Even Semester (2022)

BINUS UNIVERSITY

BINUS INTERNATIONAL

OOP
Final Project
2048 Game

Student Information:

Name: Wilbert Wirawan Ichwan **NIM:** 2501963186

Course Code : ** COMP6699001 **Course Name : OOP

Class: L2CC **Lecturer:** Jude Joseph Lamug Martinez, MCS

Project Specification

- ### Objective

  The sole objective for my project is to create a game that is fun for everyone to play. The game is very simple and suitable for all ages to play.

- ### Description

  I recreated a game called “2048”. 2048 is a single-player puzzle video game. The entire objective of the game is just to combine blocks by moving them up, down, left, or right until it becomes 2048. It is somewhat of a puzzle game where you would need to think about your move carefully or else you would fail the game. You fail by filling the 4x4 box with numbers and you don’t have a move that you can do. The numbers are the power of two, so they are (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048).

- ### The Modules used

  For this project I used multiple modules. I used Java AWT to make the GUI of the game.I used Canvas, a canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user. I used Dimension, The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. The class is associated with certain properties of components. Several methods defined by the Component class and the LayoutManager interface return a Dimension object. I used random, to randomise the spawn of the blocks. I used swing JFrame The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, text fields are added to create a GUI.

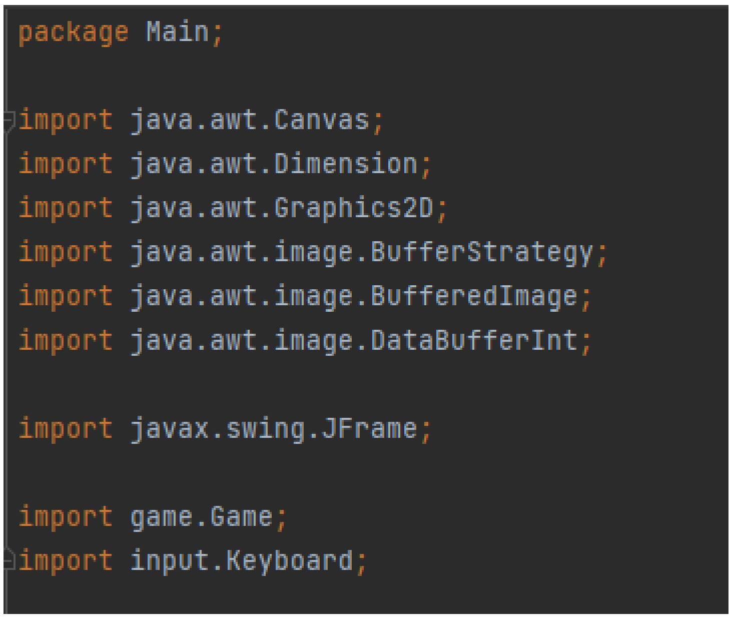
Project Design

I have 6 files for this project:

1. Main.java
2. Game.java
3. GameObject.java
4. Renderer.java
5. Sprite.java
6. Keyboard.java

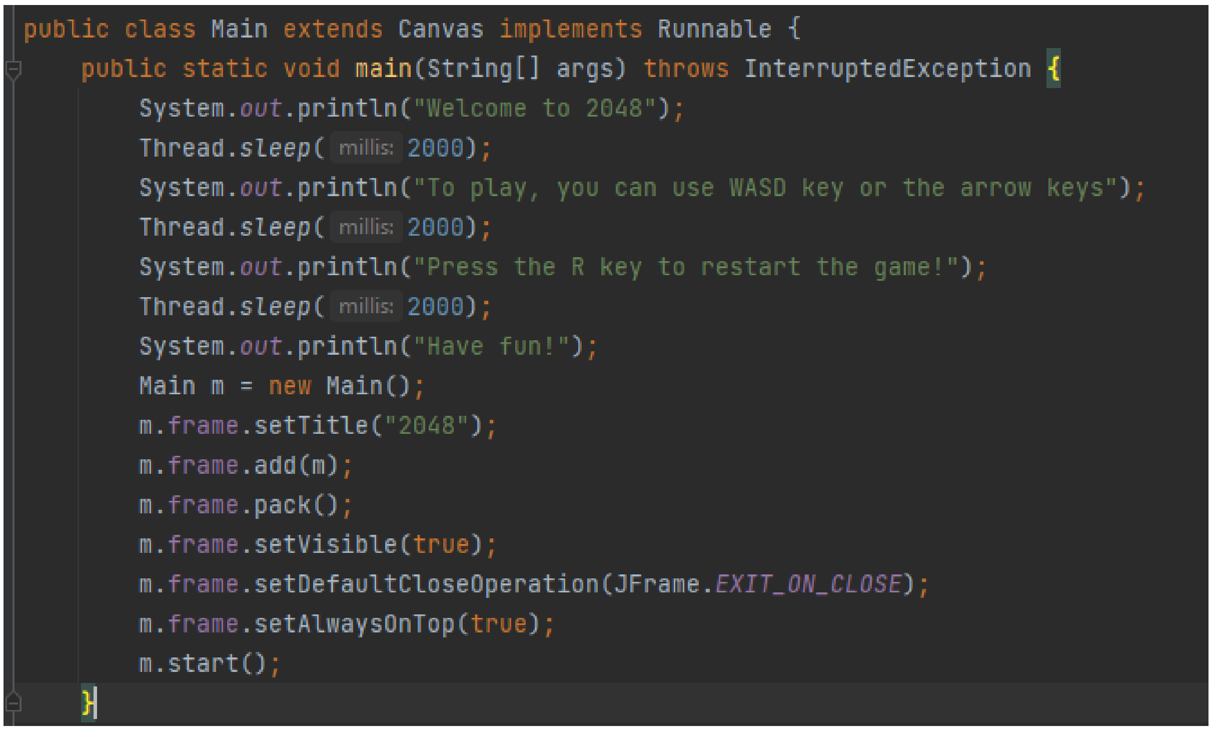
1. Main.java

This is the main file where you run the program.



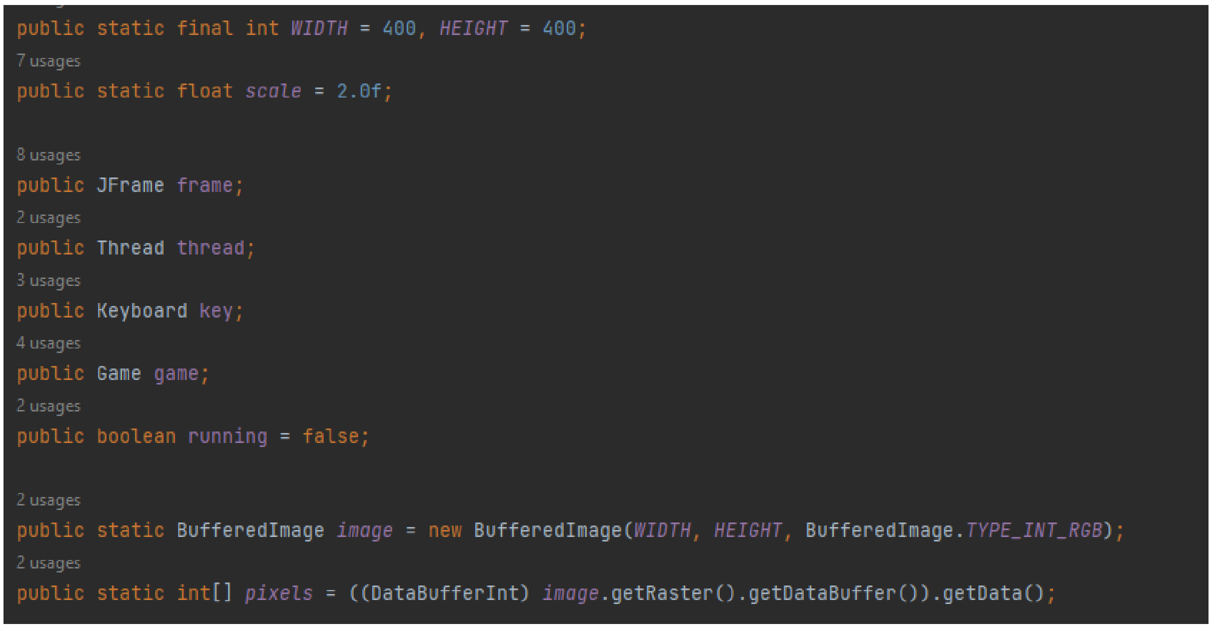
```
package Main;

import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;

import javax.swing.JFrame;

import game.Game;
import input.Keyboard;
```

The imports, all the modules I used in this file.



