Project Documentation For Project 2 “Block & Key”

Author: Justin Sumrall

Date: December 10, 2021

Table of Contents

List of Figures

1. Project Overview

A continuation of Project 1, this project aims to further polish and improve on the previous iteration with additional playable levels and added game functionality.

2. Project Requirements

This project must achieve the following objectives:

- **Adding 4 additional levels, for a total of 5.**
- **Improve collision handling such that the player cannot easily escape the bounds of the level.**
- **Improve the player health system to allow a delay between taking damage and being able to take damage again.**
- **Update player and bat enemy sprite assets.**

2.1. Derived Requirements

- **Add visual indicator to show how long the player is invulnerable after taking damage.**
- **The main loop must be able to remove assets from previous levels and load assets for the next level upon clearing a level.**

3. Design Plans

A few new classes were created to fulfill the requirements of this project: InvulnTime.java and DamageFlicker.java. In addition, the TileGrid.java, Enemy.java, and Player.java classes received updates to implement the desired additional functionality outlined in this document.

3.1. Class Additions and Modifications

The design of the implementations of the project requirements are broken down below based on which class file is involved.

3.1.1. InvulnTime.java

InvulnTime.java includes an object constructor with this.start(); so that the object will run on its own thread as soon as an object of that class is created. By using Timer and TimerTask I've created a simple task that will check if the player object is set to be vulnerable to damage, and if so, to set the player to not be vulnerable to damage, wait 1 second, and then toggle the player's state back to being vulnerable again. This creates a brief window of time for the player to get away from an enemy before taking damage again.

3.1.2. DamageFlicker.java

DamageFlicker.java also makes use of TimerTasks, but in this case, using a task that is scheduled to repeat periodically. The task toggles a flag stored in the player object repeatedly which the drawPlayer() method uses to distinguish which sprite to use. As an additional failsafe, after a brief delay once the window of time the player sprite should be changing is over the flag is set to false to ensure that the player sprite is not left stuck in the damaged state at the end of the loop.

3.1.3. TileGrid.java

TileGrid.java is the class where level data is interpreted into actual drawn tiles on the screen, in the previous version of this project it was not necessary to keep track of where the door tiles or blocks were located because there was only one level so hardcoded values were used. In this version, however, there are multiple levels, so it is required that the code be able to dynamically keep track of the door tiles, block tiles, and their current states across the different levels. To facilitate this, the refreshDoors() and refreshBlocks() methods were added to TileGrid.java.

3.1.4. Player.java

Working in conjunction with DamageFlicker.java, the Player.java's new checkSprite() method checks the current state of the player's "damaged" flag. Depending on if the player is "damaged" and if the player is supposed to be facing to the left or the right, this method will set the player's current sprite to be one of 4 possible spites. A non-damaged left-facing sprite, a non-damaged right-facing sprite, a damaged left-facing sprite, or a damaged right-facing sprite.

3.1.5. Enemy.java

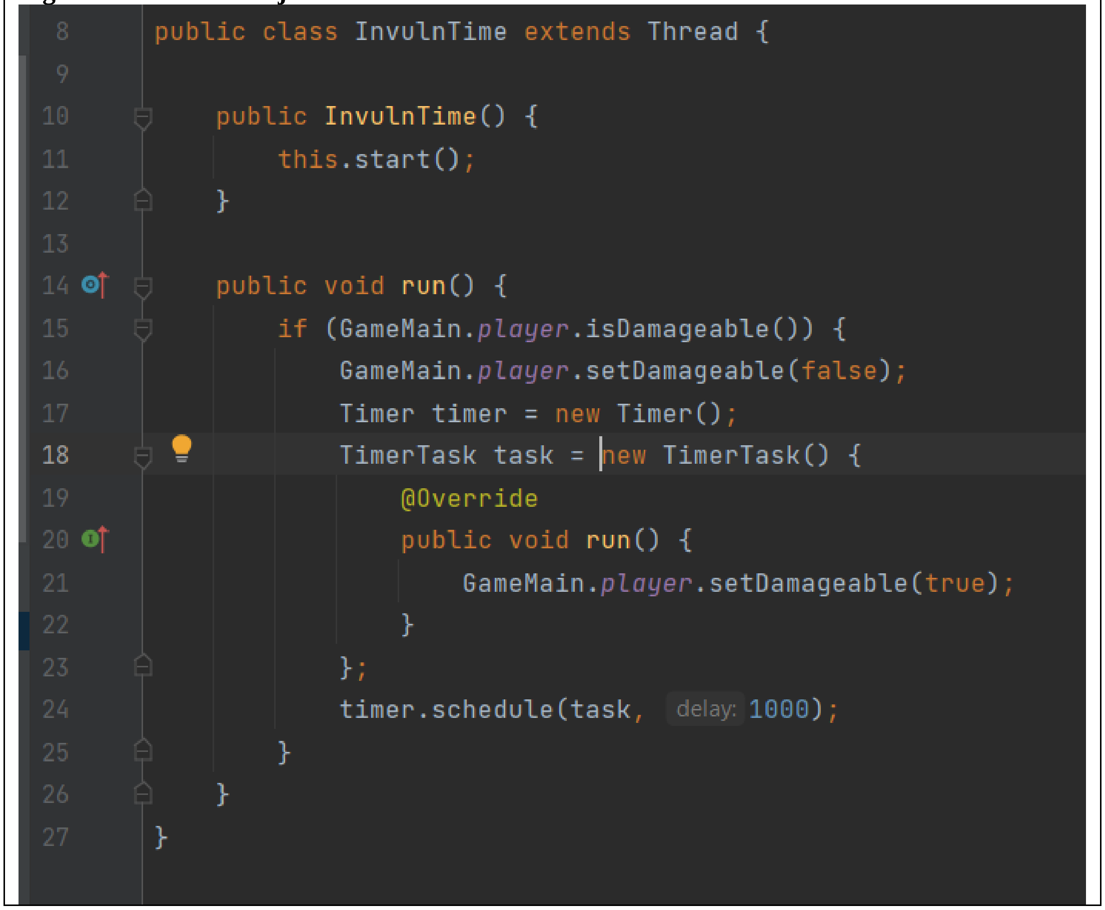
The changes to Enemy.java tie it all together for the player's health related requirements. The collision-detection for enemies has been modified to also check if the player is vulnerable to damage before decrementing their health. It also creates a new instance of both the InvulnTime and DamageFlicker objects so that those effects can run as soon as a valid damage collision between enemy and player is detected.

4. Implementation Paragraphs

Within the following paragraph sections I will show screenshots and brief descriptions of the code that fulfills the project requirements as well as the working levels and new player/bat sprite assets.

