Fatima Jinnah Women University

Opening Portals of Excellence Through Higher Education

SOFTWARE REQUIREMENT SPECIFICATION

"BOMBERMAN GAME"

VERSION 1.0

TABLE OF CONTENT

1. HISTORY

Bombarman, commonly known as Dyna Blaster in Europe, is an action video game franchise created by Hudson Soft and currently owned by Konami. Hudson Soft, a defunct Japanese software company and publisher, produced the long-running franchise Bomber Man.

The first game, also known as Bakudan Otoko, was launched in Japan in July 1983. Since then, it has inspired a number of copies and spin-offs that have been made available on a variety of platforms, as well as a number of anime and manga adaptations. More than 10 million copies of the series have been sold as of 2002.

2. INTRODUCTION

1.1 : OVERVIEW

Gaming is the Peak of Creativity, It’s not easy to become a Good Player but a Good Developer. In This Document all the system features, Interface, description, implementation and testing of bomber man game is listed. Here we are creating an interactive game based on the classic bomber man game.

1.2 : PURPOSE

The purpose of the game is just provide fun and increase intelligence of kids, youngsters by this game, The aim of the game is to defeat enemies with a wise manner an reach an exit point to progress through levels.

1.3 : PRODUCT SCOPE

Bomberman is a strategic, maze-based computer video Game with an action genre game and a Player. The objective of the game is to eliminate all of the “Crates” that are distributed around the block of the game screen.

3. REQUIREMENTS

3.1 : ELICITATION

In this step we will have a discussion with our team or the clients which want this game, the discussion include some main points which are based on:

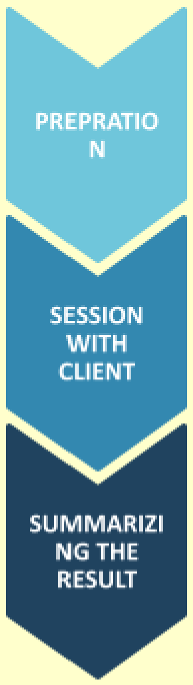
- Communication Skills.
- Brainstorming.

3.1.1 COMMUNICATION SKILLS

We will involve a rich communication activity that entails our customer or team to interact and communicate their needs with us. Our analysts who have good communication skills on the other hand should have the capacity to grasp our game domain knowledge, Because Misunderstandings and communication gaps between two groups can cause negative consequences to software project.

3.1.2 BRAINSTORMING

We will now have a table discussion in which we generate new ideas and will also find the solution of a specific issue for our client. Our team will do a brief group discussion and can share their ideas with our domain experts, Software experts as well, because multiple ideas, collect requirements and information can give us more knowledge with tremendous techniques. The main points will include:



- Agenda
- Time Limit
- Identify the Participants

- Share and Record Ideas
- Get as many ideas as possible
- Give chance to speak to all the members

- Discuss ideas & remove duplicate ideas
- Distribute final list

3.2 : EXTERNAL INTERFACE REQUIRMENTS

3.2.1: USER INTERFACE

The user interface in the Bomberman game video game is to carry out the task with in the game. All the game rules will be followed. In this a user have to destroy all crates through the red colored big ball, the bomb and we just place the bomb with the crate which we have to break to complete level. All the guidance of Moving a player is mentioned which is that we have to use arrow keys to move Player (Bomberman) and to drop the bomb with space key.

3.2.2: HARDWARE INTERFACE

The hardware modules in our project Bomberman game are not very large in number because we implemented our project by coding in java. The hardware interface are like the controller interfaces. Only Desktop is used to play his Game. We just use computer screen and mouse. Using keyboard we can play this game. Additionally, we have to use key board arrow keys and space key.

3.2.3: SOFTWARE INTERFACE

Software interfaces are just used to support graphics for our game code because software is the main driver for our game code. The software that we used for coding our Bomberman game is Eclipse with the JDK version 3.0.2. There was a proper class of the bomb and also defined its size in which it is clearly mentioned when to start up the bomb, when to explode it, radius and no of players can play. We used other classes like power up, enemy, player, floor, etc. we imported libraries of javax.swing.*, java.awt.* to make the frame or the interface of our game.

