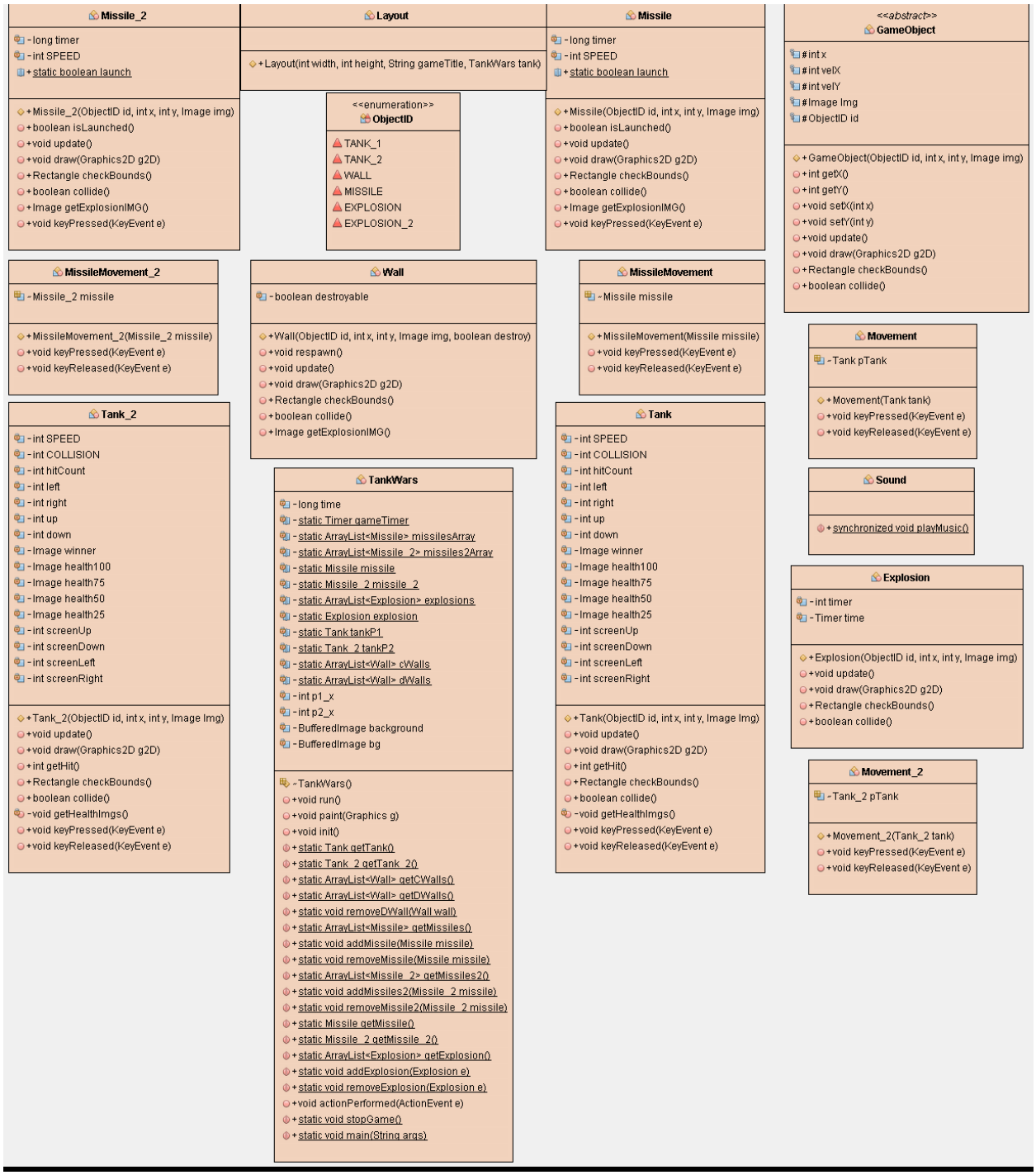
## Koala Controls & Explanation:

This guide overs 14 java files for the Koala game, all of them are designed for a single player mode. The Koala game uses the arrow keys to move around. There are three koalas in the game and all of them move in the same directions at the same time, which means they response to the keys. There are TNT and blades in the game, which the Koalas must be beware off, if the koalas come encounter with either one of those then koalas will die on the spot. Meanwhile, there are rocks in the game which the koalas can use to explode the TNT and clear their path. But when it comes to the blade, they have no choice but to avoid encountering, or else they will be sliced at the spot. Another very important item in the game are locks and switches. One koala cannot get through the locks unless a different or same koala goes around and touch the switch of that lock, which will deactivate the lock and will clear the path of the koalas. If all three of the koalas gets through the locks, blades, and rocks then they simply must reach to the exit, to win the game.

## Tank Game Screen:



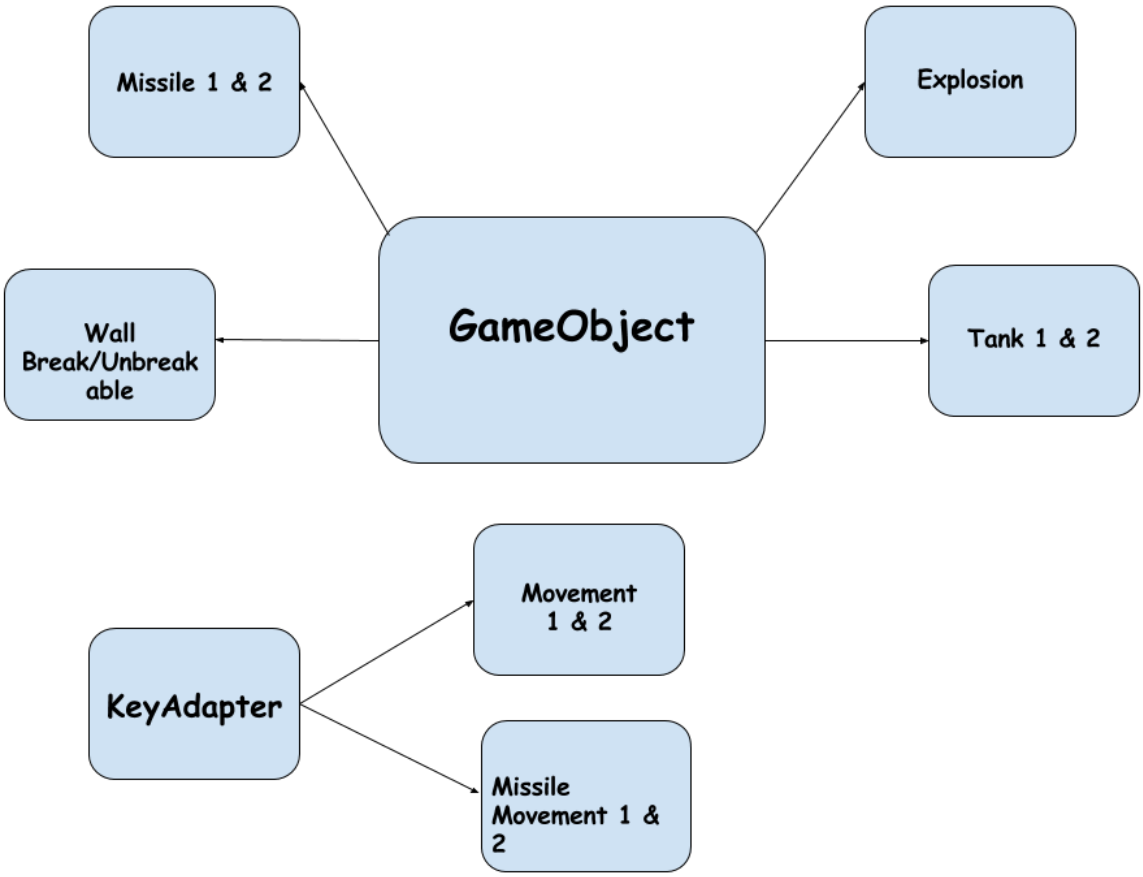
# Structure of Tank Game:



## Tank Game Class Hierarchy:

Most of the classes extend the hierarchy, GameObject such as the Tank, Explosion, and Walls.