4.1. InvulnTime.java

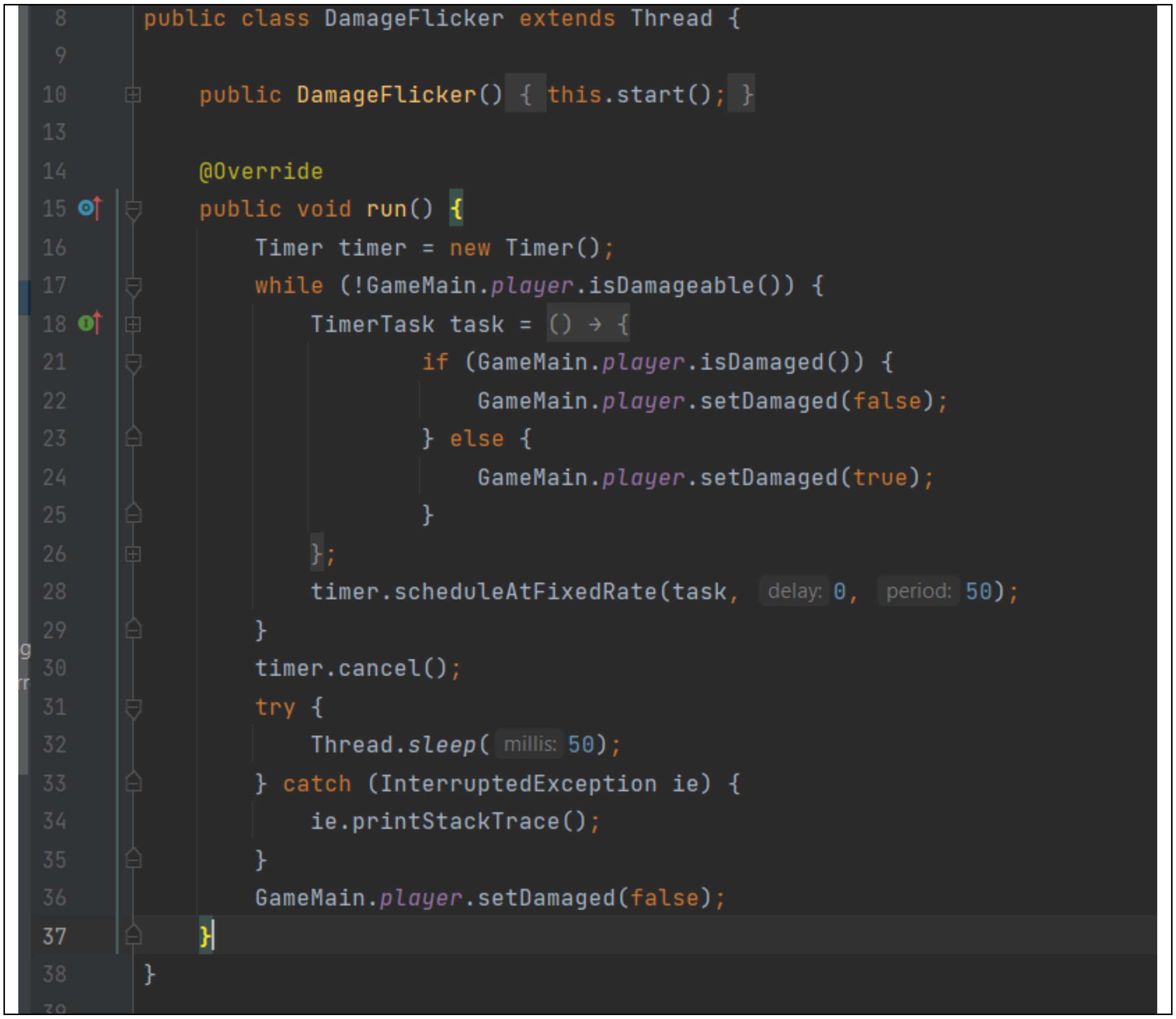
Figure 1 - InvulnTime.java



```java
public class InvulnTime extends Thread {

    public InvulnTime() {
        this.start();
    }

    public void run() {
        if (GameMain.player.isDamageable()) {
            GameMain.player.setDamageable(false);
            Timer timer = new Timer();
            TimerTask task = new TimerTask() {
                @Override
                public void run() {
                    GameMain.player.setDamageable(true);
                }
            };
            timer.schedule(task, delay: 1000);
        }
    }
}
```

The entire contents of the InvulnTime java class, toggles a player flag off, then back on again 1 second later.

4.2. DamageFlicker.java

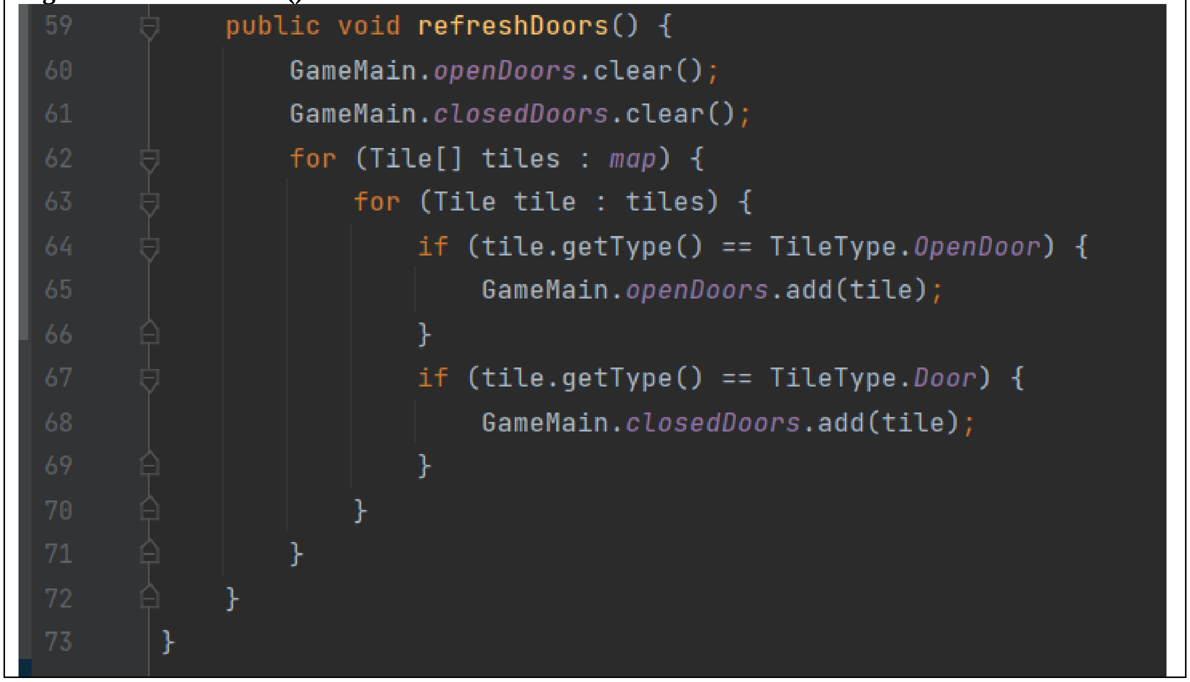


```
public class DamageFlicker extends Thread {

    public DamageFlicker() { this.start(); }

    @Override
    public void run() {
        Timer timer = new Timer();
        while (!GameMain.player.isDamageable()) {
            TimerTask task = () → {
                    if (GameMain.player.isDamaged()) {
                        GameMain.player.setDamaged(false);
                    } else {
                        GameMain.player.setDamaged(true);
                    }
            };
            timer.scheduleAtFixedRate(task, delay: 0, period: 50);
        }
        timer.cancel();
        try {
            Thread.sleep( millis: 50);
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }
        GameMain.player.setDamaged(false);
    }
}
```

The entire contents of the DamageFlicker java class, if player flag for isDamageable is false then an object of this class will toggle the player's isDamaged flag every 50 milliseconds until the player is damageable again, then sleep for an extra 50 milliseconds, then set the isDamaged flag to false as a failsafe to prevent the players getting stuck in the Damaged state.