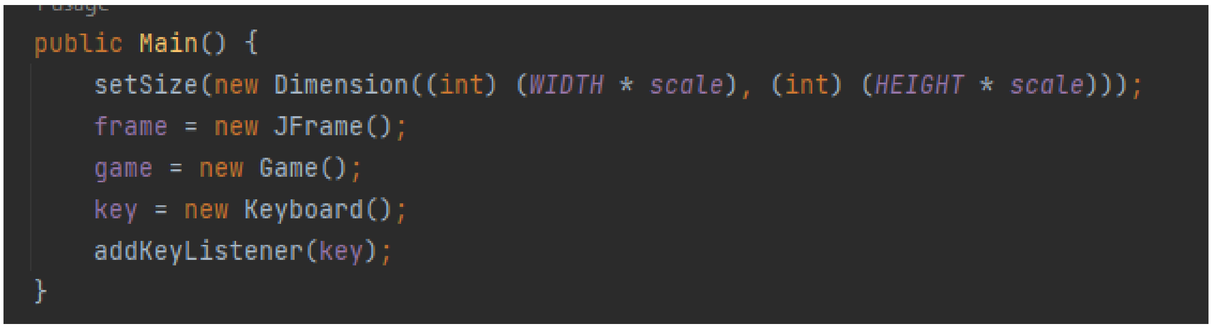
```java
public class Main extends Canvas implements Runnable {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Welcome to 2048");
        Thread.sleep( millis: 2000);
        System.out.println("To play, you can use WASD key or the arrow keys");
        Thread.sleep( millis: 2000);
        System.out.println("Press the R key to restart the game!");
        Thread.sleep( millis: 2000);
        System.out.println("Have fun!");
        Main m = new Main();
        m.frame.setTitle("2048");
        m.frame.add(m);
        m.frame.pack();
        m.frame.setVisible(true);
        m.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        m.frame.setAlwaysOnTop(true);
        m.start();
    }
```

The main run file, the tutorial and the JFrame settings.



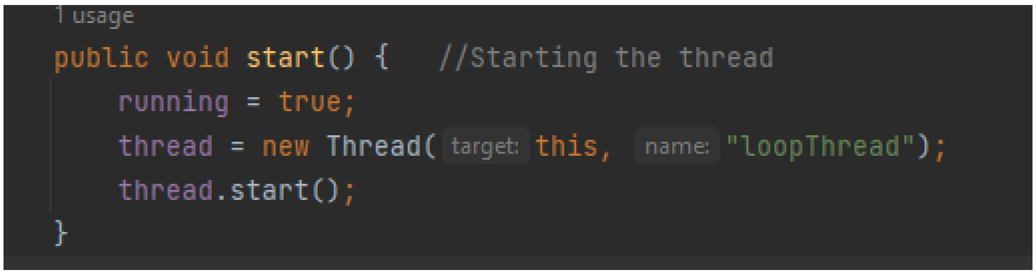
```java
public static final int WIDTH = 400, HEIGHT = 400;
7 usages
public static float scale = 2.0f;

8 usages
public JFrame frame;
2 usages
public Thread thread;
3 usages
public Keyboard key;
4 usages
public Game game;
2 usages
public boolean running = false;

2 usages
public static BufferedImage image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
2 usages
public static int[] pixels = ((DataBufferInt) image.getRaster().getDataBuffer()).getData();
```

The declarations.



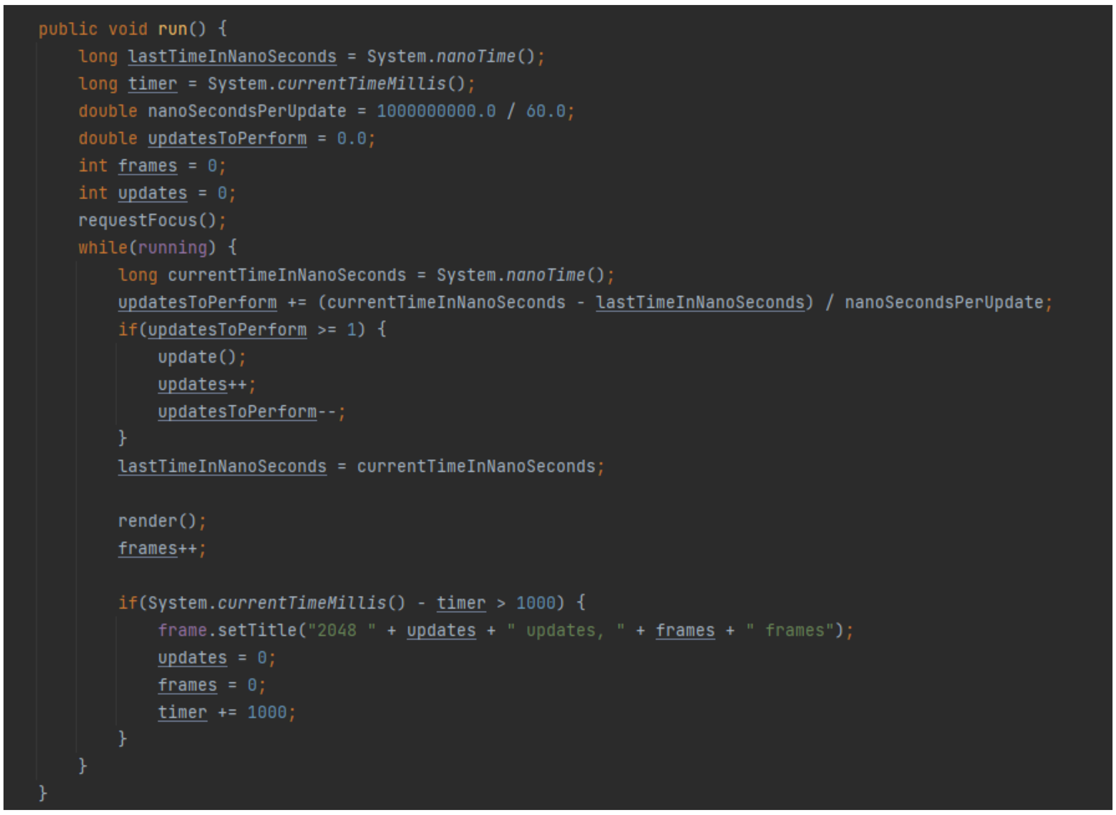
```java
public Main() {
    setSize(new Dimension((int) (WIDTH * scale), (int) (HEIGHT * scale)));
    frame = new JFrame();
    game = new Game();
    key = new Keyboard();
    addKeyListener(key);
}
```

The constructor.



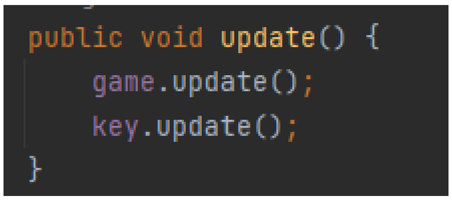
```java
1 usage
public void start() {   //Starting the thread
    running = true;
    thread = new Thread( target: this,  name: "loopThread");
    thread.start();
}
```

The start method, and starting the thread to run it.



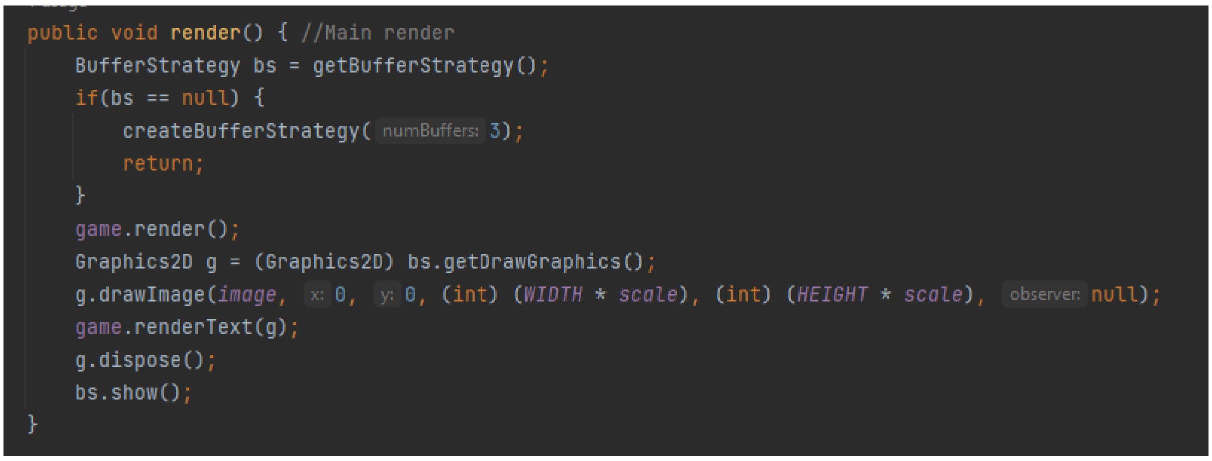
```java
public void run() {
    long lastTimeInNanoSeconds = System.nanoTime();
    long timer = System.currentTimeMillis();
    double nanoSecondsPerUpdate = 1000000000.0 / 60.0;
    double updatesToPerform = 0.0;
    int frames = 0;
    int updates = 0;
    requestFocus();
    while(running) {
        long currentTimeInNanoSeconds = System.nanoTime();
        updatesToPerform += (currentTimeInNanoSeconds - lastTimeInNanoSeconds) / nanoSecondsPerUpdate;
        if(updatesToPerform >= 1) {
            update();
            updates++;
            updatesToPerform--;
        }
        lastTimeInNanoSeconds = currentTimeInNanoSeconds;

        render();
        frames++;

        if(System.currentTimeMillis() - timer > 1000) {
            frame.setTitle("2048 " + updates + " updates, " + frames + " frames");
            updates = 0;
            frames = 0;
            timer += 1000;
        }
    }
}
```

The run method, basically it is going to call on updates 60 times a second and renders as many times a computer can handle. If the updates to perform are bigger than 1 then, the update is going to be run and adds the updates variable. Then its going to replace the last time in nanoseconds with the current one because in the next iteration it the current one is going to be the last one by then.



```java
public void update() {
    game.update();
    key.update();
}
```

Update.