The arrows indicate parent-child relation between the classes.



## Tank Game Classes Overview:

**GameObject.java:** The hierarchy of all the objects that extends this class. It implements common data and methods, but the differences are enclosed in the children classes.

**Layout.java:** Deploys the frame of the game in a JPanel.

**ObjectID.java:** An enum class that identifies the name of its classes.

**Sound.java:** Plays the background music as well as other miscellaneous game sounds.

**Explosion.java:** Extends GameObject. It creates an explosion animation when the collision detection detects an intersection with the walls, missile, or tank. This object is kept in an Array List in the core of the game.

**Missile.java:** A class explicitly for Tank 1. This class extends the GameObject and it creates missile objects when the user presses the key.

**Missile2.java:** Same as Missile, but this class is explicitly for Tank 2.

**MissileMovement.java:** Extends KeyAdapter. This class listens to the key when Tank 1 shoots its missile.

**MissileMovement2.java:** Extends KeyAdapter. This class listens to the key when Tank 2 shoots its missile.

**Tank.java:** Extends GameObject. This class is can be represented as one of the most important class as it checks for most of the collision in the game and uses its own set of key movements.

**Tank2.java:** Extends GameObject. This class is can be represented as one of the most important class as it checks for most of the collision in the game and uses its own set of key movements.

**Movement.java:** Extends KeyAdapter. This class listens to the key movements for Tank 1.

**Movement_2.java:** Extends KeyAdapter. This class listens to the key movements for Tank 2.

**Wall.java:** Extends GameObject. This class is used for creating wall objects in the core of the game in the Array List and it draws each wall object out one at a time.

**TankWars.java:** The is the core of the game. It uses all the classes created in the project to construct and draw images that ultimately fits into a single game. The structures it uses are basically static variables, Array Lists, Timers, accessors, mutators, and methods that loads certain images for the objects. It also implements a Runnable and ActionListener component which is part of the java library.

## GameObject.java

```
public abstract class GameObject {
    protected int x, y;
    protected int velX = 0;
    protected int velY = 0;
    protected Image Img;
    protected ObjectID id;

    public GameObject(ObjectID id,int x, int y, Image img) {
        this.x = x;
        this.y = y;
        this.Img = img;
        this.id = id;
    }

    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
    public void setX(int x) {
        this.x = x;
    }
    public void setY(int y) {
        this.y = y;
    }

    public abstract void update();

    public abstract void draw(Graphics2D g2D);

    public abstract Rectangle checkBounds();

    public abstract boolean collide();

}
```

## Layout.java

```
public class Layout {

    public Layout(int width, int height, String gameTitle, TankWars tank) {
        tank.setPreferredSize(new Dimension(width, height));

        JFrame gameInterface = new JFrame(gameTitle);
        gameInterface.add(tank);
        gameInterface.pack();
        gameInterface.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        gameInterface.setVisible(true);
        gameInterface.setResizable(false);

    }
}
```

## Sound.java

```
public class Sound {
    public static synchronized void playMusic() {
        try {
            Clip musicClip = AudioSystem.getClip();
            AudioInputStream input = AudioSystem.getAudioInputStream(new File("Resources/Music.wav"));
            musicClip.open(input);
            musicClip.start();
        } catch (LineUnavailableException | UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Sound.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

## ObjectID.java

```
public enum ObjectID {
    TANK_1,
    TANK_2,
    WALL,
    MISSILE,
    EXPLOSION,
    EXPLOSION_2;
}
```

Explosion.java

```
public class Explosion extends GameObject{
    private int timer = 0;
    private Timer time = new Timer();

    public Explosion(ObjectID id, int x, int y, Image img) {
        super(id, x, y, img);

    }

    @Override
    public void update() {
        TankWars.removeExplosion(this);
    }

    @Override
    public void draw(Graphics2D g2D) {
        g2D.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return null;
    }

    @Override
    public boolean collide() {
        return false;
    }
```

## Missile.java

```
public class Missile extends GameObject{
    private long timer = 0;
    private final int SPEED = 4;
    public static boolean launch = false;

    public Missile(ObjectID id, int x, int y, Image img) {
        super(id, x, y, img);

    }

    public boolean isLaunched() {
        return launch;
    }

    @Override
    public void update() {
        if (!collide()) {
            x += 5;
        } else {
            x += 0;
            TankWars.removeMissile(this);
        }
    }

    @Override
    public void draw(Graphics2D g2D) {
        // Since x based on x coordinates of tank, doesn't work really well when moving and shooting
        g2D.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public boolean collide() {
        // Missile 1 to Walls (Concrete)
        ArrayList <Wall> cWalls = TankWars.getCWalls();
        for (int i = 0; i < cWalls.size(); i++) {
            if (checkBounds().intersects(cWalls.get(i).checkBounds())) {
                TankWars.addExplosion(new Explosion(ObjectID.EXPLOSION, cWalls.get(i).x - 25, cWalls.get(i).y,
                return true;
            }
        }

        // Missile 1 to Missile 2 Collision
        ArrayList<Missile 2> missiles = TankWars.getMissiles2();
```

## Missile2.java

```
public class Missile_2 extends GameObject{
    private long timer = 0;
    private final int SPEED = 4;
    public static boolean launch = false;


    public Missile_2(ObjectID id, int x, int y, Image img) {
        super(id, x, y, img);
    }

    public boolean isLaunched() {
        return launch;
    }

    @Override
    public void update() {
        if (!collide()) {
            x -= 5;
        } else {
            x += 0;
            TankWars.removeMissile2(this);
        }
    }

    @Override
    public void draw(Graphics2D g2D) {
        // Since x based on x coordinates of tank, doesn't work really well when moving and shooting
        g2D.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public boolean collide() {
        // Missile 2 to Walls Collision (Concrete)
        ArrayList <Wall> cWalls = TankWars.getCWalls();
        for (int i = 0; i < cWalls.size(); i++) {
            if (checkBounds().intersects(cWalls.get(i).checkBounds())) {
                TankWars.removeMissile2(this);
                TankWars.addExplosion(new Explosion(ObjectID.EXPLOSION, cWalls.get(i).x + 10, cWalls.get(i).y,
                return true;
            }
        }
```

## MissileMovement.java

```
public class MissileMovement extends KeyAdapter {
    Missile missile;

    public MissileMovement(Missile missile) {
        this.missile = missile;
    }

    @Override
    public void keyPressed(KeyEvent e) {
        missile.keyPressed(e);
    }

    @Override
    public void keyReleased(KeyEvent e) {

    }
}
```

## MissileMovement2.java

```
public class MissileMovement_2 extends KeyAdapter {
    Missile_2 missile;

    public MissileMovement_2(Missile_2 missile) {
        this.missile = missile;
    }

    @Override
    public void keyPressed(KeyEvent e) {
        missile.keyPressed(e);
    }

    @Override
    public void keyReleased(KeyEvent e) {

    }
}
```

## Tank.java

```
public class Tank extends GameObject{
    private final int SPEED = 2;
    private final int COLLISION = 1;
    private int hitCount;

    private int left;
    private int right;
    private int up;
    private int down;

    private Image winner;
    private Image health100;
    private Image health75;
    private Image health50;
    private Image health25;

    /*****************************/
    private int screenUp = -5;
    private int screenDown = 664;
    private int screenLeft = -5;
    private int screenRight = 1015;
    /*****************************/


    public Tank(ObjectID id, int x, int y, Image Img) {
        super(id, x, y, Img);
        getHealthImgs();
    }