4.3. TileGrid.java

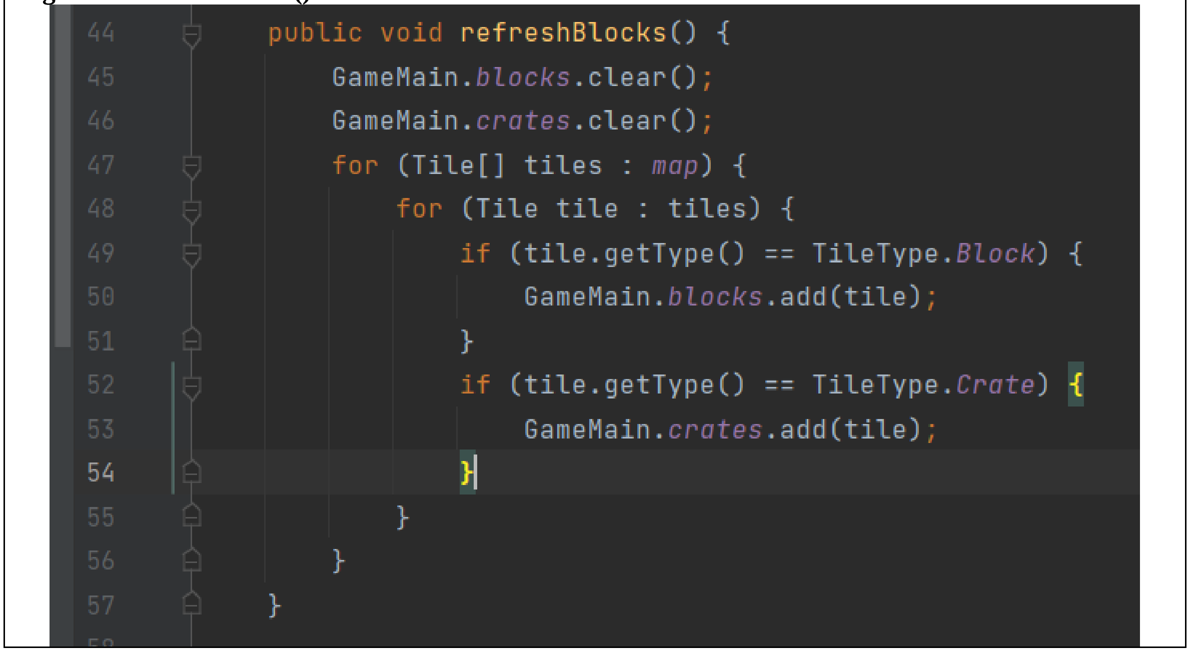
Figure 2 - refreshDoors()



```java
    public void refreshDoors() {
        GameMain.openDoors.clear();
        GameMain.closedDoors.clear();
        for (Tile[] tiles : map) {
            for (Tile tile : tiles) {
                if (tile.getType() == TileType.OpenDoor) {
                    GameMain.openDoors.add(tile);
                }
                if (tile.getType() == TileType.Door) {
                    GameMain.closedDoors.add(tile);
                }
            }
        }
    }
}
```

The refreshDoors() method clears arrayLists that store the Tiles containing doors or opened doors and then repopulates them by iterating through the current level map, and adding any Tiles of those types to the appropriate arrayList. This is so keys picked up in the level can dynamically open closed doors without having to hardcode in locations for each door.

Figure 3 - refreshBlocks()

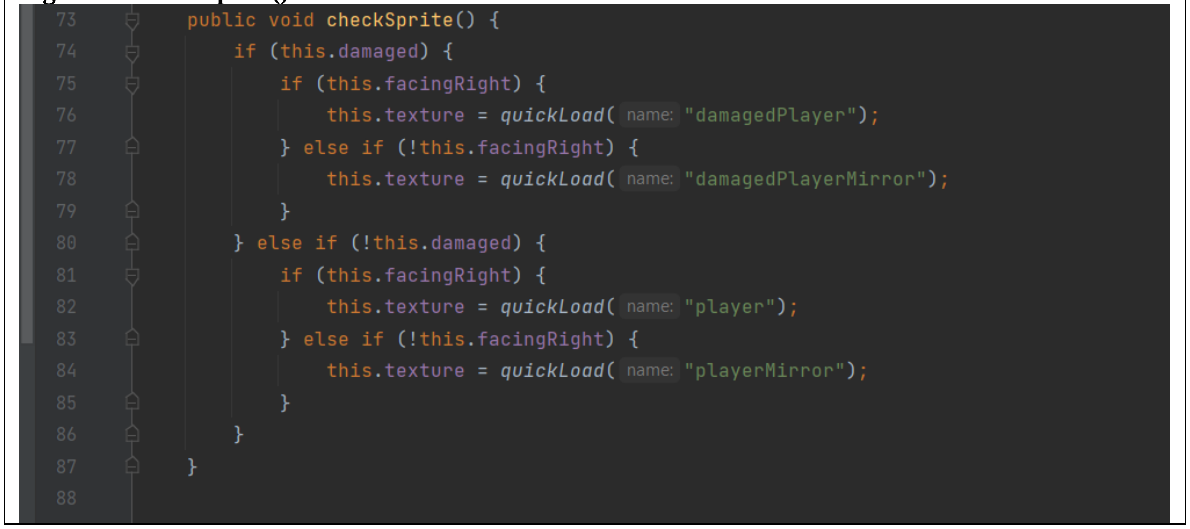


```java
    public void refreshBlocks() {
        GameMain.blocks.clear();
        GameMain.crates.clear();
        for (Tile[] tiles : map) {
            for (Tile tile : tiles) {
                if (tile.getType() == TileType.Block) {
                    GameMain.blocks.add(tile);
                }
                if (tile.getType() == TileType.Crate) {
                    GameMain.crates.add(tile);
                }
            }
        }
    }
```

The refreshBlocks() method works like the refreshDoors() one except it is clearing and repopulating arrayLists for Tiles containing Blocks (which cannot be passed through by the player) and Crates (which CAN be passed through by the player, but the player does not fall through them). Collision behavior is tied to these arrayLists so for this method is necessary to dynamically change level maps after clearing a level.

4.4. Player.java

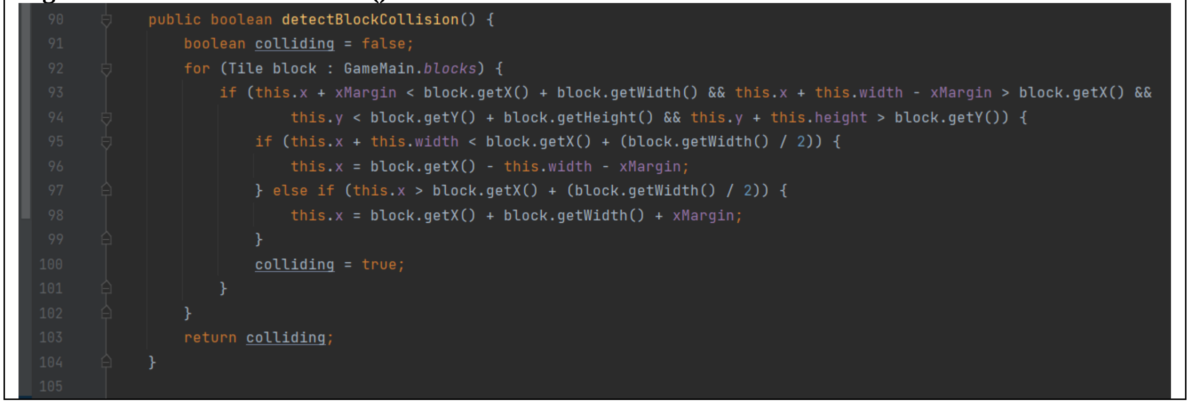
Figure 4 - checkSprite()



```
public void checkSprite() {
    if (this.damaged) {
        if (this.facingRight) {
            this.texture = quickLoad( name: "damagedPlayer");
        } else if (!this.facingRight) {
            this.texture = quickLoad( name: "damagedPlayerMirror");
        }
    } else if (!this.damaged) {
        if (this.facingRight) {
            this.texture = quickLoad( name: "player");
        } else if (!this.facingRight) {
            this.texture = quickLoad( name: "playerMirror");
        }
    }
}
```

This method allows the flag toggled by DamageFlicker.java to actually change the current sprite for the player. This method is required because attempting to change the sprite for the player outside of the Player class causes display context errors which is why boolean flags are toggled outside the class and then checked inside the class.