3.2.4: COMMUNICATION INTERFACE

As our Bomberman game is only single player game at a time only one player can play it. We just developed it using java coding so there is no such communicating interface neither with the players nor with the enemies there. This game does not require internet connectivity. This is the kind of online game can be downloaded.

3.3 : SYSTEM FEATURE

3.3.1: FUNCTIONAL REQUIREMENTS

As all the requirements in our game are necessary which includes the strategy of a game, as our game is a battle royal so destroy as many crates as user have to collect powerups and become stronger & hopefully you get lucky and get a lot of powerups.

Invalid Action can cause a loss in game, so rules are necessary in every game, some rules which should be followed in bomber man game are:

- Bombs explodes in 4 directions and the length of the explosion depends on the firepower of the original player who places it.
- Colliding with an explosion will kill the player.
- Bombs are solid when players collide with it from outside.
- Soft walls have a 50% chance to drop a random power up upon getting destroyed.

3.3.2: DESCRIPTION
The high priority in our game is has assigned to crates and a player so he can destroy the crates as proceed to next level.

Then the medium priority in our game is assigned to a enemies and the bomb in our given block.

The lowest priority in our game is assigned to the power that a player have to eat after he destroy a crate.

3.3.3: RESPONSE SEQUENCE
First the user will open the game and then the response he will get is that the game has been opened and started. Now the player will destroy all the crates and enemies to protect himself, step by step.

3.4 : NON-FUNCTIONAL REQUIREMENTS

3.4.1: PERFROMANCE
The performance of the game is not very fast as compare to rest of the videos game as our player take almost a second to respond and move from one place to another.

3.4.1.1 : PLAYER
The Player performance is slightly low as when he drops the bomb it takes some second to move from one place to another.

3.4.1.2 : ENEMY
Enemies on our game plays a vital role to chase enemies and to increase their points by destroying them with a bomb. Enemies will only move in a specific direction weather vertical or horizontal depend upon the area they have allotted and they have a good performance.

3.4.1.2 : BUMB
The bomb will explode in 5 sec as we have set in our back end so the explosion of bomb also have a good performance.

3.4.2: SAFETY REQUIREMENT

The only safety requirement we have to remember is just not to place bomb in wrong direction which results in loss of player.

3.4.3: SOFTWARE ATTRIBUTES

3.4.3.1 : FLEXIBILITY

This game provide flexibility to a user in case he has no internet connection, without internet connection player can play this game with a good speed.

3.4.3.2 : CORRECTNESS

The place where the user will try to move its player or where it explode the bomb, it will happen exactly the same.

3.4.3.3 : REALIBILITY

This game is secure and reliable, it will calculate points with reliability and never give wrong points nor restart level without leaving the game or destroy with bomb or enemies.