    @Override
    public void update() {
        //System.out.println("Tank 1: " + x + ", " + y);
        if(!collide()) {
            if (y > screenDown) {           // Game Frame Boundaries
                y += -1;                    //
            } else if (y < screenUp) {      //
                y += 1;                     //
            } else if (x > screenRight) {
                x += -1;
            } else if ( x < screenLeft) {
                x += 1;
            } else {
                x += velX;
                y += velY;
            }
        } else {
            if((x + x) > right)
                x += COLLISION;
            if ((y + y) > down)
```

## Tank2.java

```
public class Tank_2 extends GameObject{
    private final int SPEED = 2;
    private final int COLLISION = 1;
    private int hitCount;

    private int left;
    private int right;
    private int up;
    private int down;

    private Image winner;
    private Image health100;
    private Image health75;
    private Image health50;
    private Image health25;

    /*****************************/
    private int screenUp = -5;
    private int screenDown = 664;
    private int screenLeft = -5;
    private int screenRight = 1015;
    /*****************************/

    public Tank_2(ObjectID id, int x, int y, Image Img) {
        super(id, x, y, Img);
        getHealthImgs();
    }

    @Override
    public void update() {
        //System.out.println("Tank 2: " + x + ", " + y);
        if(!collide()) {
            if (y > screenDown) {           // Game Frame Boundaries
                y += -1;                    //
            } else if (y < screenUp) {      //
                y += 1;                     //
            } else if (x > screenRight) {
                x += -1;
            } else if ( x < screenLeft) {
                x += 1;
            } else {
                x += velX;
                y += velY;
            }
        }
        else {
            if((x + x) > right)
                x += COLLISION;
            if ((y + y) > down)
```

## Movement.java

```
public class Movement extends KeyAdapter {
    Tank pTank;

    public Movement(Tank tank) {
        pTank = tank;
    }

    @Override
    public void keyPressed(KeyEvent e) {
        //int code = e.getKeyCode();
            pTank.keyPressed(e);
    }

    @Override
    public void keyReleased(KeyEvent e) {
        //int code = e.getKeyCode();
            pTank.keyReleased(e);
    }
}
```

## Movement2.java

```
public class Movement_2 extends KeyAdapter {
    Tank_2 pTank;

    public Movement_2(Tank_2 tank) {
        pTank = tank;
    }

    @Override
    public void keyPressed(KeyEvent e) {
        int code = e.getKeyCode();
            pTank.keyPressed(e);
    }

    @Override
    public void keyReleased(KeyEvent e) {
        int code = e.getKeyCode();
            pTank.keyReleased(e);
    }
}
```

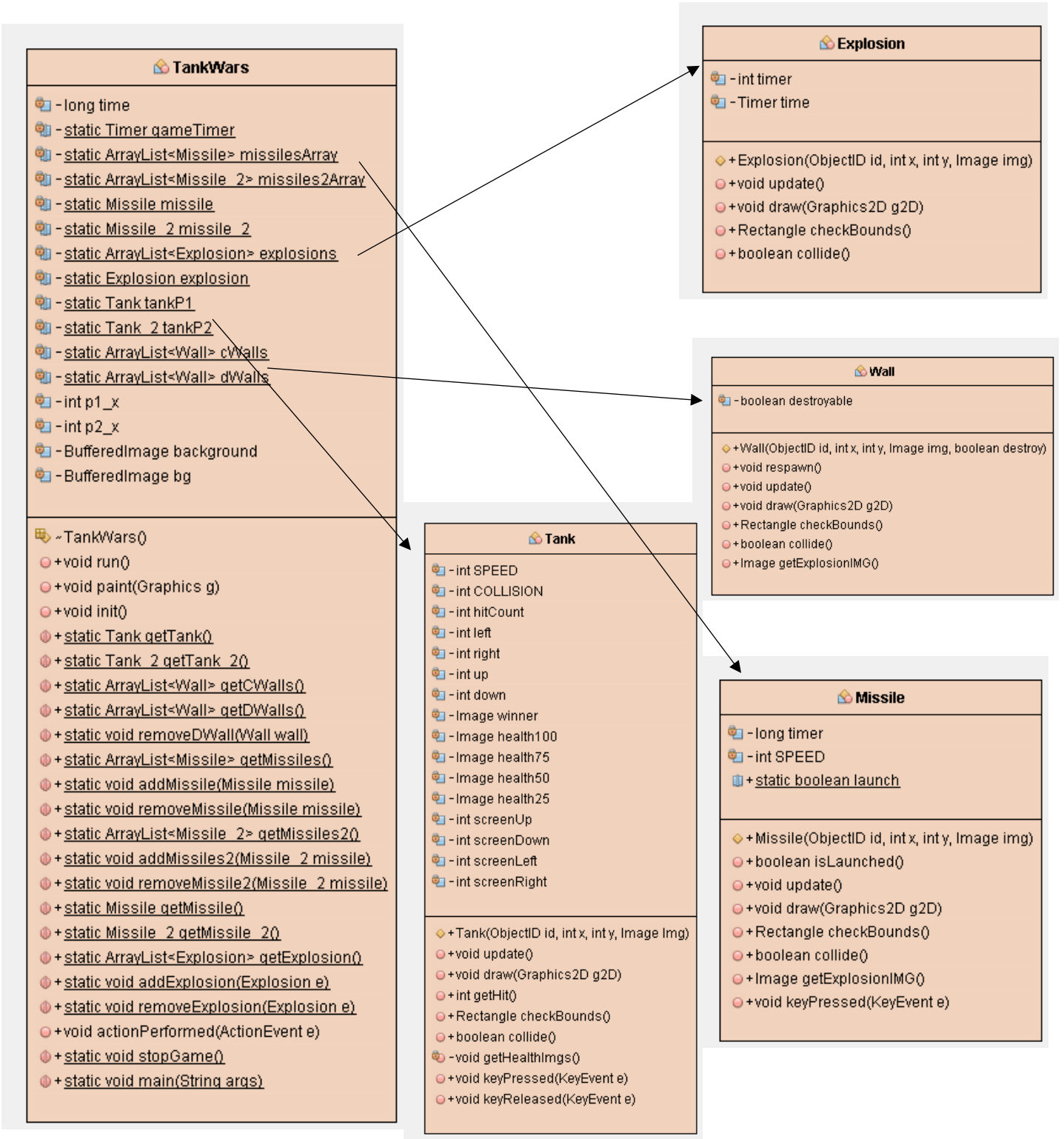
## Wall.java

```
public class Wall extends GameObject {
    private boolean destroyable;

    public Wall(ObjectID id, int x, int y, Image img, boolean destroy) {
        super(id, x, y, img);
        this.destroyable = destroy;
    }

    public void respawn() {

    }

    @Override
    public void update() {
        if (collide()) {
            TankWars.removeDWall(this);
        }
    }

    @Override
    public void draw(Graphics2D g2D) {
        g2D.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public boolean collide() {
        // Missile 1 to Wall Collision (destroyable)
        ArrayList <Missile> missiles = TankWars.getMissiles();
        for (int i = 0; i < missiles.size(); i++) {
            if (checkBounds().intersects(missiles.get(i).checkBounds())) {
                TankWars.removeMissile(missiles.get(i));
                TankWars.addExplosion(new Explosion(ObjectID.EXPLOSION, x - 25, y - 10, getExplosionIMG()));
                return true;
            }
        }