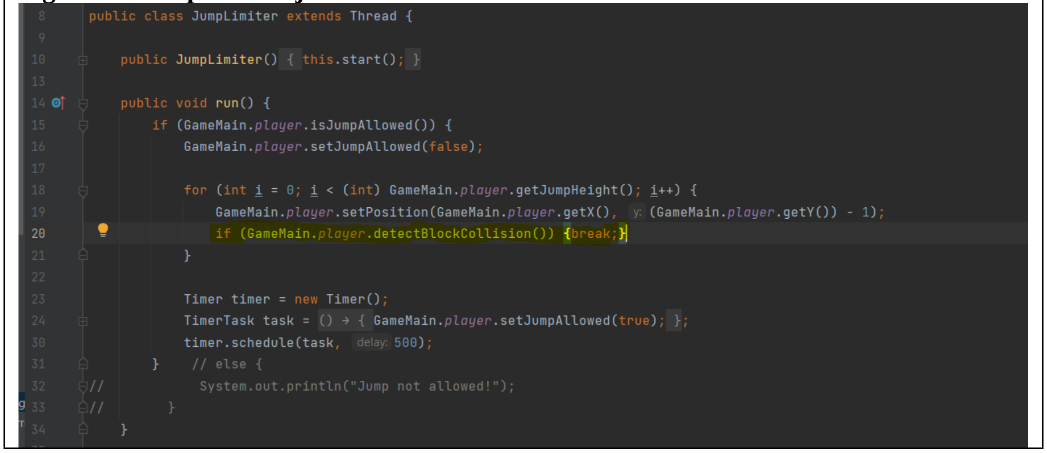
Figure 5 - detectBlockCollision()



```
public boolean detectBlockCollision() {
    boolean colliding = false;
    for (Tile block : GameMain.blocks) {
        if (this.x + xMargin < block.getX() + block.getWidth() && this.x + this.width - xMargin > block.getX() &&
                this.y < block.getY() + block.getHeight() && this.y + this.height > block.getY()) {
            if (this.x + this.width < block.getX() + (block.getWidth() / 2)) {
                this.x = block.getX() - this.width - xMargin;
            } else if (this.x > block.getX() + (block.getWidth() / 2)) {
                this.x = block.getX() + block.getWidth() + xMargin;
            }
            colliding = true;
        }
    }
    return colliding;
}
```

This method returns a boolean value based on if the player’s position ever overlaps with the position of one of the Block Tiles in the blocks arrayList. It also adjusts the players x-axis position if this condition is true depending on which side of the block the player overlaps with. This returned value is used in the JumpLimiter class to preemptively break out of the jump loop if it would cause the player to jump into a space occupied by a Block, as shown in the next figure.

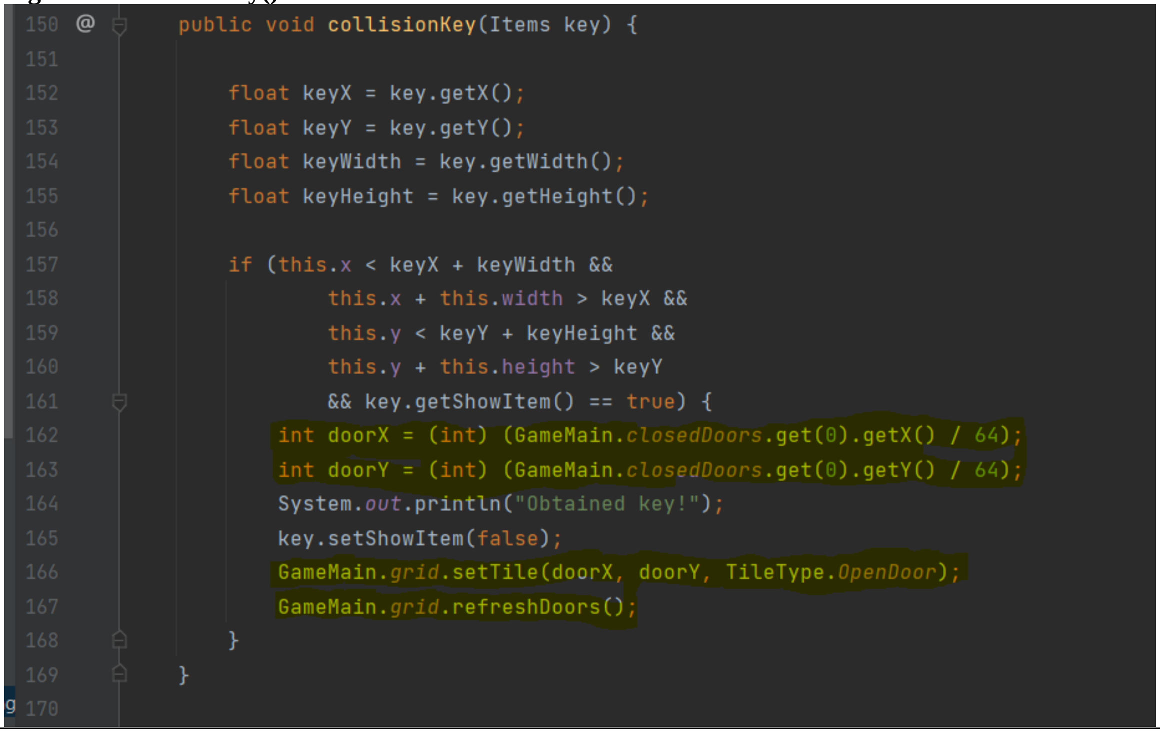
Figure 6 - JumpLimiter.java



```
public class JumpLimiter extends Thread {

    public JumpLimiter() { this.start(); }

    public void run() {
        if (GameMain.player.isJumpAllowed()) {
            GameMain.player.setJumpAllowed(false);

            for (int i = 0; i < (int) GameMain.player.getJumpHeight(); i++) {
                GameMain.player.setPosition(GameMain.player.getX(),  y: (GameMain.player.getY()) - 1);
                if (GameMain.player.detectBlockCollision()) {break;}
            }

            Timer timer = new Timer();
            TimerTask task = () → { GameMain.player.setJumpAllowed(true); };
            timer.schedule(task,  delay: 500);
        }    // else {
//            System.out.println("Jump not allowed!");
//        }
    }
```

On the highlighted line here, you can see that if the returned boolean from detectBlockCollision is true, then we break out of this for loop early. Effectively preventing the player from entering a Block tile from underneath by jumping.

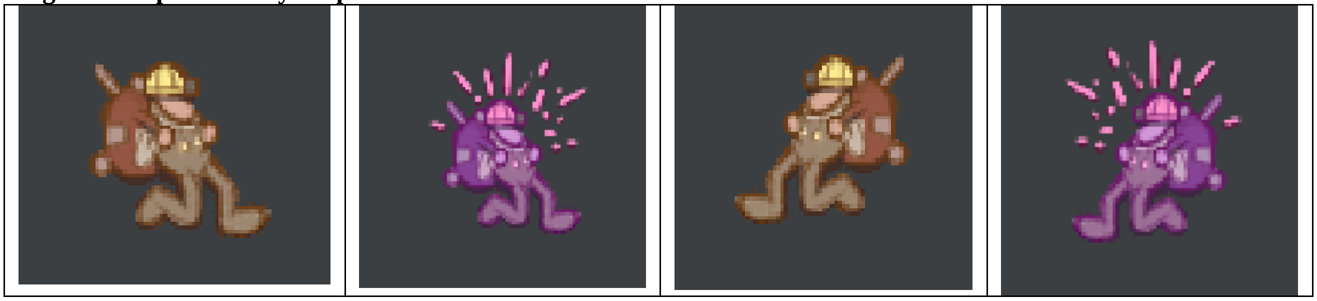
Figure 7 - collisionKey()



```
    public void collisionKey(Items key) {

        float keyX = key.getX();
        float keyY = key.getY();
        float keyWidth = key.getWidth();
        float keyHeight = key.getHeight();

        if (this.x < keyX + keyWidth &&
                this.x + this.width > keyX &&
                this.y < keyY + keyHeight &&
                this.y + this.height > keyY
                && key.getShowItem() == true) {
            int doorX = (int) (GameMain.closedDoors.get(0).getX() / 64);
            int doorY = (int) (GameMain.closedDoors.get(0).getY() / 64);
            System.out.println("Obtained key!");
            key.setShowItem(false);
            GameMain.grid.setTile(doorX, doorY, TileType.OpenDoor);
            GameMain.grid.refreshDoors();
        }
    }
```

The highlighted portions of the collisionKey() method indicate where it was modified. In the previous version since there was only one level, this method used hardcoded coordinates for changing the closed door Tile to an opened door Tile. Now it grabs the X-axis and Y-axis coordinates from whatever tile is stored in the closedDoors arrayList and uses that to find and update the closed door to an opened door. In this way the same method can be used to handle each level's door dynamically instead of needing redundant code with different hardcoded values inside.