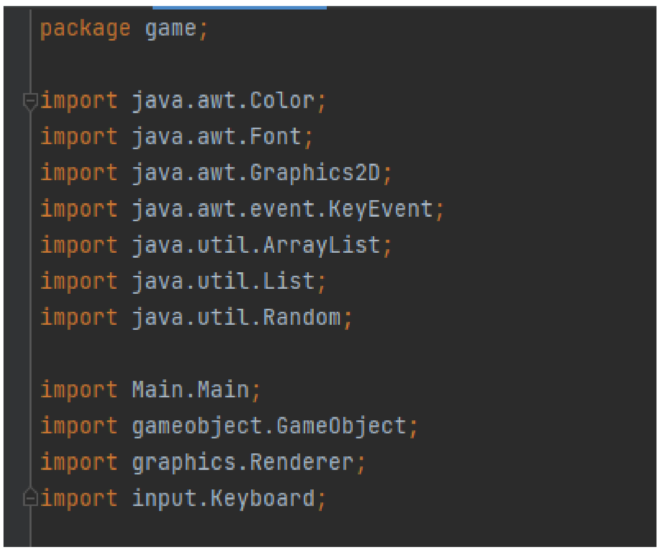


```java
public void render() { //Main render
    BufferStrategy bs = getBufferStrategy();
    if(bs == null) {
        createBufferStrategy( numBuffers: 3);
        return;
    }
    game.render();
    Graphics2D g = (Graphics2D) bs.getDrawGraphics();
    g.drawImage(image,  x: 0,  y: 0, (int) (WIDTH * scale), (int) (HEIGHT * scale),  observer: null);
    game.renderText(g);
    g.dispose();
    bs.show();
}
```

Render the game with bufferstrategy. BufferStrategy is the mechanism with which to organize complex memory on a particular Canvas or Window. Then I created 3 buffers. Then call game to render. Then I used Graphics2D to "draw" the game.

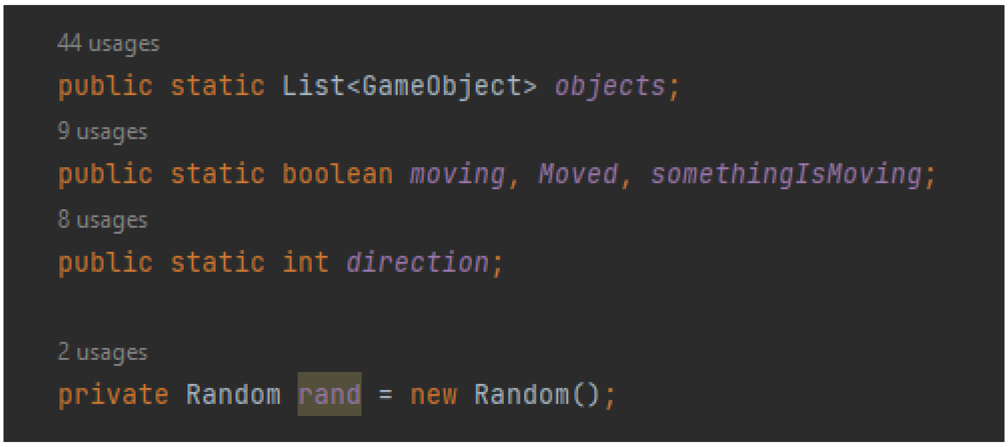
2. Game.java

This is the file where the game logic is.



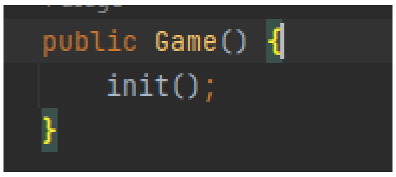
```java
package game;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import Main.Main;
import gameobject.GameObject;
import graphics.Renderer;
import input.Keyboard;
```

The imports of the modules.



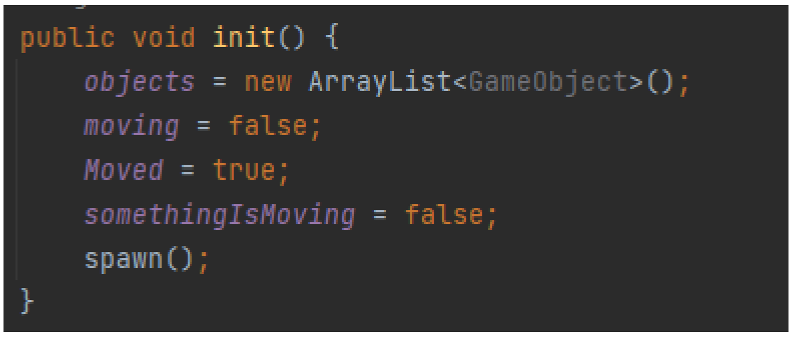
```
44 usages
public static List<GameObject> objects;
9 usages
public static boolean moving, Moved, somethingIsMoving;
8 usages
public static int direction;

2 usages
private Random rand = new Random();
```

Declarations.



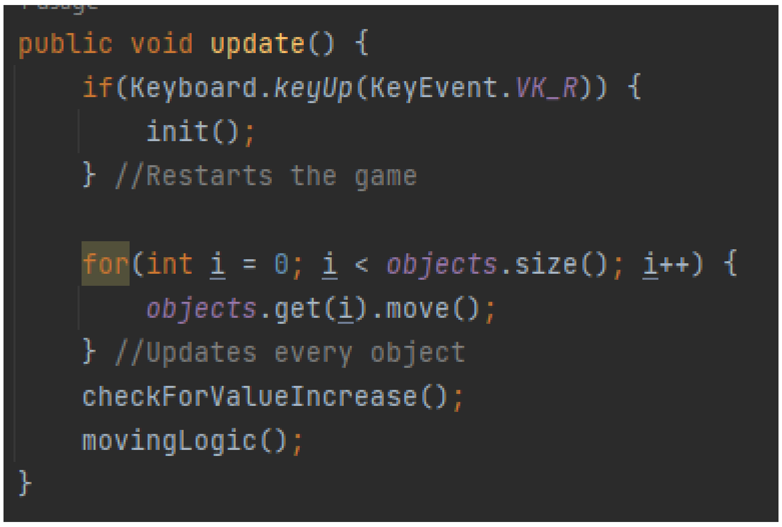
```
public Game() {
    init();
}
```

Initializing the game.



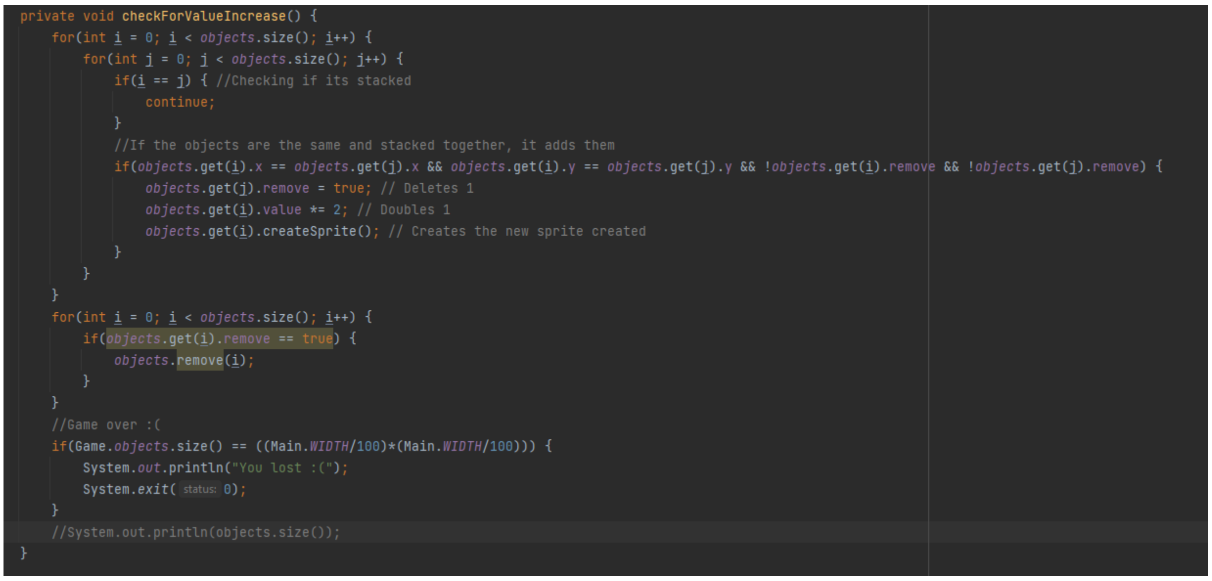
```
public void init() {
    objects = new ArrayList<GameObject>();
    moving = false;
    Moved = true;
    somethingIsMoving = false;
    spawn();
}
```

The init method.