        // Missile 2 to Wall Collision (destroyable)
        ArrayList <Missile_2> missiles2 = TankWars.getMissiles2();
        for (int i = 0; i < missiles2.size(); i++) {
            if (checkBounds().intersects(missiles2.get(i).checkBounds())) {
                TankWars.removeMissile2(missiles2.get(i));
                TankWars.addExplosion(new Explosion(ObjectID.EXPLOSION, x + 10, y - 10, getExplosionIMG()));
                return true;
            }
```

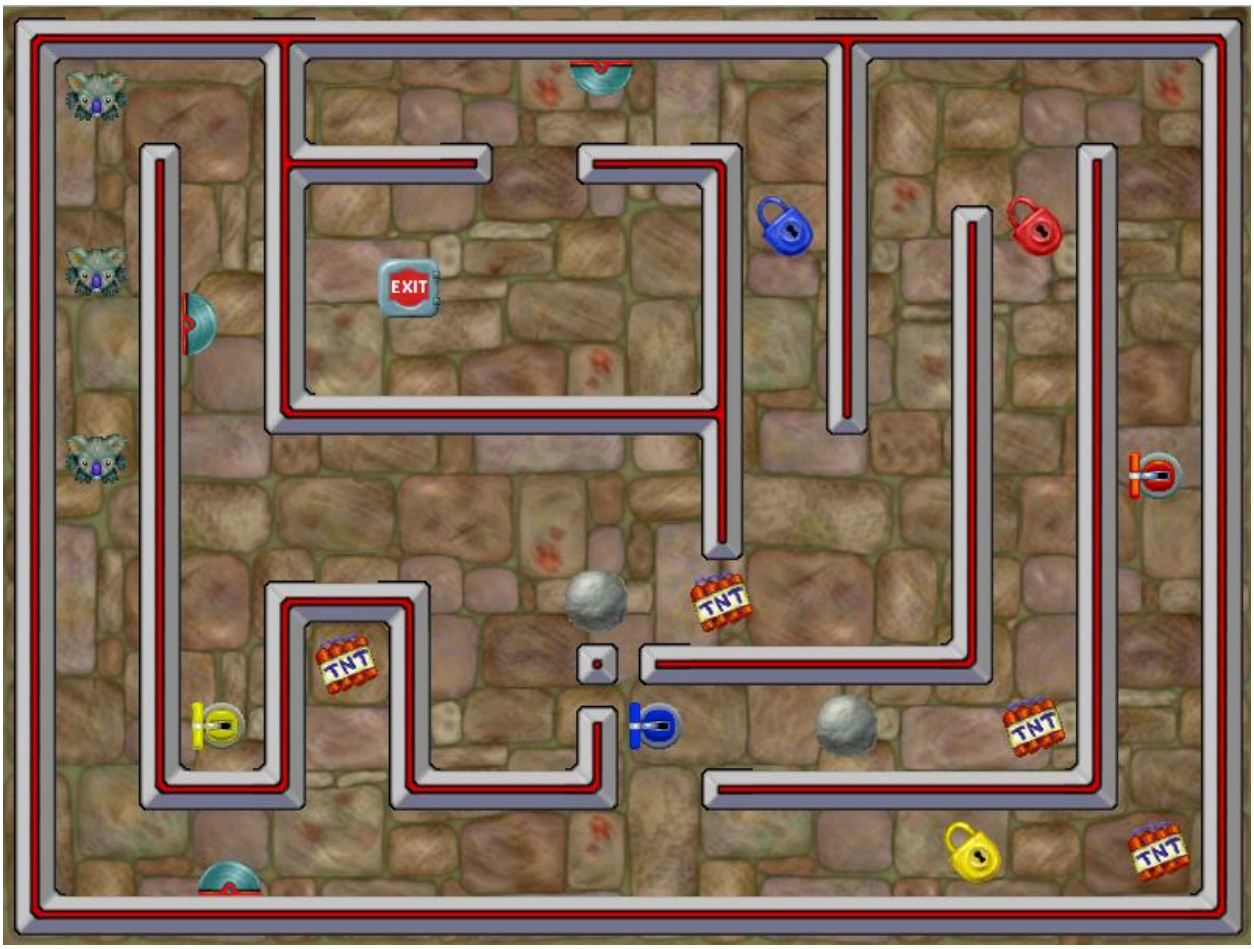
## TankWars.java

```
public class TankWars extends JPanel implements Runnable, ActionListener {
    private long time = 0;
    private static Timer gameTimer;

    private static ArrayList <Missile> missilesArray = new ArrayList<>(); // Tank 1's Missiles
    private static ArrayList <Missile_2> missiles2Array = new ArrayList<>(); // Tank 2's Missiles
    private static Missile missile;
    private static Missile_2 missile_2;

    private static ArrayList <Explosion> explosions = new ArrayList<>();
    private static Explosion explosion;

    //private static ArrayList<Tank> player1 = new ArrayList<>();
    private static Tank tankP1;
    private static Tank_2 tankP2;

    private static ArrayList<Wall> cWalls = new ArrayList<>(); // Concrete walls
    private static ArrayList<Wall> dWalls = new ArrayList<>(); // Destroyable walls



    private int p1_x = 0, p1_y = 0; // Initial player 1 coordinates
    private int p2_x = 1015, p2_y = 655; // Initial player 2 coordinates

    private BufferedImage background;
        private BufferedImage bg;

    TankWars() {
        init();
        addKeyListener(new Movement(tankP1));
        addKeyListener(new Movement_2(tankP2));
        addKeyListener(new MissileMovement(missile));
        addKeyListener(new MissileMovement_2(missile_2));
        Sound.playMusic();
        gameTimer = new Timer(5, this);
        gameTimer.start();
    }