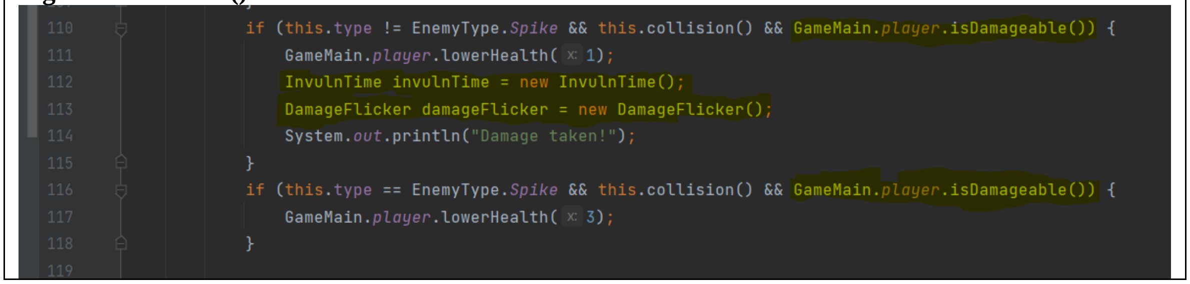
Figure 8 - Updated Player Sprites



Right-facing Non-damaged player | Right-facing damaged player | Left-facing non-damaged player | Left-facing damaged player

4.5. Enemy.java

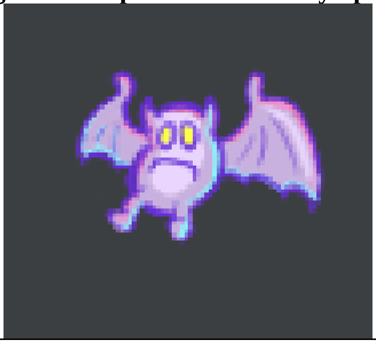
Figure 9 - collision()



```java
if (this.type != EnemyType.Spike && this.collision() && GameMain.player.isDamageable()) {
    GameMain.player.lowerHealth( x: 1);
    InvulnTime invulnTime = new InvulnTime();
    DamageFlicker damageFlicker = new DamageFlicker();
    System.out.println("Damage taken!");
}
if (this.type == EnemyType.Spike && this.collision() && GameMain.player.isDamageable()) {
    GameMain.player.lowerHealth( x: 3);
}
```

The Enemy class’s collision() method has been modified to incorporate the new InvulnTime and DamageFlicker objects as well as the Player’s new isDamageable flag. It now checks if a player is vulnerable to damage before attempting damage collision and if a valid collision is detected it instantiates a copy of the InvulnTime and DamageFlicker objects to handle the invulnerability period after taking damage as well as adjusting the player’s sprite during that invulnerability period.

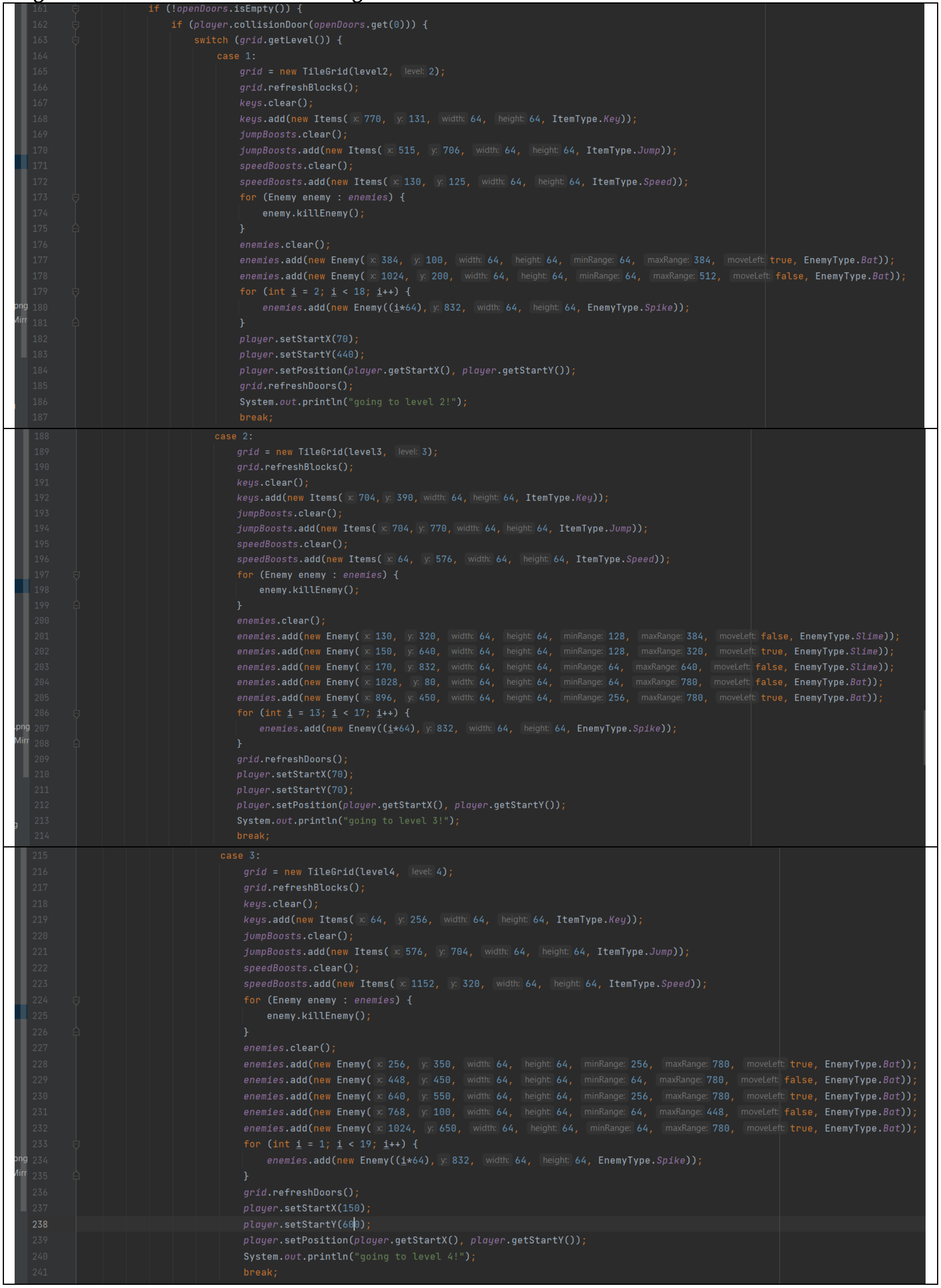
Figure 10 - Updated Bat Enemy Sprite



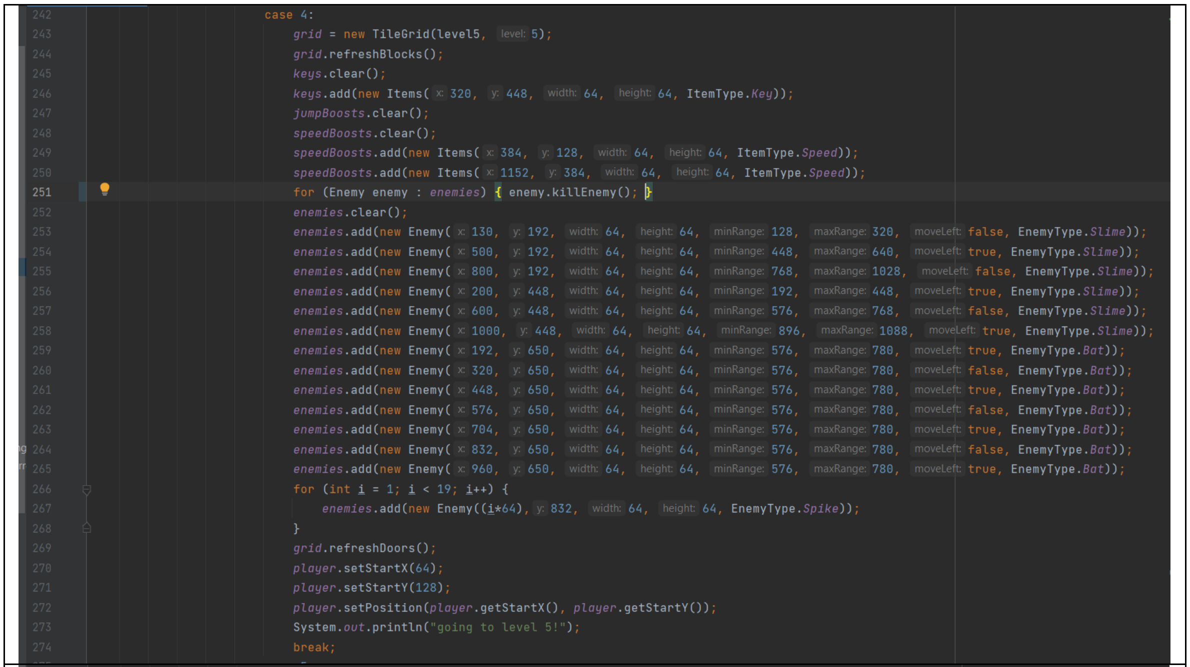
The bat enemy sprite was updated to use a custom asset.