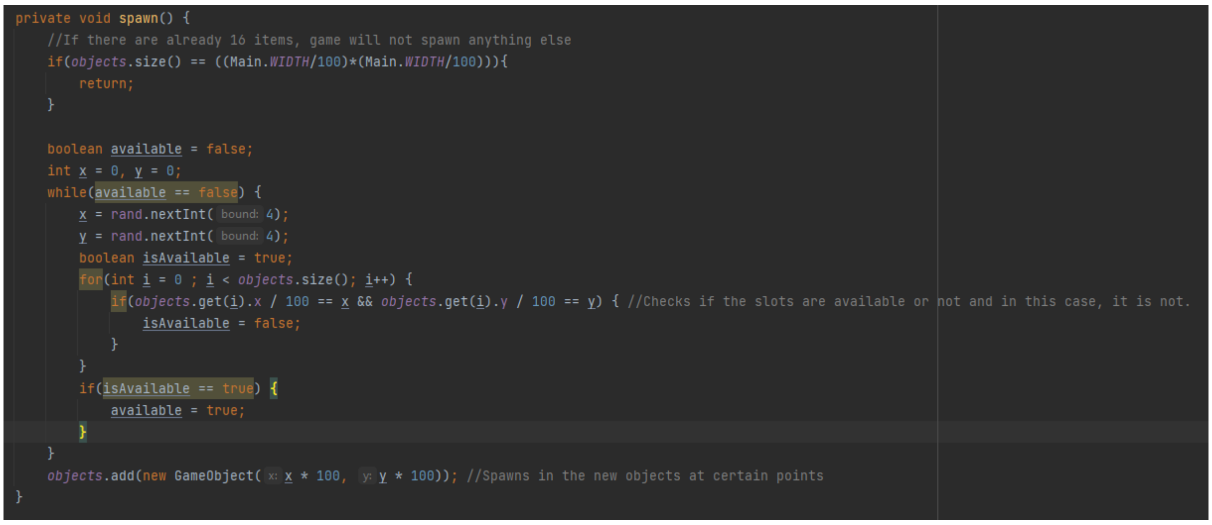
```java
public void update() {
    if(Keyboard.keyUp(KeyEvent.VK_R)) {
        init();
    } //Restarts the game

    for(int i = 0; i < objects.size(); i++) {
        objects.get(i).move();
    } //Updates every object
    checkForValueIncrease();
    movingLogic();
}
```

Update method. Key event where if the player presses R it resets the board by re-initializing the game. And the for loop is for updating every object then it calls on the checkForValueIncrease() and the movingLogic() methods.



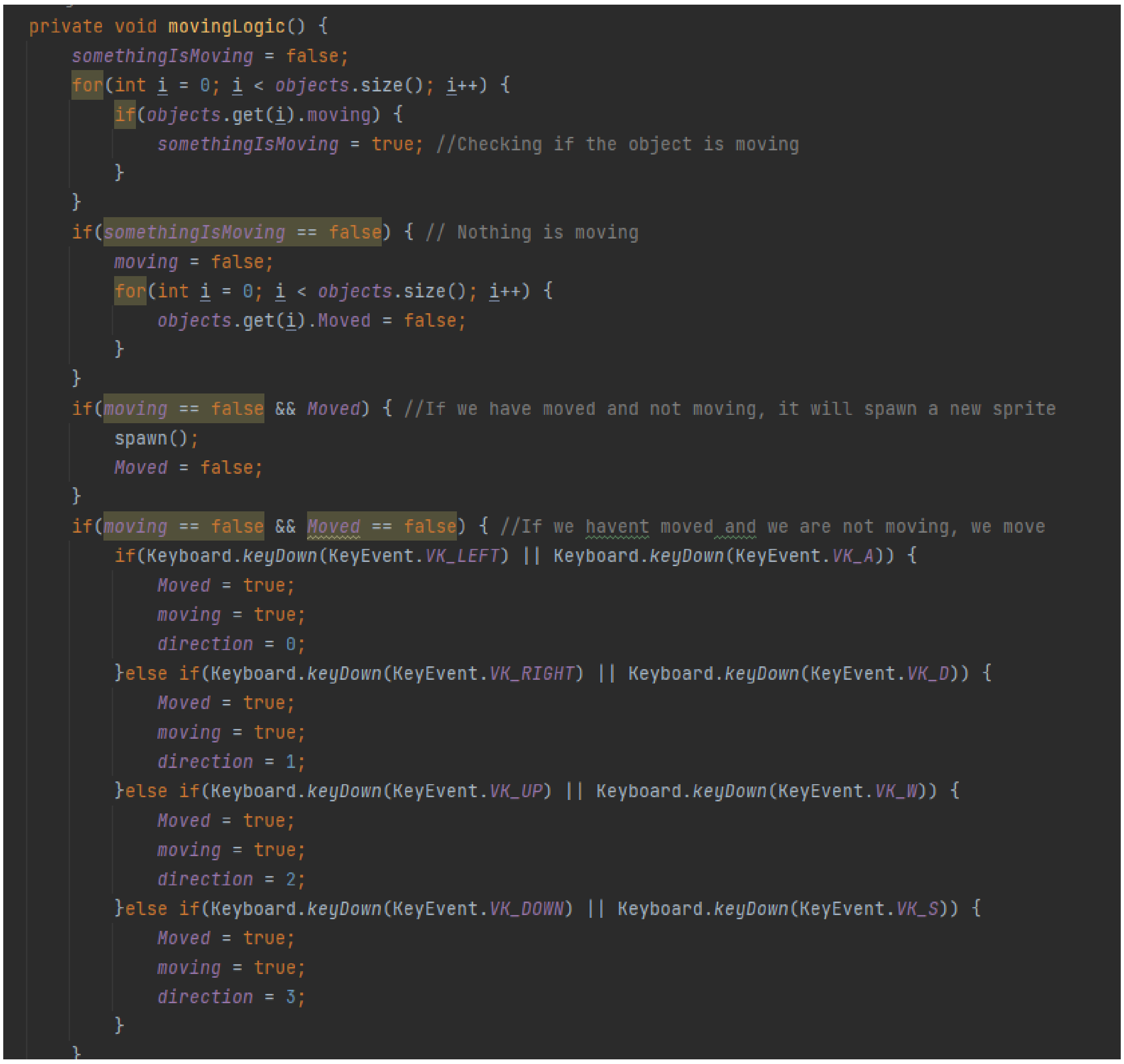
```java
private void checkForValueIncrease() {
    for(int i = 0; i < objects.size(); i++) {
        for(int j = 0; j < objects.size(); j++) {
            if(i == j) { //Checking if its stacked
                continue;
            }
            //If the objects are the same and stacked together, it adds them
            if(objects.get(i).x == objects.get(j).x && objects.get(i).y == objects.get(j).y && !objects.get(i).remove && !objects.get(j).remove) {
                objects.get(j).remove = true; // Deletes 1
                objects.get(i).value *= 2; // Doubles 1
                objects.get(i).createSprite(); // Creates the new sprite created
            }
        }
    }
    for(int i = 0; i < objects.size(); i++) {
        if(objects.get(i).remove == true) {
            objects.remove(i);
        }
    }
    //Game over :(
    if(Game.objects.size() == ((Main.WIDTH/100)*(Main.WIDTH/100))) {
        System.out.println("You lost :(");
        System.exit( status: 0);
    }
    //System.out.println(objects.size());
}
```

This is one of the main methods behind the game logic itself. It basically checks if the objects that are next to each other can merge and double the value of it. The 1st to for loops is to check if they are indeed the same number and are stacked. The if function is for checking the state of the object. If it is the same then it removes one of them and doubles the value of one of them making it seem like it has merged. The for loop after that is to remove said object and the last if function is to check if the board is completely filled or not. If it is, then the game has ended.



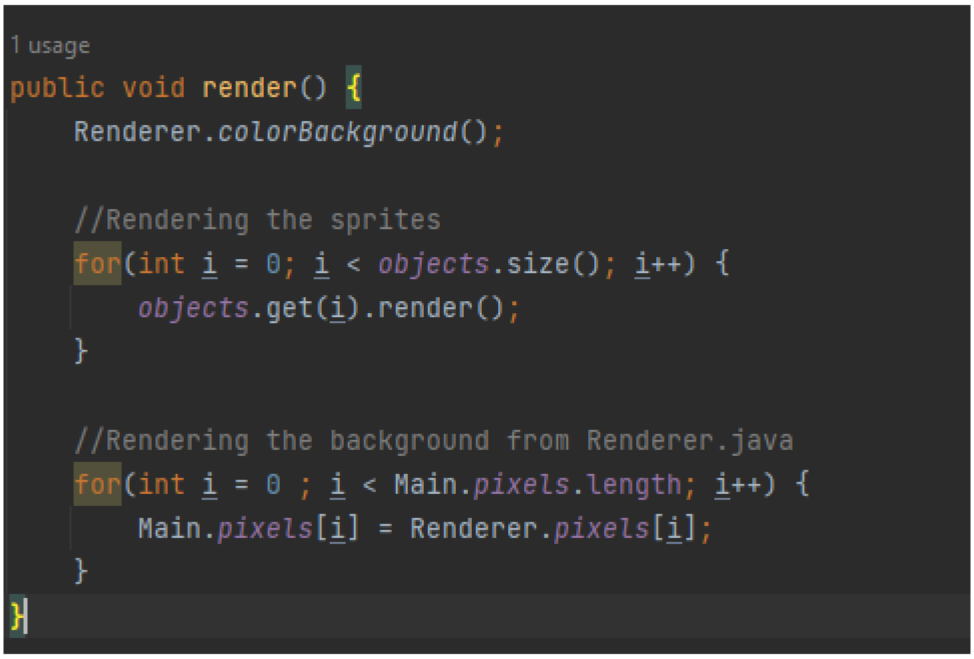
```java
private void spawn() {
    //If there are already 16 items, game will not spawn anything else
    if(objects.size() == ((Main.WIDTH/100)*(Main.WIDTH/100))){
        return;
    }

    boolean available = false;
    int x = 0, y = 0;
    while(available == false) {
        x = rand.nextInt( bound: 4);
        y = rand.nextInt( bound: 4);
        boolean isAvailable = true;
        for(int i = 0 ; i < objects.size(); i++) {
            if(objects.get(i).x / 100 == x && objects.get(i).y / 100 == y) { //Checks if the slots are available or not and in this case, it is not.
                isAvailable = false;
            }
        }
        if(isAvailable == true) {
            available = true;
        }
    }
    objects.add(new GameObject( x: x * 100,  y: y * 100)); //Spawns in the new objects at certain points
}
```

This is the spawn method. First, it checks if the board is filled or not, if it is then it obviously can't spawn anymore blocks. Then it declares available as false and x, y. The while loop is to check every slot's state if they are available or not, if it is then it will spawn a random object at that index.