    @Override
    public void run() {

    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2D = (Graphics2D) g;
            g.drawImage(background, 0, 0, this);
```

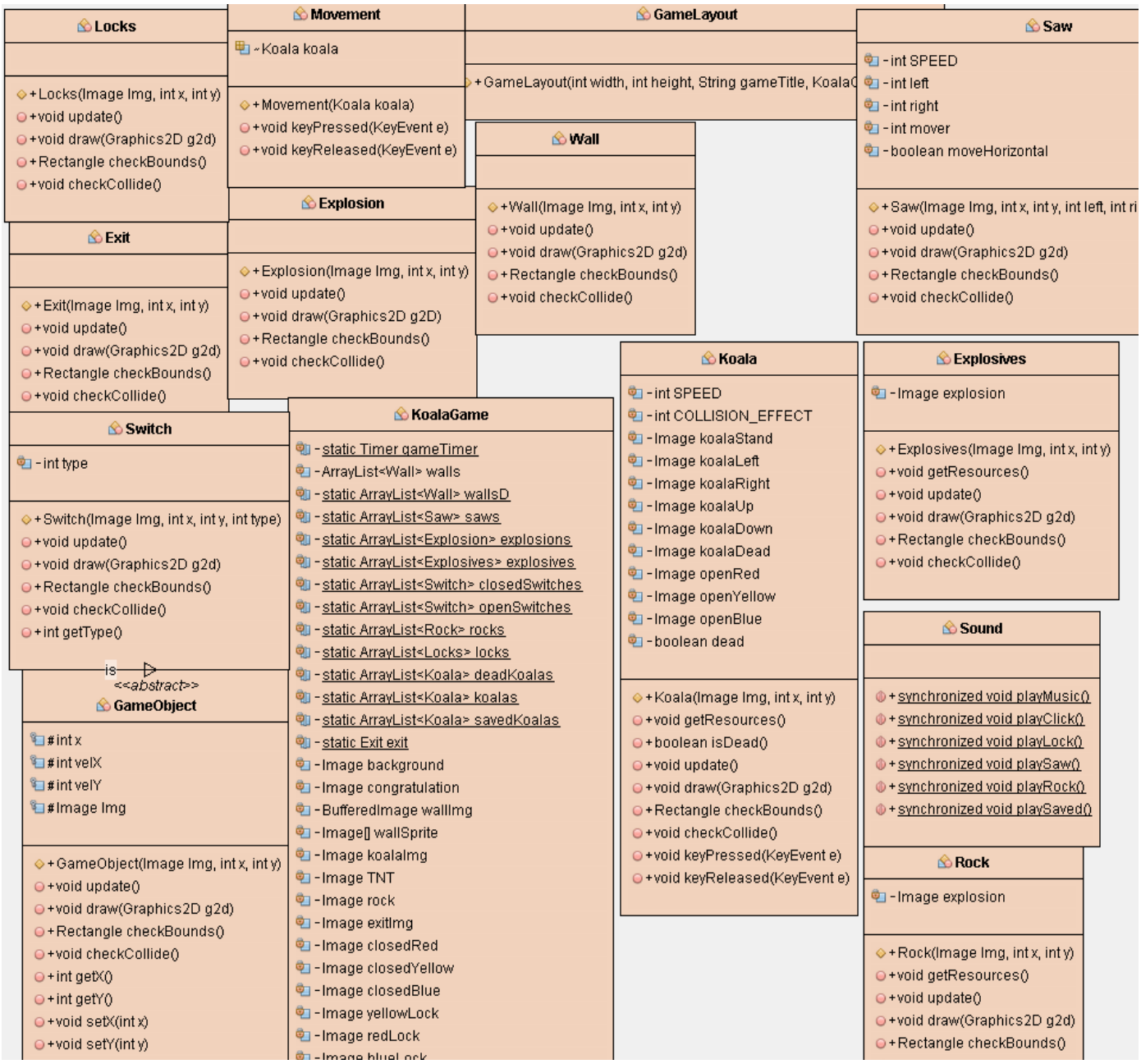
## Tank Implement Chart:



## Koala Game:



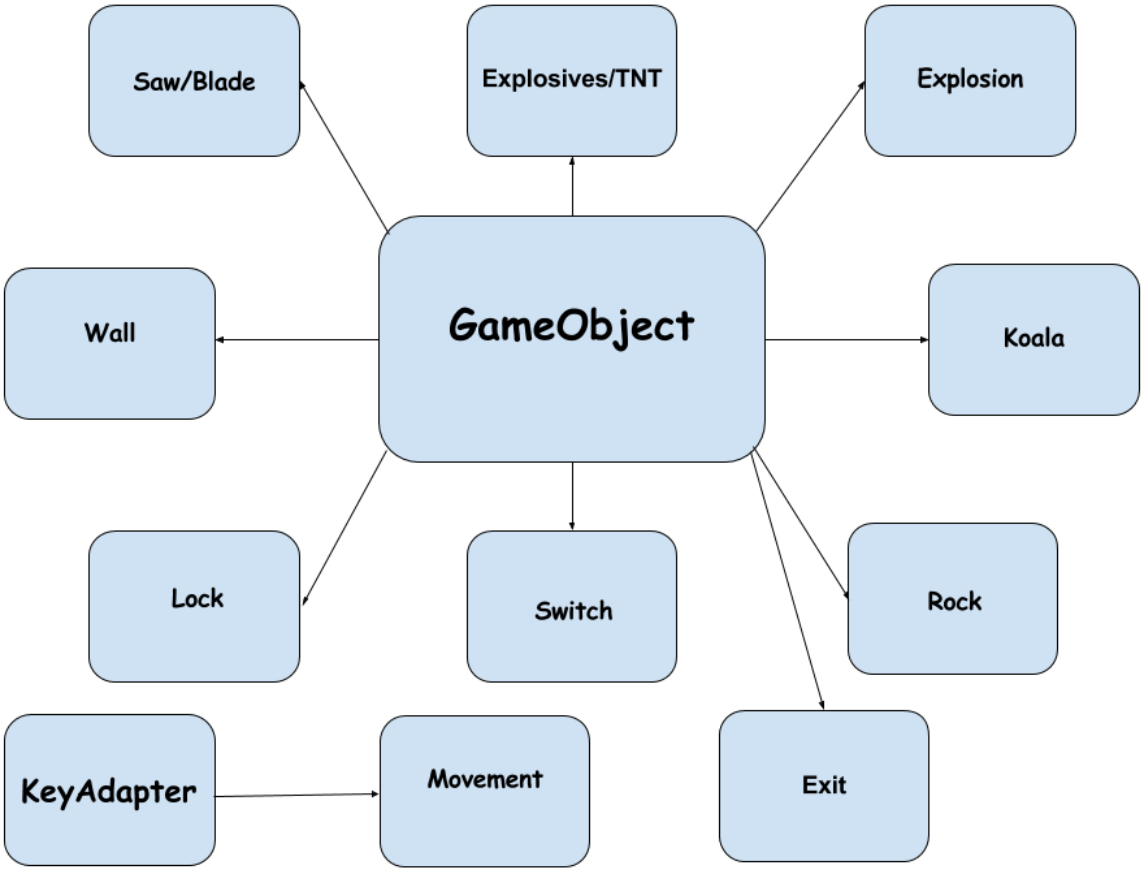
# Structure of Koala Game:



## Koala Game Class Hierarchy:

Most of the classes extend the hierarchy, GameObject such as the Koala, Explosive, and Wall.

The arrows indicate parent-child relation between the classes.



## Koala Game Classes Overview:

**GameObject.java:** The hierarchy of all the objects that extends this class. It implements common data and methods, but the differences are enclosed in the children classes.

**GameLayout.java:** Deploys the frame of the game in a JPanel.

**Sound.java:** We have multiple sounds in the game. For example, there is a music playing in the background. Next there is a click and lock sounds so when the Koalas go through the lock, it will make a sound that the lock has been unlocked. Then we have a sound which the saw makes when they slice up the Koalas, if the saw ever encounters the Koalas.

**Explosives.java:** Extends GameObject. This class is used for drawing the TNT's in the map, but the collision against the TNT is checked in the Koala class.

**Explosion.java:** Extends GameObject. This class is used for drawing the explosion animation when the Koala or Rock intersects with the TNT. Its collision for explosion is checked in the Koala and Rock classes.

**Exit.java:** Extends GameObject. This class is used for checking if a Koala reaches the exit point. If a Koala reaches this point, the Koala will be deleted from the game.

**Koala.java:** Extends GameObject. This class could be one of the most important class because most of the collision detection is checked here. It is used for creating Koala objects and listening to the movements when the Movement class is activated by pressing the keys.

**Movement.java:** Extends KeyAdapter. This class listens to the key movements for all the koalas.

**Rock.java:** Extends GameObject. This class is used for creating boulders in the game. Its purpose can be either none or for the Koala to push to destroy the TNT that is blocking the passageway. When the Koala intersects with the Rock, it will move until it is destroyed by the TNT.

**Saw.java:** Extends GameObject. This class draws the moving saws that moves from left to right or top to bottom. When the Koala interacts with the saw, it will be sliced up and died. The collision for this is done in the Koala class.

**Locks.java:** Extends GameObject. This class is used for creating the different colored locks in the game. Its purpose is to block the Koala from passing if the respective switch is not yet unlocked. The collision against blocking the Koala is done in the Koala class.

**Switch.java:** Extends GameObject. This class draws the different colored switches in the game. Its purpose is for the Koala to unlock the lock that is blocking the passage. The collision that detects this is also done in the Koala class.

**Wall.java:** Extends GameObject. This class is used for creating wall objects in the core of the game in the Array List and it draws each wall object out one at a time.

**KoalaGame.java:** The is the core of the game. It uses all the classes created in the project to construct and draw images that ultimately fits into a single game. The structures it uses are basically static variables, Array Lists, Timers, accessors, mutators, and methods that loads certain images for the objects. It also implements a Runnable and ActionListener component which is part of the java library

GameObject.java

```
public abstract class GameObject {
    protected int x, y;
    protected int velX = 0;
    protected int velY = 0;
    protected Image Img;

    public GameObject(Image Img, int x, int y) {
        this.x = x;
        this.y = y;
        this.Img = Img;
    }

    public abstract void update();

    public abstract void draw(Graphics2D g2d);

    public abstract Rectangle checkBounds();

    public abstract void checkCollide();

    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
    public void setX(int x) {
        this.x = x;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

## GameLayout.java

```
public class GameLayout {

    public GameLayout(int width, int height, String gameTitle, KoalaGame game) {
        game.setPreferredSize(new Dimension(width, height));

        JFrame gameInterface = new JFrame(gameTitle);
        gameInterface.add(game);
        gameInterface.pack();
        gameInterface.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        gameInterface.setVisible(true);
        gameInterface.setResizable(false);

    }
}
```