4.6. GameMain.java

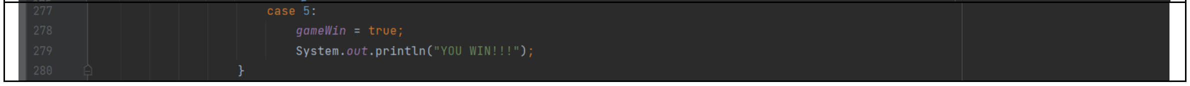
Figure 11 - Switch Case for Advancing Levels



```java
if (!openDoors.isEmpty()) {
    if (player.collisionDoor(openDoors.get(0))) {
        switch (grid.getLevel()) {
            case 1:
                grid = new TileGrid(level2,  level: 2);
                grid.refreshBlocks();
                keys.clear();
                keys.add(new Items( x: 770,  y: 131,  width: 64,  height: 64, ItemType.Key));
                jumpBoosts.clear();
                jumpBoosts.add(new Items( x: 515,  y: 706,  width: 64,  height: 64, ItemType.Jump));
                speedBoosts.clear();
                speedBoosts.add(new Items( x: 130,  y: 125,  width: 64,  height: 64, ItemType.Speed));
                for (Enemy enemy : enemies) {
                    enemy.killEnemy();
                }
                enemies.clear();
                enemies.add(new Enemy( x: 384,  y: 100,  width: 64,  height: 64,  minRange: 64,  maxRange: 384,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 1024,  y: 200,  width: 64,  height: 64,  minRange: 64,  maxRange: 512,  moveLeft: false, EnemyType.Bat));
                for (int i = 2; i < 18; i++) {
                    enemies.add(new Enemy((i*64),  y: 832,  width: 64,  height: 64, EnemyType.Spike));
                }
                player.setStartX(70);
                player.setStartY(440);
                player.setPosition(player.getStartX(), player.getStartY());
                grid.refreshDoors();
                System.out.println("going to level 2!");
                break;
            case 2:
                grid = new TileGrid(level3,  level: 3);
                grid.refreshBlocks();
                keys.clear();
                keys.add(new Items( x: 704, y: 390, width: 64, height: 64, ItemType.Key));
                jumpBoosts.clear();
                jumpBoosts.add(new Items( x: 704, y: 770, width: 64, height: 64, ItemType.Jump));
                speedBoosts.clear();
                speedBoosts.add(new Items( x: 64,  y: 576,  width: 64,  height: 64, ItemType.Speed));
                for (Enemy enemy : enemies) {
                    enemy.killEnemy();
                }
                enemies.clear();
                enemies.add(new Enemy( x: 130,  y: 320,  width: 64,  height: 64,  minRange: 128,  maxRange: 384,  moveLeft: false, EnemyType.Slime));
                enemies.add(new Enemy( x: 150,  y: 640,  width: 64,  height: 64,  minRange: 128,  maxRange: 320,  moveLeft: true, EnemyType.Slime));
                enemies.add(new Enemy( x: 170,  y: 832,  width: 64,  height: 64,  minRange: 64,  maxRange: 640,  moveLeft: false, EnemyType.Slime));
                enemies.add(new Enemy( x: 1028,  y: 80,  width: 64,  height: 64,  minRange: 64,  maxRange: 780,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 896,  y: 450,  width: 64,  height: 64,  minRange: 256,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                for (int i = 13; i < 17; i++) {
                    enemies.add(new Enemy((i*64),  y: 832,  width: 64,  height: 64, EnemyType.Spike));
                }
                grid.refreshDoors();
                player.setStartX(70);
                player.setStartY(70);
                player.setPosition(player.getStartX(), player.getStartY());
                System.out.println("going to level 3!");
                break;
            case 3:
                grid = new TileGrid(level4,  level: 4);
                grid.refreshBlocks();
                keys.clear();
                keys.add(new Items( x: 64,  y: 256,  width: 64,  height: 64, ItemType.Key));
                jumpBoosts.clear();
                jumpBoosts.add(new Items( x: 576,  y: 704,  width: 64,  height: 64, ItemType.Jump));
                speedBoosts.clear();
                speedBoosts.add(new Items( x: 1152,  y: 320,  width: 64,  height: 64, ItemType.Speed));
                for (Enemy enemy : enemies) {
                    enemy.killEnemy();
                }
                enemies.clear();
                enemies.add(new Enemy( x: 256,  y: 350,  width: 64,  height: 64,  minRange: 256,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 448,  y: 450,  width: 64,  height: 64,  minRange: 64,  maxRange: 780,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 640,  y: 550,  width: 64,  height: 64,  minRange: 256,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 768,  y: 100,  width: 64,  height: 64,  minRange: 64,  maxRange: 448,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 1024,  y: 650,  width: 64,  height: 64,  minRange: 64,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                for (int i = 1; i < 19; i++) {
                    enemies.add(new Enemy((i*64),  y: 832,  width: 64,  height: 64, EnemyType.Spike));
                }
                grid.refreshDoors();
                player.setStartX(150);
                player.setStartY(600);
                player.setPosition(player.getStartX(), player.getStartY());
                System.out.println("going to level 4!");
                break;
```



```java
            case 4:
                grid = new TileGrid(level5,  level: 5);
                grid.refreshBlocks();
                keys.clear();
                keys.add(new Items( x: 320,  y: 448,  width: 64,  height: 64, ItemType.Key));
                jumpBoosts.clear();
                speedBoosts.clear();
                speedBoosts.add(new Items( x: 384,  y: 128,  width: 64,  height: 64, ItemType.Speed));
                speedBoosts.add(new Items( x: 1152,  y: 384,  width: 64,  height: 64, ItemType.Speed));
                for (Enemy enemy : enemies) { enemy.killEnemy(); }
                enemies.clear();
                enemies.add(new Enemy( x: 130,  y: 192,  width: 64,  height: 64,  minRange: 128,  maxRange: 320,  moveLeft: false, EnemyType.Slime));
                enemies.add(new Enemy( x: 500,  y: 192,  width: 64,  height: 64,  minRange: 448,  maxRange: 640,  moveLeft: true, EnemyType.Slime));
                enemies.add(new Enemy( x: 800,  y: 192,  width: 64,  height: 64,  minRange: 768,  maxRange: 1028,  moveLeft: false, EnemyType.Slime));
                enemies.add(new Enemy( x: 200,  y: 448,  width: 64,  height: 64,  minRange: 192,  maxRange: 448,  moveLeft: true, EnemyType.Slime));
                enemies.add(new Enemy( x: 600,  y: 448,  width: 64,  height: 64,  minRange: 576,  maxRange: 768,  moveLeft: false, EnemyType.Slime));
                enemies.add(new Enemy( x: 1000,  y: 448,  width: 64,  height: 64,  minRange: 896,  maxRange: 1088,  moveLeft: true, EnemyType.Slime));
                enemies.add(new Enemy( x: 192,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 320,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 448,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 576,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 704,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                enemies.add(new Enemy( x: 832,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: false, EnemyType.Bat));
                enemies.add(new Enemy( x: 960,  y: 650,  width: 64,  height: 64,  minRange: 576,  maxRange: 780,  moveLeft: true, EnemyType.Bat));
                for (int i = 1; i < 19; i++) {
                    enemies.add(new Enemy((i*64),  y: 832,  width: 64,  height: 64, EnemyType.Spike));
                }
                grid.refreshDoors();
                player.setStartX(64);
                player.setStartY(128);
                player.setPosition(player.getStartX(), player.getStartY());
                System.out.println("going to level 5!");
                break;
```



```java
            case 5:
                gameWin = true;
                System.out.println("YOU WIN!!!");
        }
```

Within the main game loop, this switch case was added to detect when the player collides with an open door and based on the associated integer value of the current level it will clear all objects and repopulate the screen with the objects for the next level, up until the level count is 5 at which point the gameWin boolean flag will be toggled instead which ends the game.