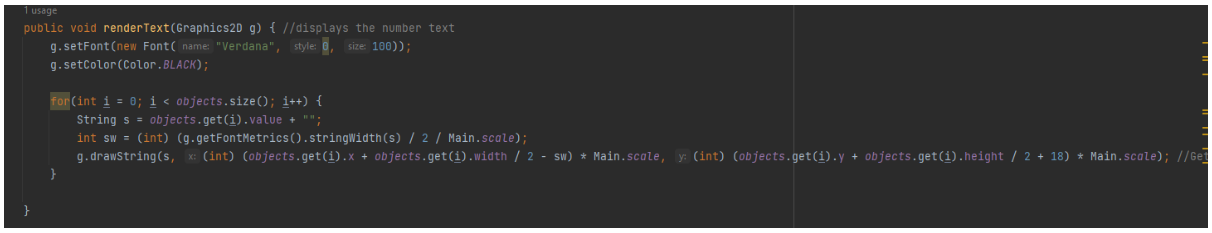
```
private void movingLogic() {
    somethingIsMoving = false;
    for(int i = 0; i < objects.size(); i++) {
        if(objects.get(i).moving) {
            somethingIsMoving = true; //Checking if the object is moving
        }
    }
    if(somethingIsMoving == false) { // Nothing is moving
        moving = false;
        for(int i = 0; i < objects.size(); i++) {
            objects.get(i).Moved = false;
        }
    }
    if(moving == false && Moved) { //If we have moved and not moving, it will spawn a new sprite
        spawn();
        Moved = false;
    }
    if(moving == false && Moved == false) { //If we havent moved and we are not moving, we move
        if(Keyboard.keyDown(KeyEvent.VK_LEFT) || Keyboard.keyDown(KeyEvent.VK_A)) {
            Moved = true;
            moving = true;
            direction = 0;
        }else if(Keyboard.keyDown(KeyEvent.VK_RIGHT) || Keyboard.keyDown(KeyEvent.VK_D)) {
            Moved = true;
            moving = true;
            direction = 1;
        }else if(Keyboard.keyDown(KeyEvent.VK_UP) || Keyboard.keyDown(KeyEvent.VK_W)) {
            Moved = true;
            moving = true;
            direction = 2;
        }else if(Keyboard.keyDown(KeyEvent.VK_DOWN) || Keyboard.keyDown(KeyEvent.VK_S)) {
            Moved = true;
            moving = true;
            direction = 3;
        }
    }
```

The moving logic behind the game. The 1st for loop is to check is something is moving or not. The if method after that is to say that if something is not moving then moving is false and then it will do a for loop to say that every slot has not moved. The if function after that is to spawn objects. If it isnt moving but it has moved then it will spawn an object and say that is has not moved. The next if statement is for us to move the objects around if nothing is moving and nothing has moved. Each of those if statements are checking for key presses, W or up arrow to move up, A or left arrow to move left, S or down arrow is to move down and D or right arrow is to move right.



```java
1 usage
public void render() {
    Renderer.colorBackground();

    //Rendering the sprites
    for(int i = 0; i < objects.size(); i++) {
        objects.get(i).render();
    }

    //Rendering the background from Renderer.java
    for(int i = 0 ; i < Main.pixels.length; i++) {
        Main.pixels[i] = Renderer.pixels[i];
    }
}
```

Renders the sprites and the background color. For loops to render each slot and renders each pixels in the board.

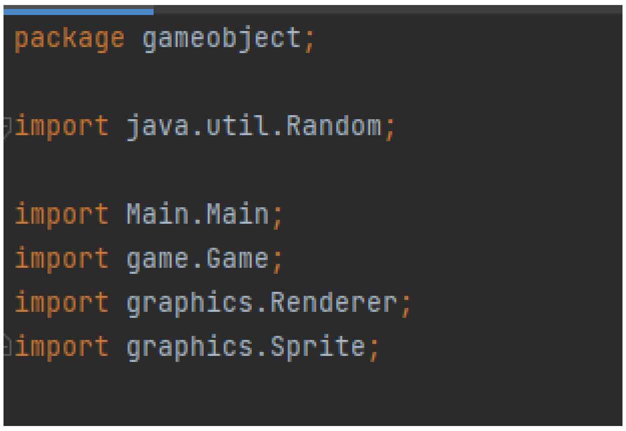


```java
1 usage
public void renderText(Graphics2D g) { //displays the number text
    g.setFont(new Font( name: "Verdana",  style: 0,  size: 100));
    g.setColor(Color.BLACK);

    for(int i = 0; i < objects.size(); i++) {
        String s = objects.get(i).value + "";
        int sw = (int) (g.getFontMetrics().stringWidth(s) / 2 / Main.scale);
        g.drawString(s,  x: (int) (objects.get(i).x + objects.get(i).width / 2 - sw) * Main.scale,  y: (int) (objects.get(i).y + objects.get(i).height / 2 + 18) * Main.scale); //Get
    }

}
```

To show the number in each block. I set the font settings like the font, the size, and the color. Then the for loop is to draw the string of each block depending on its value and coordinate.

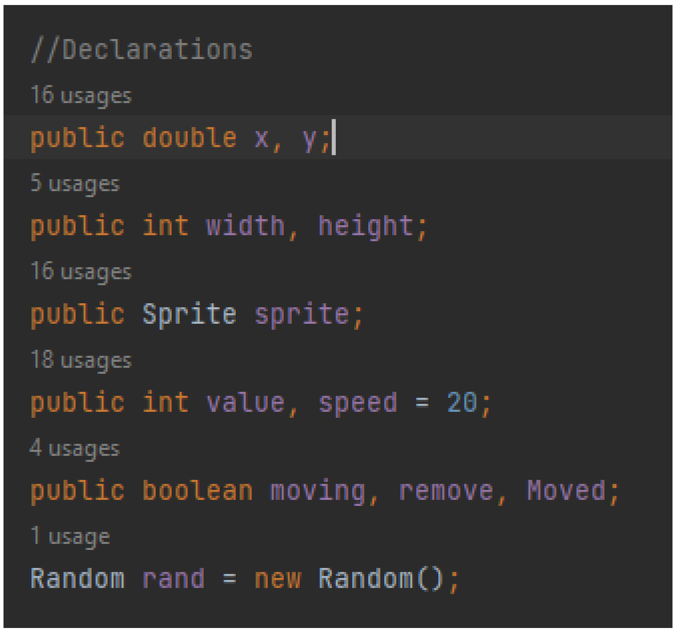
3. GameObject.java

This file is for the objects of the game.



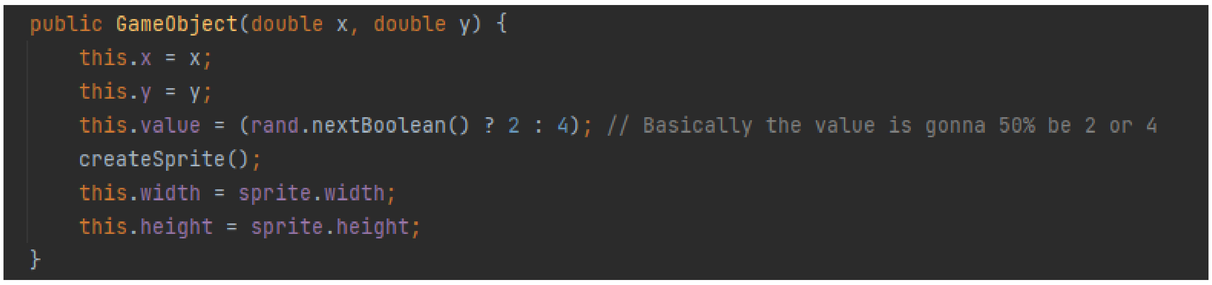
```java
package gameobject;

import java.util.Random;

import Main.Main;
import game.Game;
import graphics.Renderer;
import graphics.Sprite;
```

Imports of the modules.