## Sound.java

```
public class Sound {

    public static synchronized void playMusic() {
        try {
            Clip musicClip = AudioSystem.getClip();
            AudioInputStream input = AudioSystem.getAudioInputStream(new File("Resources/Music.wav"));
            musicClip.open(input);
            musicClip.start();
            musicClip.loop(10);
        } catch (LineUnavailableException | UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Sound.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static synchronized void playClick() {
        try {
            Clip musicClip = AudioSystem.getClip();
            AudioInputStream input = AudioSystem.getAudioInputStream(new File("Resources/click.wav"));
            musicClip.open(input);
            musicClip.start();
            playLock();
        } catch (LineUnavailableException | UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Sound.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static synchronized void playLock() {
        try {
            Clip musicClip = AudioSystem.getClip();
            AudioInputStream input = AudioSystem.getAudioInputStream(new File("Resources/lock.wav"));
            musicClip.open(input);
            musicClip.start();
        } catch (LineUnavailableException | UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Sound.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static synchronized void playSaw() {
        try {
            Clip musicClip = AudioSystem.getClip();
            AudioInputStream input = AudioSystem.getAudioInputStream(new File("Resources/saw.wav"));
            musicClip.open(input);
            musicClip.start();
        } catch (LineUnavailableException | UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Sound.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

## Explosives.java

```
public Explosives(Image Img, int x, int y) {
    super(Img, x, y);
    getResources();
}

    public void getResources() {
    Image Img;
    try {
        // Loading images for Explosion
        URL url = new URL("http://i.imgur.com/BS0tKaB.gif");
        Img = Toolkit.getDefaultToolkit().createImage(url);
        explosion = Img;
    } catch (MalformedURLException ex) {
        System.out.println("Error loading Explosion File");
    }
}

@Override
public void update() {
    checkCollide();
}

@Override
public void draw(Graphics2D g2d) {
    g2d.drawImage(Img, x, y, null);
}

@Override
public Rectangle checkBounds() {
    return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
}

@Override
public void checkCollide() {
    ArrayList<Explosion> explosions = KoalaGame.getExplosions();

    ArrayList<Koala> koalas = KoalaGame.getKoala();
    for(int i = 0; i < koalas.size(); i++) {
        if (checkBounds().intersects(koalas.get(i).checkBounds())) {
            System.out.println("Koala - bomb");
            if(explosions.size() == 1) {
                KoalaGame.removeExplosion(explosions.get(0));
            }
            KoalaGame.addExplosion(new Explosion(explosion, x, y));
        }
    }
}
```

Explosion.java

```
public class Explosion extends GameObject{

    public Explosion(Image Img, int x, int y) {
        super(Img, x, y);

    }

    @Override
    public void update() {
        // NO UPDATES NEEDED
    }

    @Override
    public void draw(Graphics2D g2D) {
        g2D.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return null;
    }

    @Override
    public void checkCollide() {
        // NO COLLISION NEEDED
    }

}
```

## Exit.java

```
public class Exit extends GameObject {

    public Exit(Image Img, int x, int y) {
        super(Img, x, y);
    }

    @Override
    public void update() {
        // LOCKS DO NOT MOVE
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public void checkCollide() {
        // DO NOTHING, COLLISION CHECK IN KOALA CLASS
    }

}
```

## Koala.java:

```
public class Koala extends GameObject {
    private final int SPEED = 1;
    private final int COLLISION_EFFECT = 1;

    private Image koalaStand;
    private Image koalaLeft;
    private Image koalaRight;
    private Image koalaUp;
    private Image koalaDown;
    private Image koalaDead;

    private Image openRed;
    private Image openYellow;
    private Image openBlue;

    private boolean dead = false;

    public Koala(Image Img, int x, int y) {
        super(Img, x, y);
        koalaStand = Img; // Original, Standing position
        getResources();
    }

    public void getResources() {
        Image Img;
        try {
            // Loading images for Explosion
            URL url = new URL("http://i.imgur.com/b2Sfl5w.gif");
            Img = Toolkit.getDefaultToolkit().createImage(url);
            koalaDown = Img;
            url = new URL("http://i.imgur.com/gNHWU5g.gif");
            Img = Toolkit.getDefaultToolkit().createImage(url);
            koalaUp = Img;
            url = new URL("http://i.imgur.com/kuVN3U3.gif");
            Img = Toolkit.getDefaultToolkit().createImage(url);
            koalaLeft = Img;
            url = new URL("http://i.imgur.com/7ZZLL9z.gif");
            Img = Toolkit.getDefaultToolkit().createImage(url);
            koalaRight = Img;
        } catch (MalformedURLException ex) {
            System.out.println("Error loading Koala Files...");
        }

        ImageIcon ImgD = new ImageIcon("Resources/koala_dead.gif");
        koalaDead = ImgD.getImage();
        ImgD = new ImageIcon("Resources/switch_yellow_open.gif");
        openYellow = ImgD.getImage();
        ImgD = new ImageIcon("Resources/switch_red_open.gif");
```

## Rock.java

```
public class Rock extends GameObject{
    private Image explosion;

    public Rock(Image Img, int x, int y) {
        super(Img, x, y);
        getResources();
    }

    public void getResources() {
        Image Img;
        try {
            // Loading images for Explosion
            URL url = new URL("http://i.imgur.com/BS0tKaB.gif");
            Img = Toolkit.getDefaultToolkit().createImage(url);
            explosion = Img;
        } catch (MalformedURLException ex) {
            System.out.println("Error loading Explosion File");
        }
    }

    @Override
    public void update() {
        checkCollide();
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public void checkCollide() {
        // Koala to Rock Collision
        ArrayList<Koala> koalas = KoalaGame.getKoala();
        for(int i = 0; i < koalas.size(); i++) {
            if(checkBounds().intersects(koalas.get(i).checkBounds())) {
                x += 1;
```

## Locks.java

```
public class Locks extends GameObject {

    public Locks(Image Img, int x, int y) {
        super(Img, x, y);
    }

    @Override
    public void update() {
        // LOCKS DO NO MOVE
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y-10, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public void checkCollide() {
```

Switch.java

```
public class Switch extends GameObject {
    private int type;

    public Switch(Image Img, int x, int y, int type) {
        super(Img, x, y);
        this.type = type;
    }

    @Override
    public void update() {
        // SWITCHES DO NOT MOVE
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle(x, y, Img.getWidth(null), Img.getHeight(null));
    }

    @Override
    public void checkCollide() {
        // DO NOTHING, AS COLLISION CHECK IN KOALA CLASS
    }