4.7. Additional Game Levels

Figure 12 - Level 2

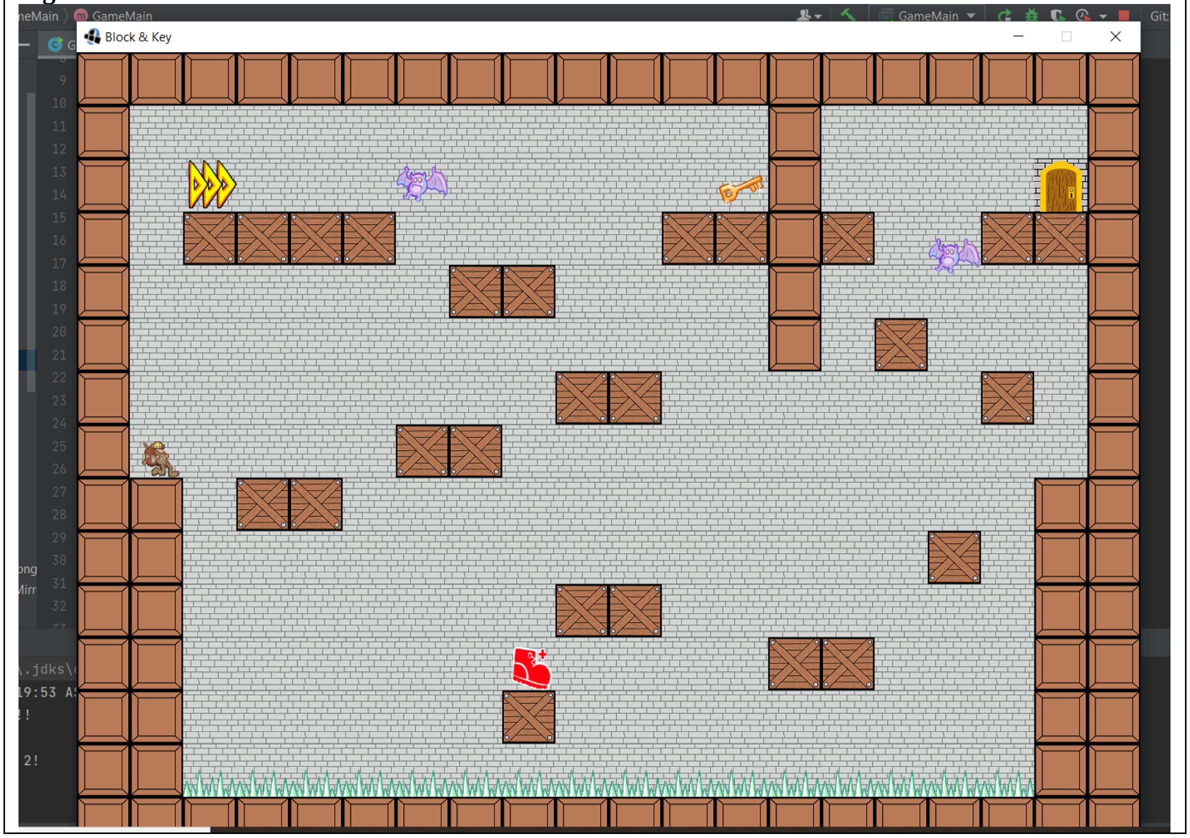


Figure 13 - Level 3

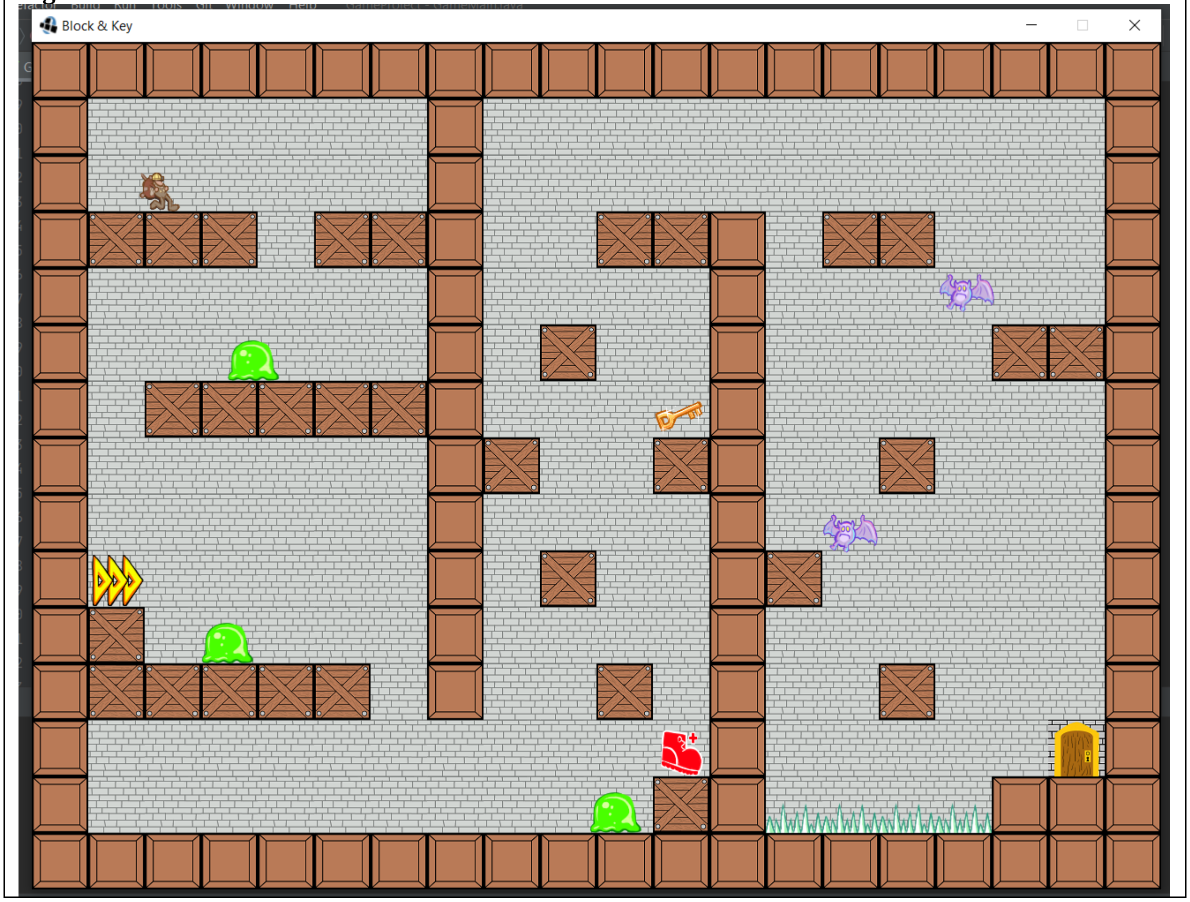


Figure 14 - Level 4

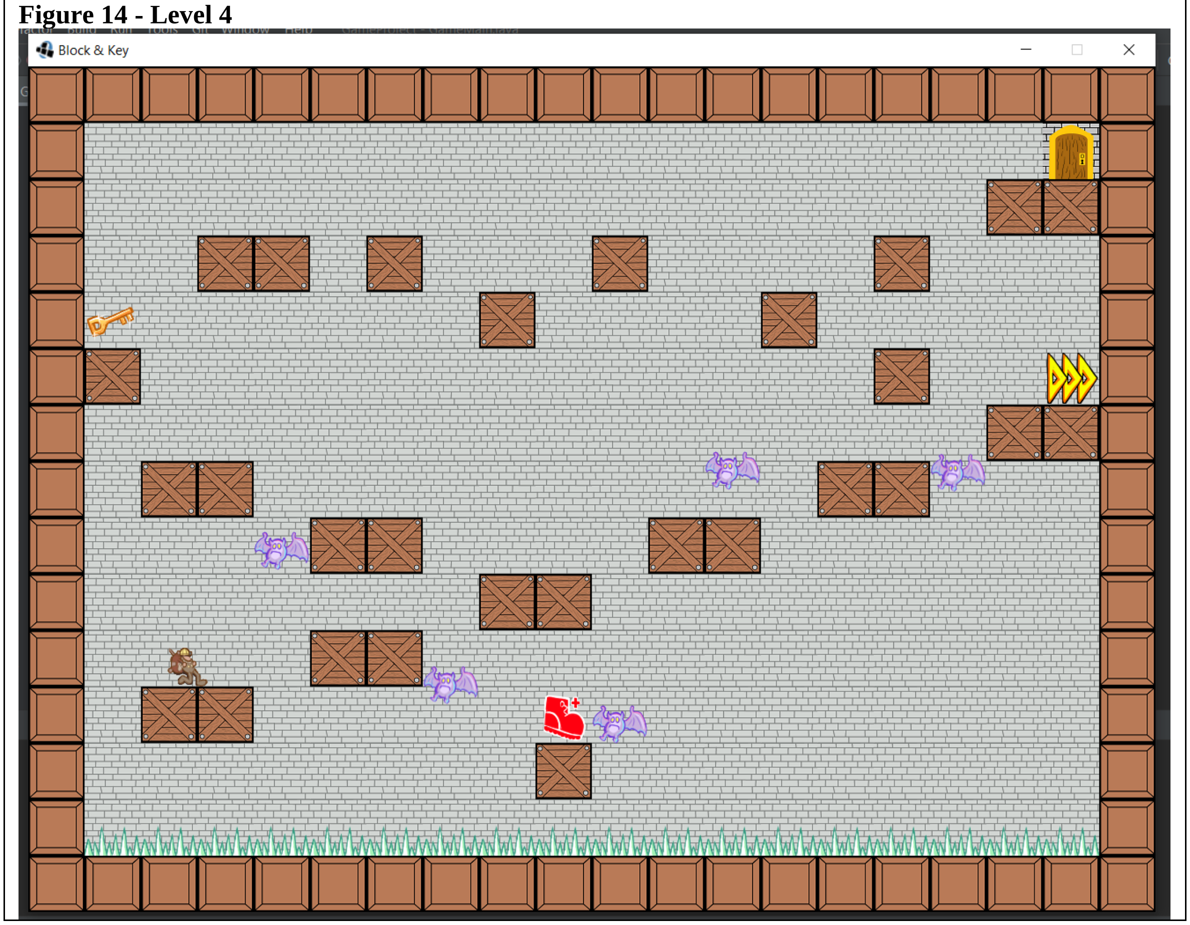
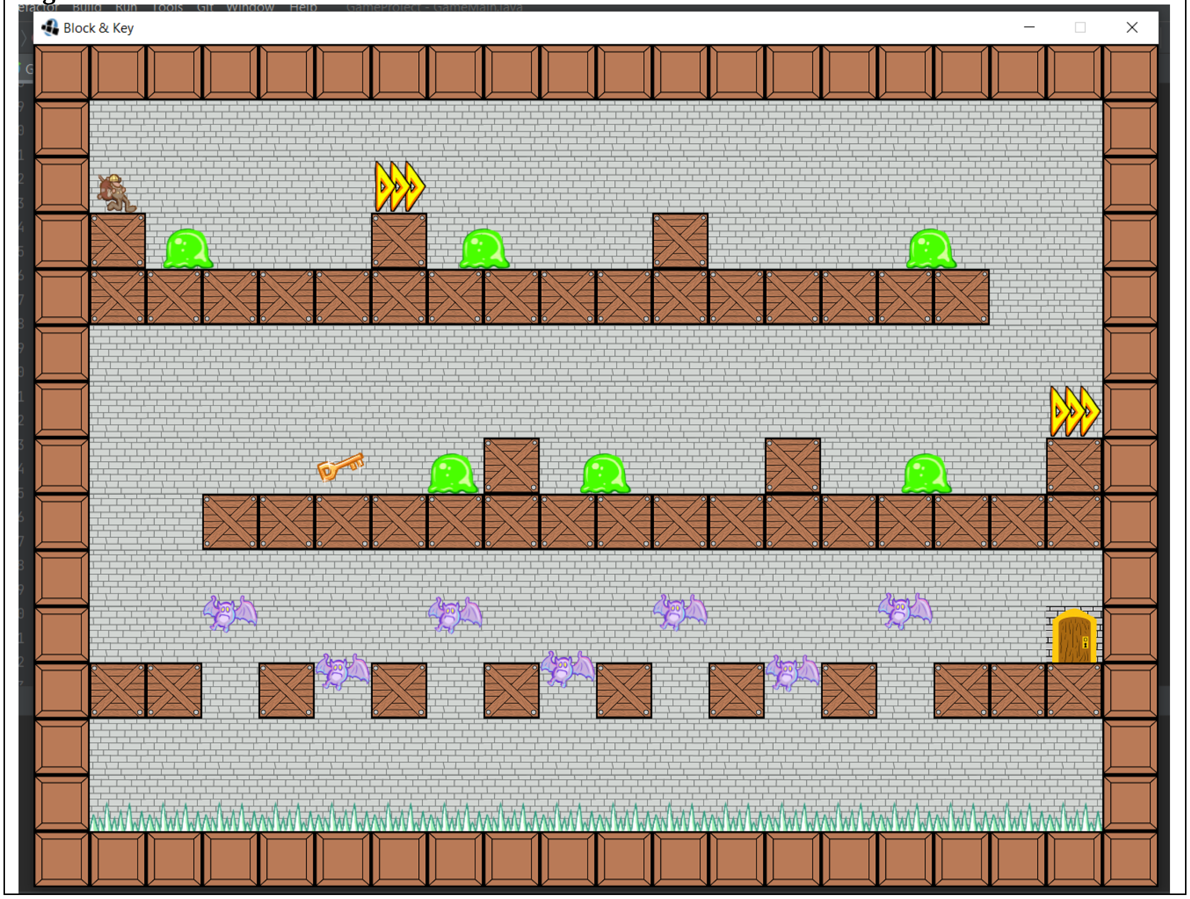


Figure 15 - Level 5



Appendix A. Requirements Traceability

This appendix shows all the requirements for this project, where in this document they are defined, and where in this document the game/code screenshots that fulfills those requirements are.

Paragraph Defined	Requirements Text	Paragraphs Implemented
2	Adding 4 additional levels, for a total of 5	4.7
2	Improve collision handling such that the player cannot easily escape the bounds of the level.	3.1.3,3.1.4,4.3,4.4
2	Improve the player health system to allow a delay between taking damage and being able to take damage again.	3.1.1,3.1.2,3.1.4,3.1.5, 4.1,4.2,4.4,4.5
2	Update player and bat enemy sprite assets.	4.4, 4.5
2.1	Add visual indicator to show how long the player is invulnerable after taking damage.	3.1.2, 4.2
2.1	The main loop must be able to remove assets from previous levels and load assets for the next level upon clearing a level.	4.6