3.4.4: EASE TO USE

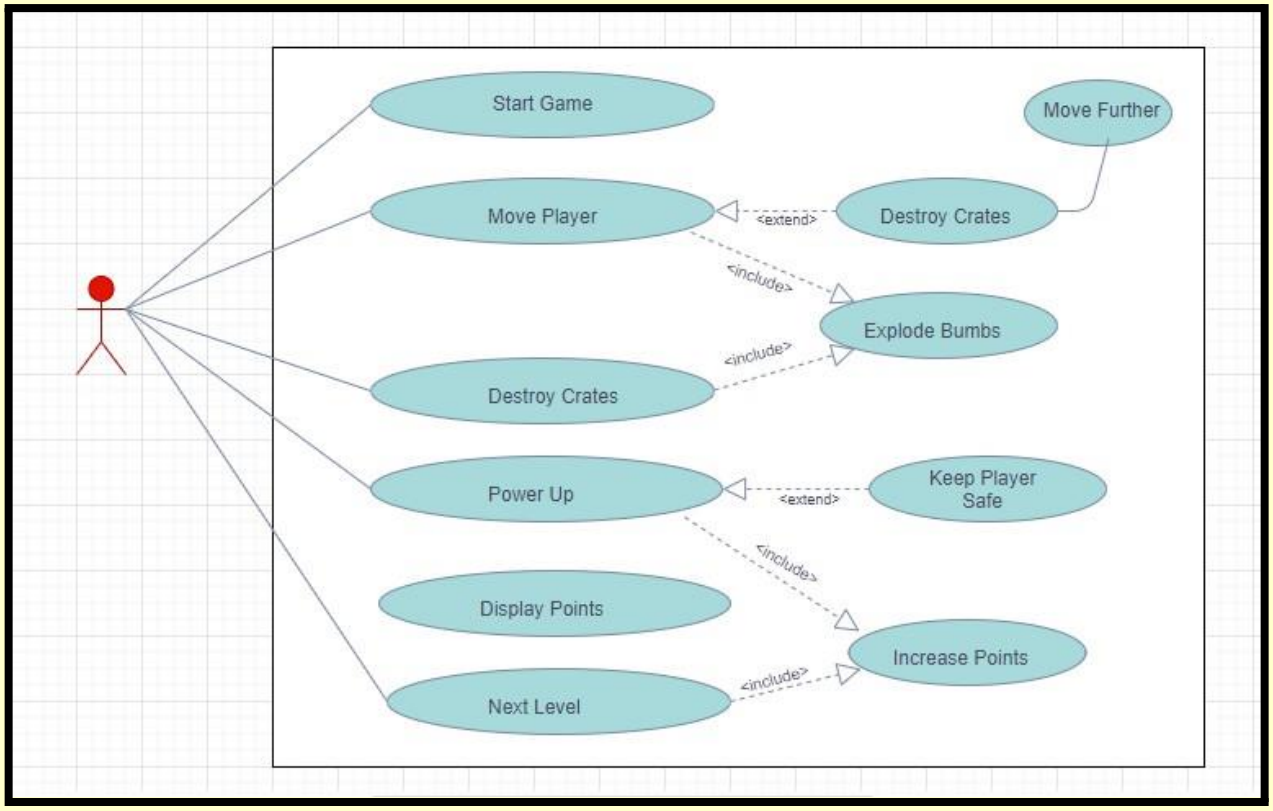
As the bomber man game has very specific characters and a simple way to play so its provide user an ease to learn so that a normal person having not much know how about the game can also play it.