    public int getType() {
        return type;
    }

}
```

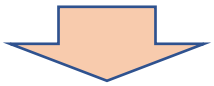
## Wall.java

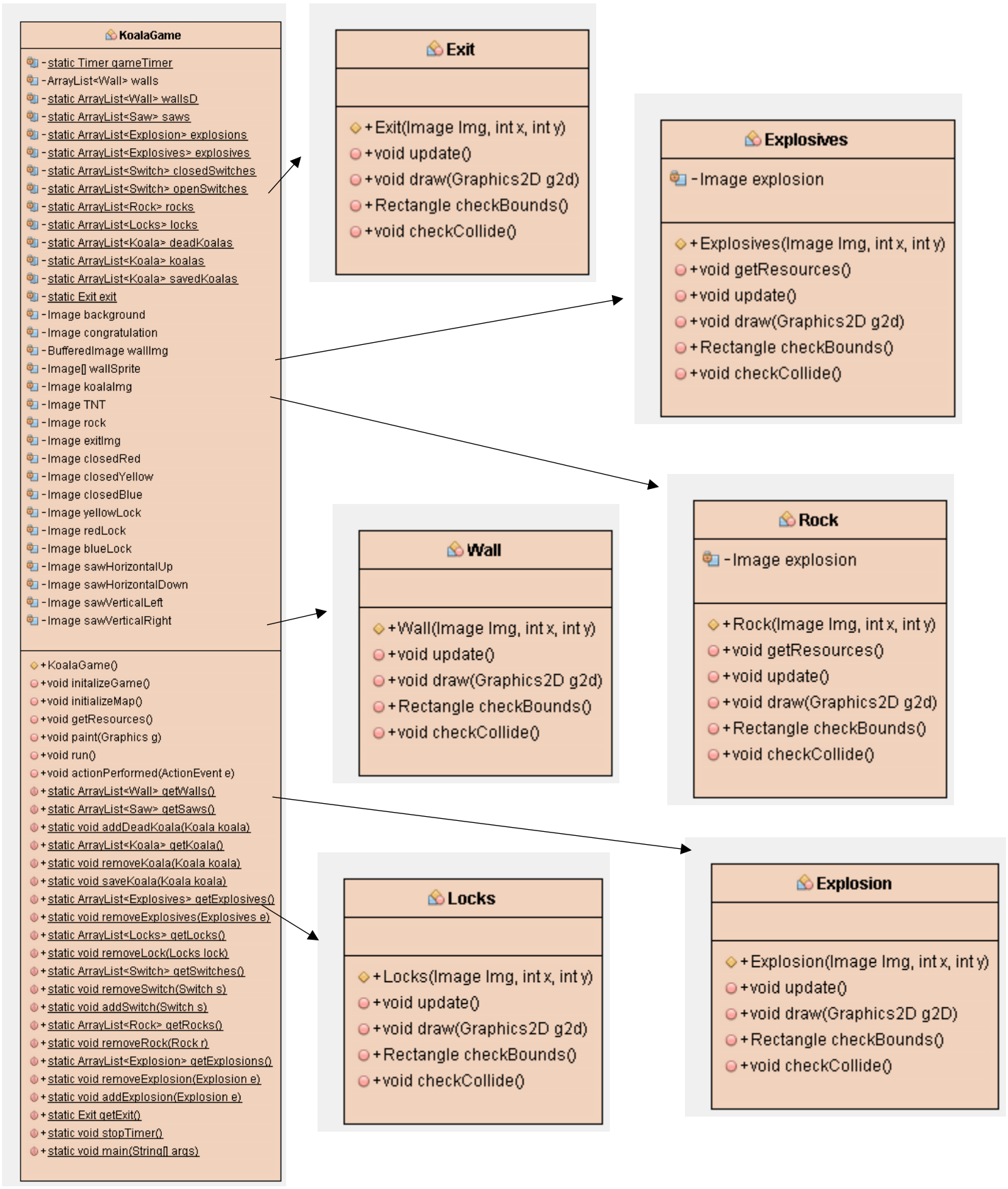
```
public class Wall extends GameObject {

    public Wall(Image Img, int x, int y) {
        super(Img, x, y);
    }

    @Override
    public void update() {
        // DO NOTHING AS WALLS DO NOT MOVE
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.drawImage(Img, x, y, null);
    }

    @Override
    public Rectangle checkBounds() {
        return new Rectangle((x+5), (y+5), (Img.getWidth(null))-10, (Img.getHeight(null))-10);
    }

    @Override
    public void checkCollide() {
        // DO NOTHING, COLLISION CHECK IN KOALA CLASS
    }

}
```

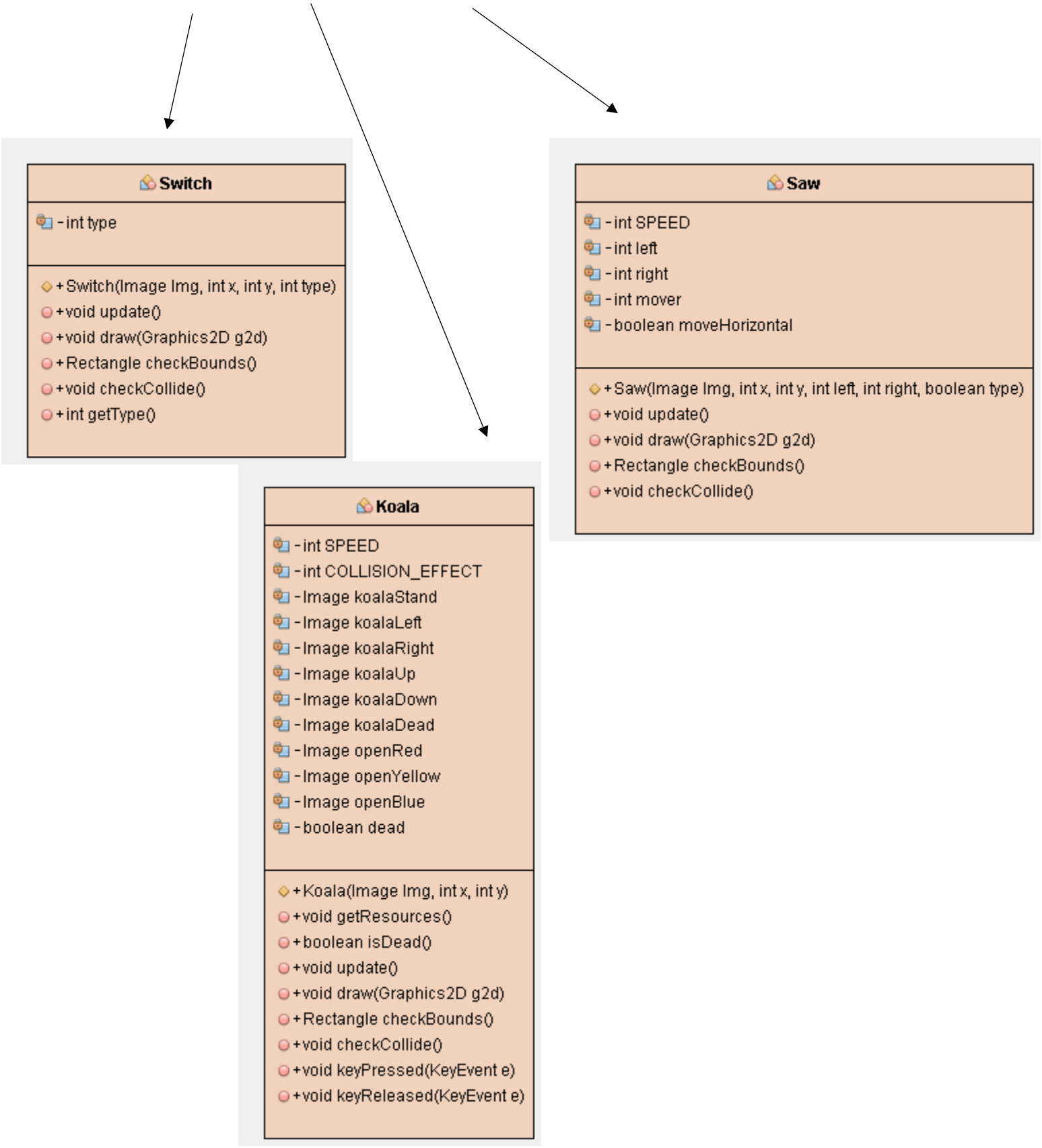
## KoalaGame.java:

```
private static ArrayList<Wall> wallsD = new ArrayList<>(); // Walls within game
private static ArrayList<Saw> saws = new ArrayList<>(); // Moving saw objects
private static ArrayList<Explosion> explosions = new ArrayList<>(); // Explosion objects
private static ArrayList<Explosives> explosives = new ArrayList<>(); // Explosive object:
private static ArrayList<Switch> closedSwitches = new ArrayList<>(); // Switch objects
private static ArrayList<Switch> openSwitches = new ArrayList<>(); // Switch objects
private static ArrayList<Rock> rocks = new ArrayList<>(); // Rock objects
private static ArrayList<Locks> locks = new ArrayList<>(); // Locks objects
private static ArrayList<Koala> deadKoalas = new ArrayList<>(); // Win game if this is er
private static ArrayList<Koala> koalas = new ArrayList<>(); // Koala objects
private static ArrayList<Koala> savedKoalas = new ArrayList<>(); // Saved koalas, Win if
private static Exit exit; // Exit object