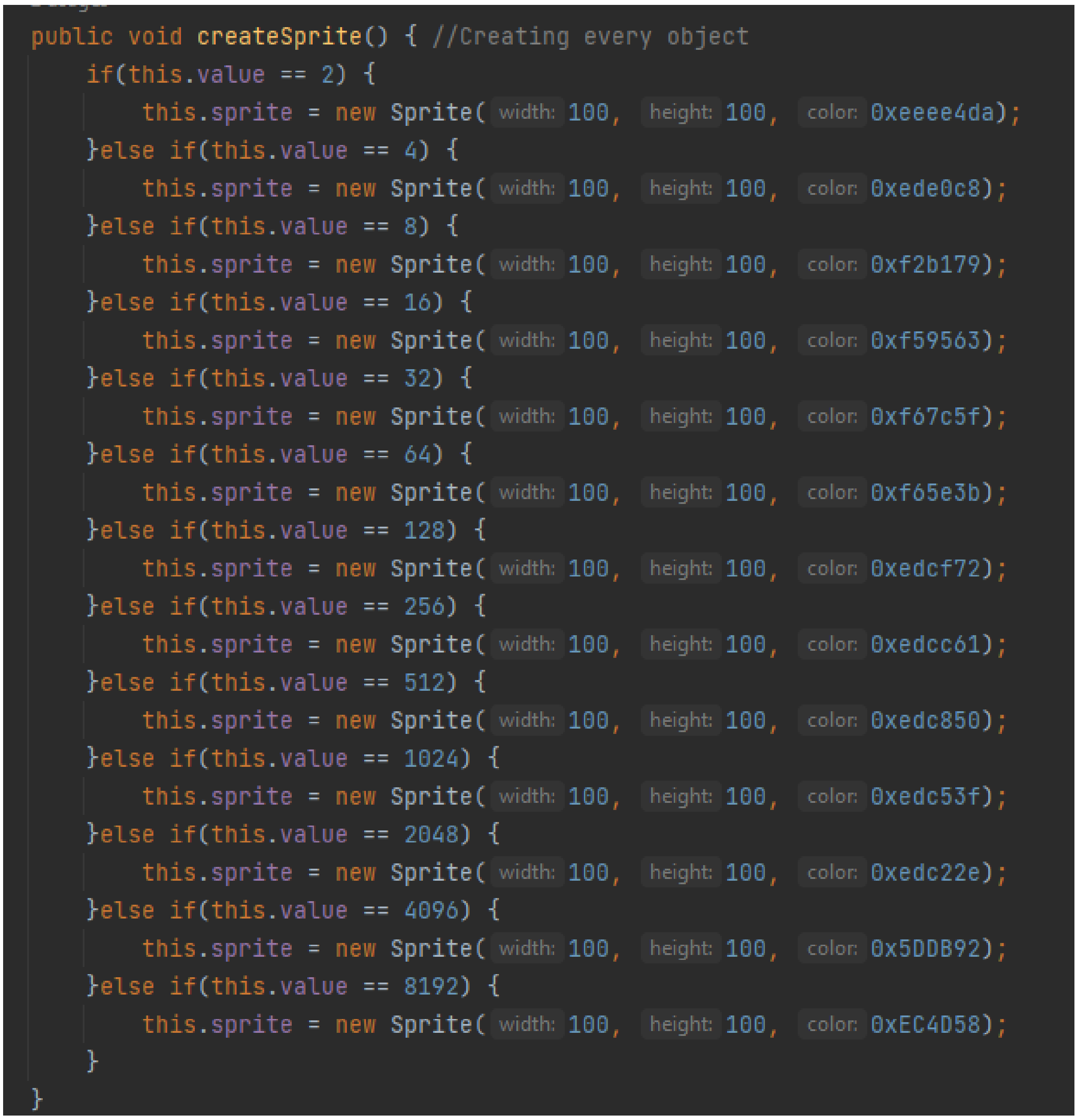
```java
//Declarations
16 usages
public double x, y;
5 usages
public int width, height;
16 usages
public Sprite sprite;
18 usages
public int value, speed = 20;
4 usages
public boolean moving, remove, Moved;
1 usage
Random rand = new Random();
```

Declarations.



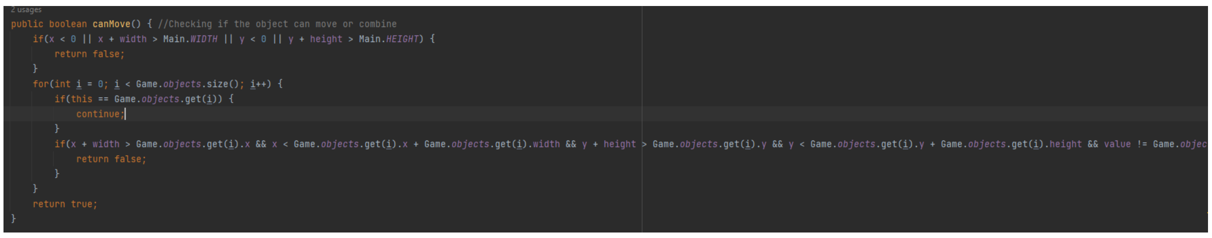
```java
public GameObject(double x, double y) {
    this.x = x;
    this.y = y;
    this.value = (rand.nextBoolean() ? 2 : 4); // Basically the value is gonna 50% be 2 or 4
    createSprite();
    this.width = sprite.width;
    this.height = sprite.height;
}
```

The constructor and the this.value one its to randomize the object spawn 50% to spawn 2 and 50% to spawn 4.



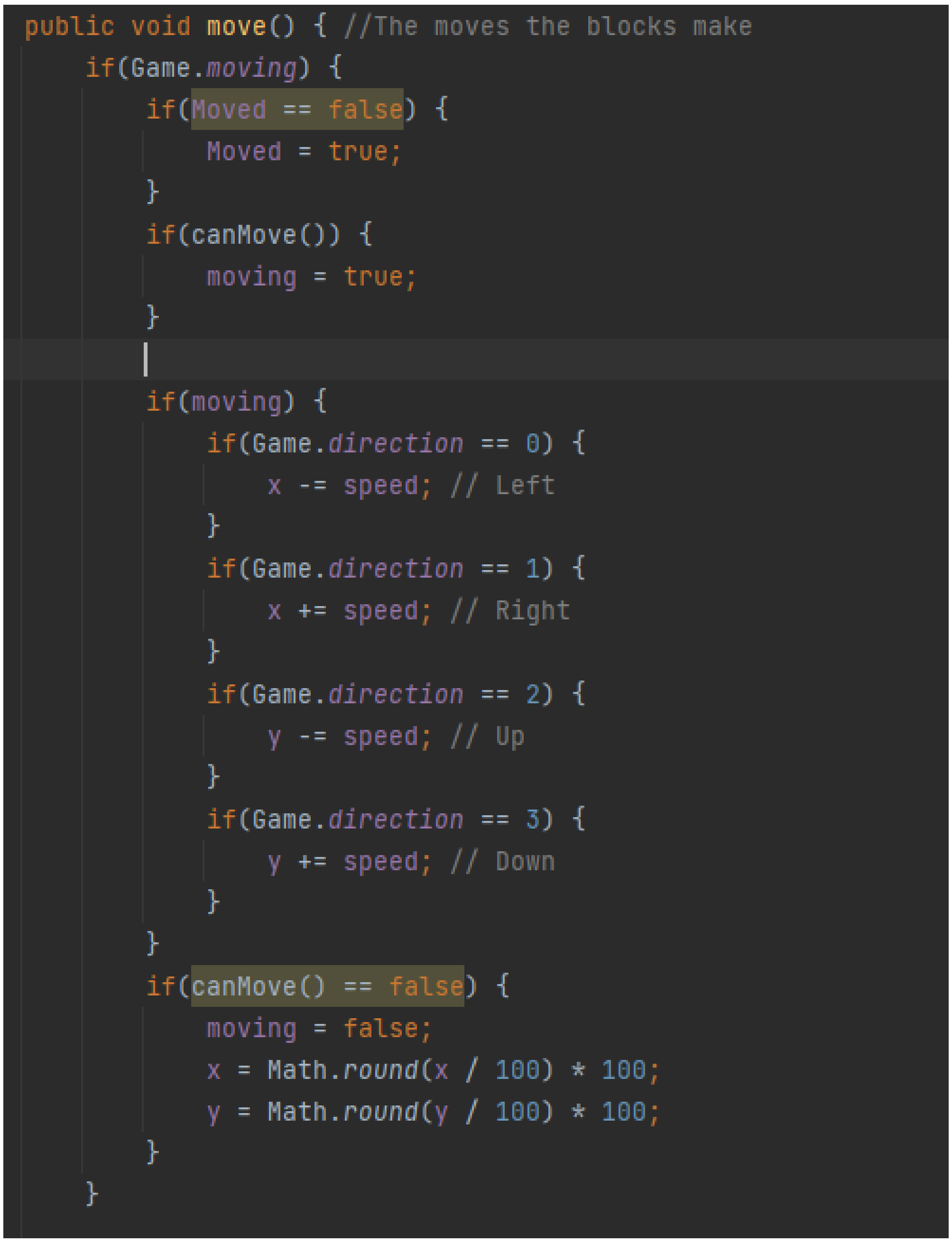
```java
public void createSprite() { //Creating every object
    if(this.value == 2) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xeeee4da);
    }else if(this.value == 4) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xede0c8);
    }else if(this.value == 8) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xf2b179);
    }else if(this.value == 16) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xf59563);
    }else if(this.value == 32) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xf67c5f);
    }else if(this.value == 64) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xf65e3b);
    }else if(this.value == 128) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xedcf72);
    }else if(this.value == 256) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xedcc61);
    }else if(this.value == 512) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xedc850);
    }else if(this.value == 1024) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xedc53f);
    }else if(this.value == 2048) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xedc22e);
    }else if(this.value == 4096) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0x5DDB92);
    }else if(this.value == 8192) {
        this.sprite = new Sprite( width: 100, height: 100, color: 0xEC4D58);
    }
}
```

Creating the sprites for each value with its own unique color and with a 100x100 size