3.4.5: REUSE OBJECTIVE

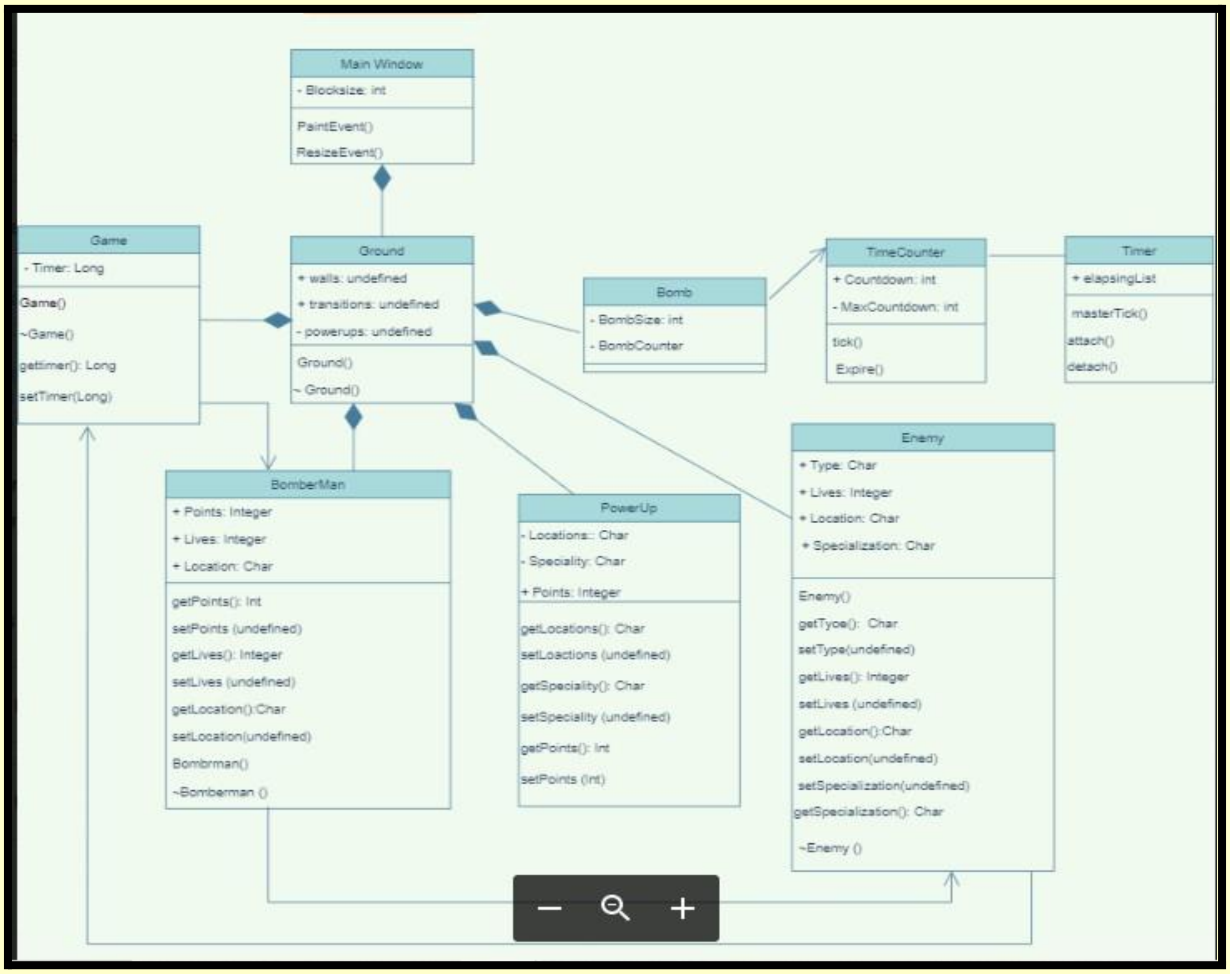
We can reuse the same bomb enemies and player to every single player bomberman game.