/*** Images used in the game ***/
private Image background;
private Image congratulation;

private BufferedImage wallImg; // Contains the wall_tile image
private Image [] wallSprite; // Contains all 16 sub-images of the Wall_tile image
private Image koalaImg; // Initial koala image
private Image TNT; // Explosion/Bomb image
private Image rock;
private Image exitImg;

private Image closedRed; // Closed Red Switch
private Image closedYellow; // Closed Yellow Switch
private Image closedBlue; // Closed Blue Switch

private Image yellowLock;
private Image redLock;
private Image blueLock;

private Image sawHorizontalUp;
private Image sawHorizontalDown;
private Image sawVerticalLeft; // Not used
private Image sawVerticalRight;


public KoalaGame() {
    getResources();
    initalizeGame();
    initializeMap();

    addKeyListener(new Movement(koalas.get(0)));
    addKeyListener(new Movement(koalas.get(1)));
    addKeyListener(new Movement(koalas.get(2)));

    gameTimer = new Timer(10, this);
    gameTimer.start();
```

Koala Implement Chart




KoalaGame
- static Timer gameTimer
- ArrayList<Wall> walls
- static ArrayList<Wall> wallsD
- static ArrayList<Saw> saws
- static ArrayList<Explosion> explosions
- static ArrayList<Explosives> explosives
- static ArrayList<Switch> closedSwitches
- static ArrayList<Switch> openSwitches
- static ArrayList<Rock> rocks
- static ArrayList<Locks> locks
- static ArrayList<Koala> deadKoalas
- static ArrayList<Koala> koalas
- static ArrayList<Koala> savedKoalas
- static Exit exit
- Image background
- Image congratulation
- BufferedImage wallImg
- Image[] wallSprite
- Image koalaImg
- Image TNT
- Image rock
- Image exitImg
- Image closedRed
- Image closedYellow
- Image closedBlue
- Image yellowLock
- Image redLock
- Image blueLock
- Image sawHorizontalUp
- Image sawHorizontalDown
- Image sawVerticalLeft
- Image sawVerticalRight
+ KoalaGame()
+ void initalizeGame()
+ void initializeMap()
+ void getResources()
+ void paint(Graphics g)
+ void run()
+ void actionPerformed(ActionEvent e)
+ static ArrayList<Wall> getWalls()
+ static ArrayList<Saw> getSaws()
+ static void addDeadKoala(Koala koala)
+ static ArrayList<Koala> getKoala()
+ static void removeKoala(Koala koala)
+ static void saveKoala(Koala koala)
+ static ArrayList<Explosives> getExplosives()
+ static void removeExplosives(Explosives e)
+ static ArrayList<Locks> getLocks()
+ static void removeLock(Locks lock)
+ static ArrayList<Switch> getSwitches()
+ static void removeSwitch(Switch s)
+ static void addSwitch(Switch s)
+ static ArrayList<Rock> getRocks()
+ static void removeRock(Rock r)
+ static ArrayList<Explosion> getExplosions()
+ static void removeExplosion(Explosion e)
+ static void addExplosion(Explosion e)
+ static Exit getExit()
+ static void stopTimer()
+ static void main(String[] args)
Exit
+ Exit(Image img, int x, int y)
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Explosives
- Image explosion
+ Explosives(Image img, int x, int y)
+ void getResources()
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Wall
+ Wall(Image img, int x, int y)
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Rock
- Image explosion
+ Rock(Image img, int x, int y)
+ void getResources()
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Locks
+ Locks(Image img, int x, int y)
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Explosion
+ Explosion(Image img, int x, int y)
+ void update()
+ void draw(Graphics2D g2D)
+ Rectangle checkBounds()
+ void checkCollide()


Switch
- int type
+ Switch(Image Img, int x, int y, int type)
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
+ int getType()
Saw
- int SPEED
- int left
- int right
- int mover
- boolean moveHorizontal
+ Saw(Image Img, int x, int y, int left, int right, boolean type)
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
Koala
- int SPEED
- int COLLISION_EFFECT
- Image koalaStand
- Image koalaLeft
- Image koalaRight
- Image koalaUp
- Image koalaDown
- Image koalaDead
- Image openRed
- Image openYellow
- Image openBlue
- boolean dead
+ Koala(Image Img, int x, int y)
+ void getResources()
+ boolean isDead()
+ void update()
+ void draw(Graphics2D g2d)
+ Rectangle checkBounds()
+ void checkCollide()
+ void keyPressed(KeyEvent e)
+ void keyReleased(KeyEvent e)

## Bugs & Problems:

1) Tank Game tends to randomly freeze during gameplay, especially when tanks are moving or when one doesn't move for a period.
2) Collision detection in Tank Game for the Tank tends to make the images shake around since we are moving the image by 1 when collision detects.
3) We have problems about Collision in our Koala game. For example, sometimes when the Koala encounters a wall; instead of being stopped by the wall, the Koala will directly walk through the wall, as if there is nothing there. Another example would be, if three or more Koalas would collide into each other at the same time then one of the Koala will pop right through the wall and will go out of boundaries.
4) In the Koala game, our rock would only move to one side so we made our rock class very simple that koala can only move the rock in the right direction to avoid collision problems.

## What We Learned:

1) We become more familiar with going GitHub since we had multiple problems with merging in our codes when we would make similar changes in the games. Changes in the classes will most likely involve the main game such as the TankWars and KoalaGame classes.
2) It was a good experience modifying the Tank Game's code and reusing it for the Koala Game. This gave us the opportunity to learn about how we can take one function and use it in multiple ways. Also, we learned about different approaches of simply doing the same

thing. Our Tank Game sources turns out to be a complete mess, but generating the second game improved our codes and made it cleaner.

3) These games have helped us use object oriented programming skills on a large application, which we learned new skills, as well us refresh our old skills. Furthermore, we learned how to develop big applications from scratch because we remember, from previous classes that involved java, we were asked to modify and add to the code. However, in this class, we basically worked from nothing.
4) Both games also helped us become more familiar with Java library resources such as the implementation of Runnable, ActionListener, JPanel, and other libraries.

## Results & Conclusion:

1) There were certain several issues we faced in both game but we could overcome majority of the issues with teamwork. However, there are problems that we both did not know how to do and we must research them. Of course, some of the problems never got fixed, as mentioned above.
2) Our code would conflict with one another when we both would try to push our code back on GitHub and it would show us error because our code would overlap one another. So, we would copy the whole code first from GitHub and then push it back. Or sometimes we would see the changes what one of us made and the other one would just email the code to one person so we can easily push all our code back on GitHub without having any issues.

3) We had problems with the explosions and tank movements so we ended up making individuals classes for tanks. In order words, we made missile movement 1 and 2 classes, movement 1 and 2 classes, and tank 1 and tank 2 classes to avoid problems and to make our game work better.
4) Similar problems we encountered in the tank and koala game was the wall class. For some reason, the tank and the koalas would not recognize the wall and it would just go over it so we had to look at multiple similar codes online to finally figure out the problem and fix it.
5) Overall, both of our games our working good and one can simple play the Tank or the Koala game for entertainment anytime.

## Sources:


1. Neyland, Tom. "2D Java Game. Moving sprite above tiled images." Swing - 2D Java Game. Moving sprite above tiled images - Stack Overflow. N.p., 15 Nov. 2010. Web. 11 Aug. 2017.

2. Pek. "How can I play sound in Java?" Audio - How can I play sound in Java? - Stack Overflow. N.p., 25 Aug. 2008. Web. 07 Aug. 2017.