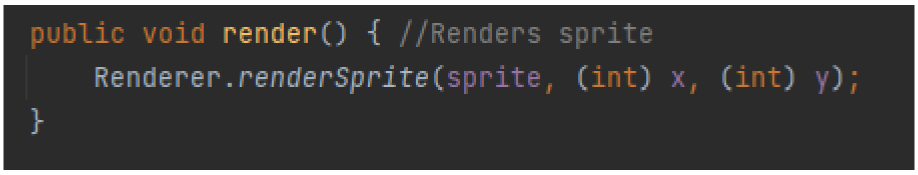
```java
2 usages
public boolean canMove() { //Checking if the object can move or combine
    if(x < 0 || x + width > Main.WIDTH || y < 0 || y + height > Main.HEIGHT) {
        return false;
    }
    for(int i = 0; i < Game.objects.size(); i++) {
        if(this == Game.objects.get(i)) {
            continue;
        }
        if(x + width > Game.objects.get(i).x && x < Game.objects.get(i).x + Game.objects.get(i).width && y + height > Game.objects.get(i).y && y < Game.objects.get(i).y + Game.objects.get(i).height && value != Game.objec
            return false;
        }
    }
    return true;
}
```

This method is to check if the block can move or not. The first if statement is to check if the blocks are outside the grid or not. The long if statement is checking if the value are the same or not and if it can merge with each other or not.



```java
public void move() { //The moves the blocks make
    if(Game.moving) {
        if(Moved == false) {
            Moved = true;
        }
        if(canMove()) {
            moving = true;
        }

        if(moving) {
            if(Game.direction == 0) {
                x -= speed; // Left
            }
            if(Game.direction == 1) {
                x += speed; // Right
            }
            if(Game.direction == 2) {
                y -= speed; // Up
            }
            if(Game.direction == 3) {
                y += speed; // Down
            }
        }
        if(canMove() == false) {
            moving = false;
            x = Math.round(x / 100) * 100;
            y = Math.round(y / 100) * 100;
        }
    }
```

If the block is moving, it checks the direction it goes to and moves it there. If it cant move then moving is false and sets the index.

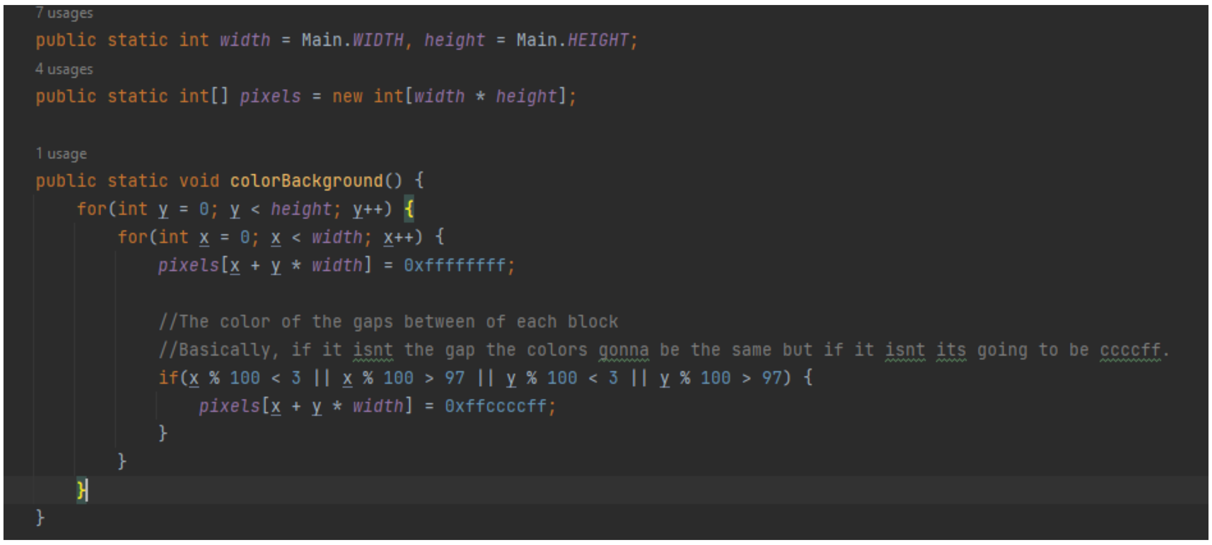


```java
public void render() { //Renders sprite
    Renderer.renderSprite(sprite, (int) x, (int) y);
}
```

Renders the sprite.

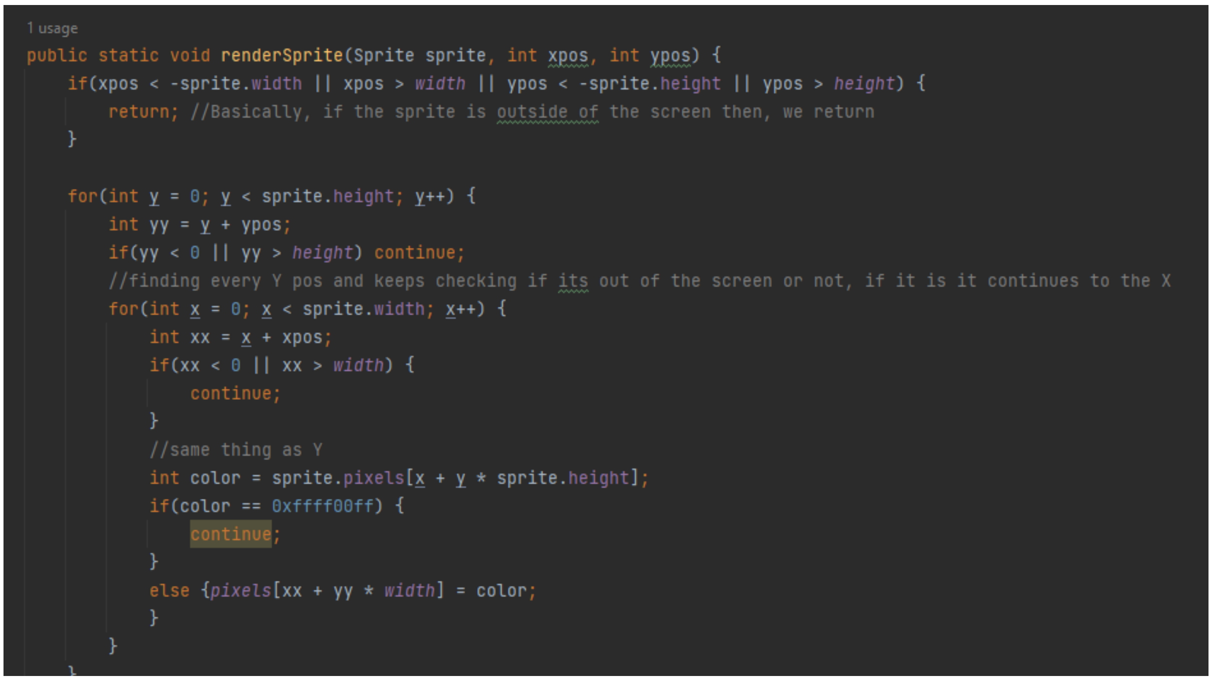
4. Renderer.java

This file is to render the background and the sprites of the game (the objects)



```
7 usages
public static int width = Main.WIDTH, height = Main.HEIGHT;
4 usages
public static int[] pixels = new int[width * height];

1 usage
public static void colorBackground() {
    for(int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            pixels[x + y * width] = 0xffffffff;

            //The color of the gaps between of each block
            //Basically, if it isnt the gap the colors gonna be the same but if it isnt its going to be ccccff.
            if(x % 100 < 3 || x % 100 > 97 || y % 100 < 3 || y % 100 > 97) {
                pixels[x + y * width] = 0xffccccff;
            }
        }
    }
}
```

The first 2 for loops is to set the color of each of the slot / blocks in the background. The last if statement is to check for the gaps between the numbers and change it to that color.



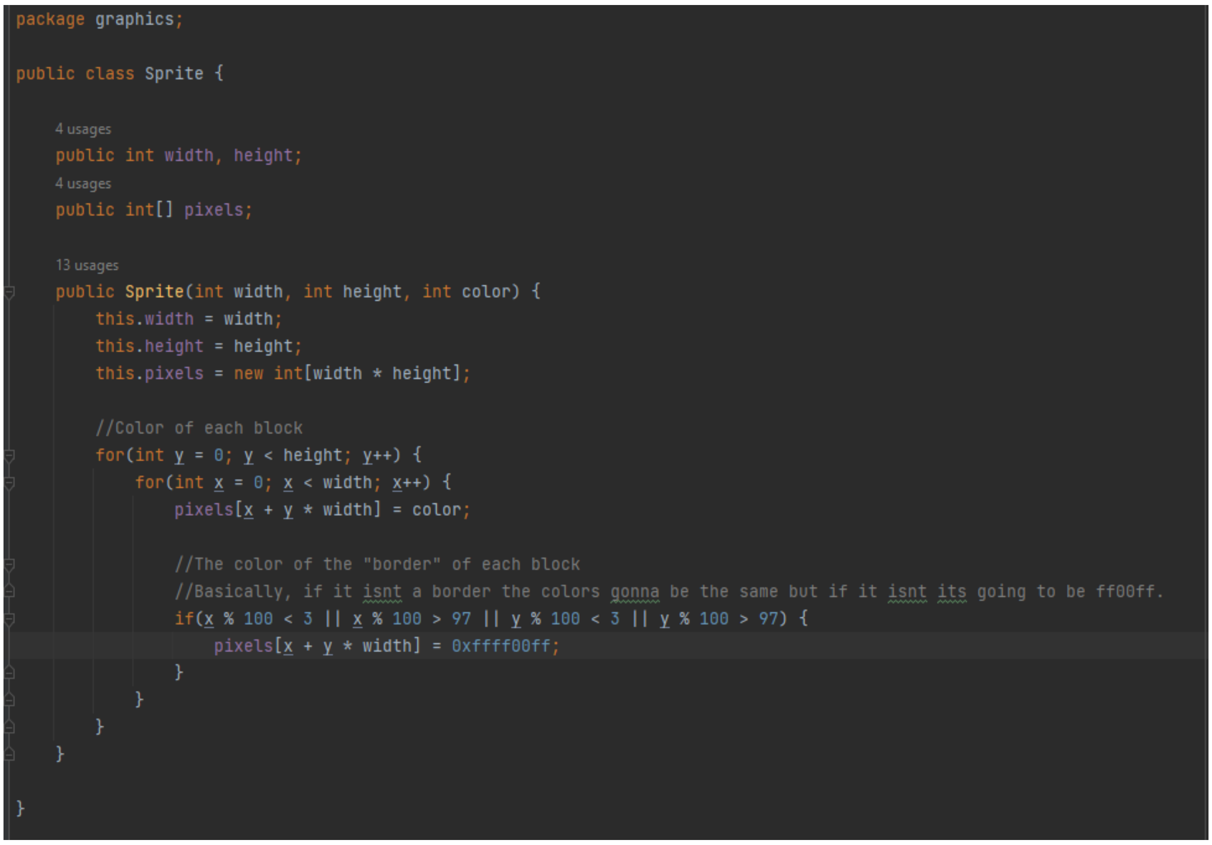
```
1 usage
public static void renderSprite(Sprite sprite, int xpos, int ypos) {
    if(xpos < -sprite.width || xpos > width || ypos < -sprite.height || ypos > height) {
        return; //Basically, if the sprite is outside of the screen then, we return
    }

    for(int y = 0; y < sprite.height; y++) {
        int yy = y + ypos;
        if(yy < 0 || yy > height) continue;
        //finding every Y pos and keeps checking if its out of the screen or not, if it is it continues to the X
        for(int x = 0; x < sprite.width; x++) {
            int xx = x + xpos;
            if(xx < 0 || xx > width) {
                continue;
            }
            //same thing as Y
            int color = sprite.pixels[x + y * sprite.height];
            if(color == 0xffff00ff) {
                continue;
            }
            else {pixels[xx + yy * width] = color;
            }
        }
    }
```

The 1st if statement checks if the sprite is outside of the grid or not.

The next for loop is finding every Y pos and keeps checking if its out of the screen or not, if it is it continues to the X and then it does the same for the X. Then it checks if the color is the same then it keeps it, if it isnt then it changes it.

5. Sprite.java

This file is behind the sprites, in the file its basically just behind the coloration of the game.

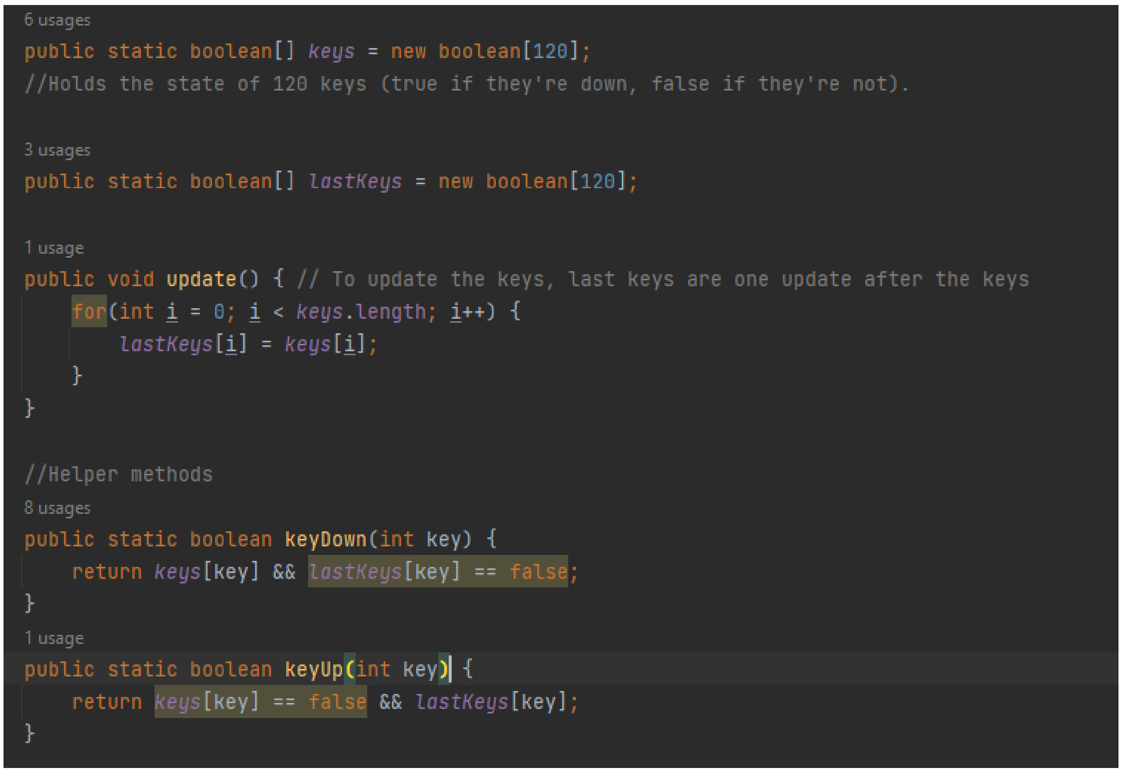


```java
package graphics;

public class Sprite {

    4 usages
    public int width, height;
    4 usages
    public int[] pixels;

    13 usages
    public Sprite(int width, int height, int color) {
        this.width = width;
        this.height = height;
        this.pixels = new int[width * height];

        //Color of each block
        for(int y = 0; y < height; y++) {
            for(int x = 0; x < width; x++) {
                pixels[x + y * width] = color;

                //The color of the "border" of each block
                //Basically, if it isnt a border the colors gonna be the same but if it isnt its going to be ff00ff.
                if(x % 100 < 3 || x % 100 > 97 || y % 100 < 3 || y % 100 > 97) {
                    pixels[x + y * width] = 0xffff00ff;
                }
            }
        }
    }

}
```

The for loops colors each of the block and then it has an if statement checks for the border of the blocks and changes it if it is a border.

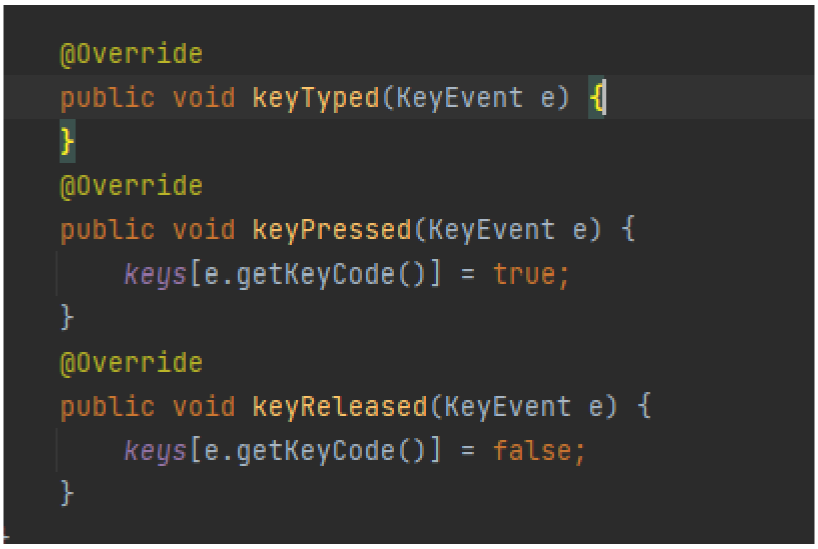
6. Keyboard.java

This file is behind the key presses, key updates, etc.



```
6 usages
public static boolean[] keys = new boolean[120];
//Holds the state of 120 keys (true if they're down, false if they're not).

3 usages
public static boolean[] lastKeys = new boolean[120];

1 usage
public void update() { // To update the keys, last keys are one update after the keys
    for(int i = 0; i < keys.length; i++) {
        lastKeys[i] = keys[i];
    }
}

//Helper methods
8 usages
public static boolean keyDown(int key) {
    return keys[key] && lastKeys[key] == false;
}
1 usage
public static boolean keyUp(int key) {
    return keys[key] == false && lastKeys[key];
}
```

This is to check the state of the keys in the keyboard. The first one is to update the keys of the keyboard. The next methods check if the key is pressed (down) or up (released) and returns the boolean



```
@Override
public void keyTyped(KeyEvent e) {
}
@Override
public void keyPressed(KeyEvent e) {
    keys[e.getKeyCode()] = true;
}
@Override
public void keyReleased(KeyEvent e) {
    keys[e.getKeyCode()] = false;
}
```

The default methods of keypresses. keyPressed is invoked when a key has been pressed. keyReleased is invoked when a key has been released. keyTyped is invoked when a key has been typed.