4. DESIGN MODEL

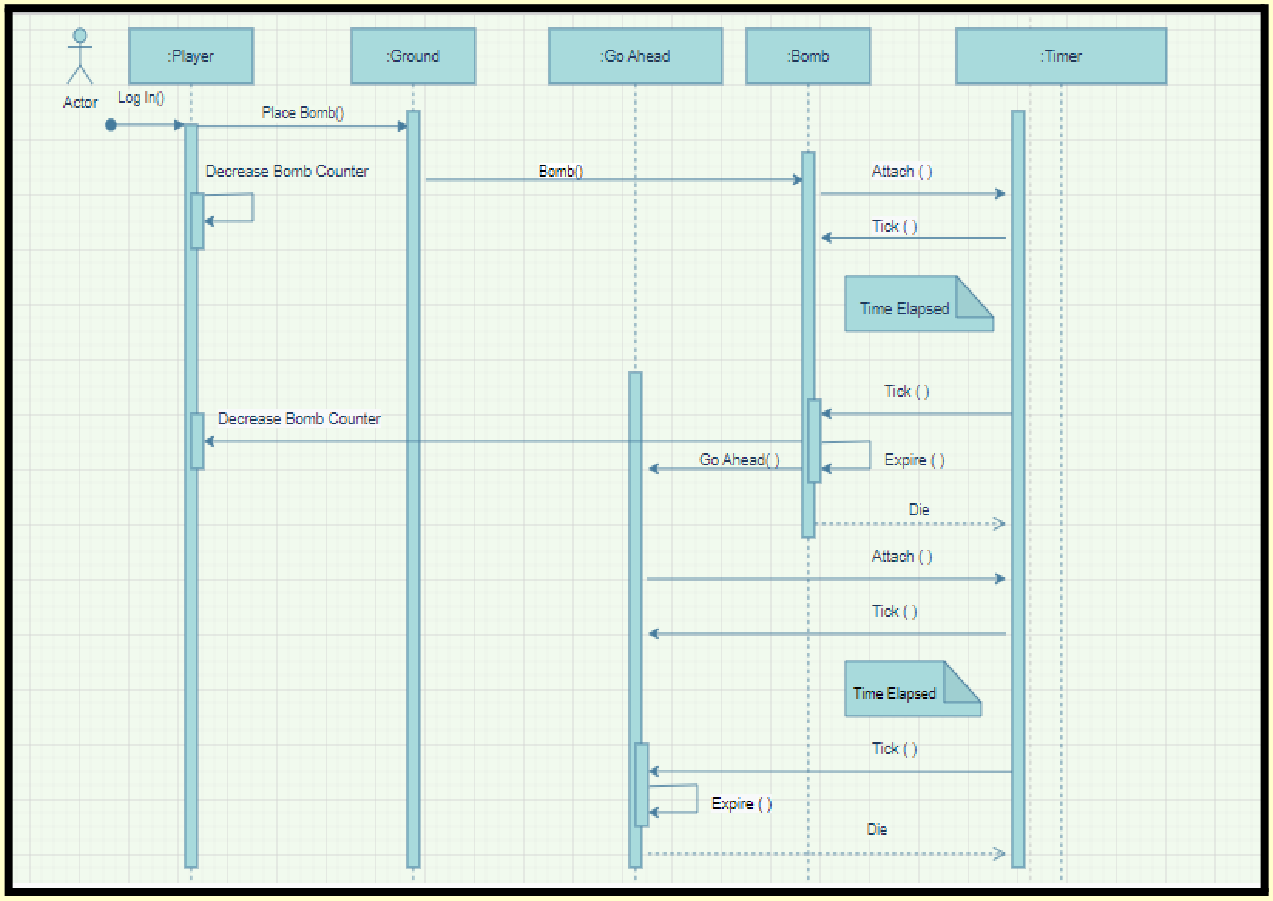
4.1 : USECASE DIAGRAM



4.2 : CLASS DIAGRAM

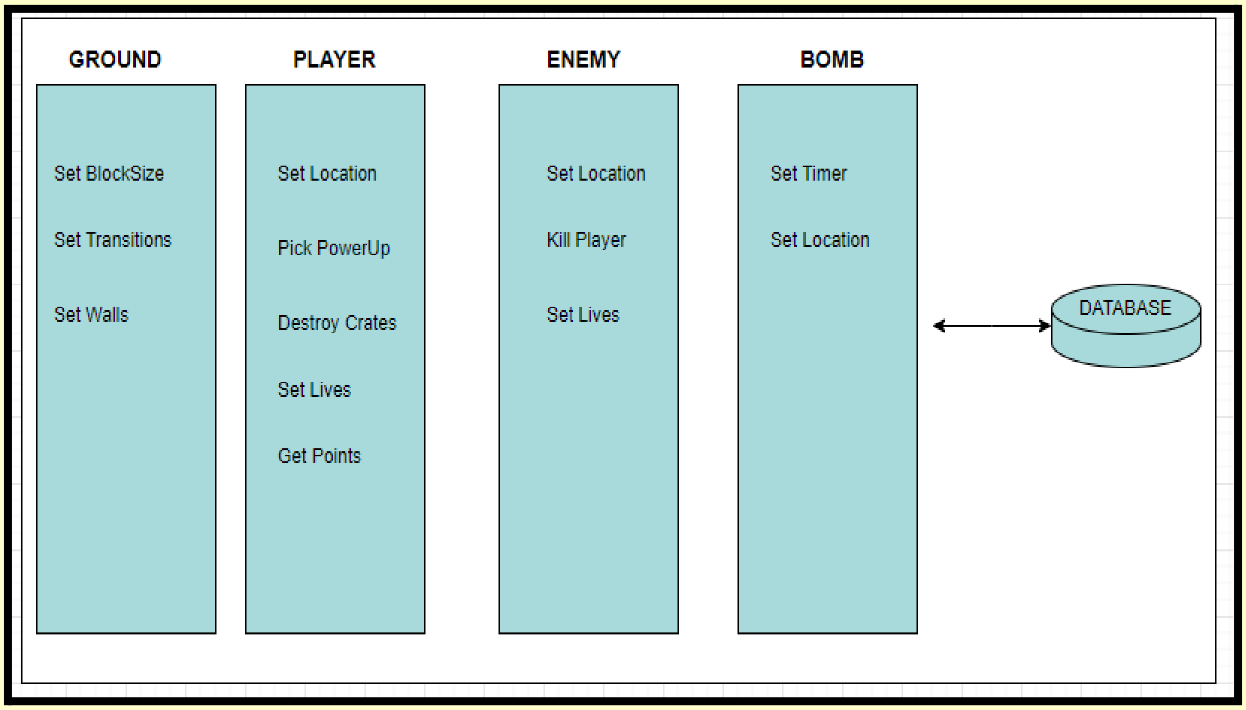


4.3 : SEQUENCE DIAGRAM



5. ARCHITECTURE DIAGRAM

In this model we make an architecture diagram of BomberMan game which includes Ground, Player, Enemy and Bomb of what are their tasks and what they have to do. All the data will then include in data base.



6. IMPLEMENTATION

6.1 : BOMBERMAN GAME

6.1.1 : PACKAGE

```
package game;
```

6.1.2 : CLASSES

6.1.2.1 : ABSTRACT CHARACTER

```
package game;

publicclass AbstractCharacter
{
privatefinalstaticintSIZE = 30;
privateintx;
privateinty;
privateintpixelsPerStep;

protected AbstractCharacter(intx, inty, intpixelsPerStep) {
        this.x = x;
        this.y = y;
        this.pixelsPerStep = pixelsPerStep;
    }
publicenum Move
     {
        DOWN(0, 1),
        UP(0, -1),
        RIGHT(1, 0),
        LEFT(-1, 0);

        privatefinalintdeltaX;
        privatefinalintdeltaY;
        Move(finalintdeltaX, finalintdeltaY) {
        this.deltaX = deltaX;
        this.deltaY = deltaY;
        }
     }

publicvoid move(Move move) {
        y += move.deltaY * pixelsPerStep; x
        += move.deltaX * pixelsPerStep;
     }

publicvoid moveBack(Move currentDirection) { if
        (currentDirection == Move.DOWN) {
        move(Move.UP);
        } elseif (currentDirection == Move.UP) {
        move(Move.DOWN);
        } elseif (currentDirection == Move.LEFT) {
        move(Move.RIGHT);
        } elseif (currentDirection == Move.RIGHT) {
        move(Move.LEFT);
        }
     }

publicint getSize() {
        returnSIZE;
     }

publicint getX() {
        returnx;
     }

publicint getY() {
        returny;
     }

publicint getColIndex() {
        return Floor.pixelToSquare(x);
     }

publicint getRowIndex() {
        return Floor.pixelToSquare(y);
     }
}
```

6.1.2.2 : ABSTRACT POWERUP

```
package game;

publicclass AbstractPowerup
{

privatefinalstaticintPOWERUP_SIZE = 30;
privatefinalintx;
privatefinalinty;
private String name = null;

public AbstractPowerup(intx, inty) {
	this.x = x;
	this.y = y;
    }

publicvoid addToPlayer(Player player) {
    }

publicint getPowerupSize() {
	returnPOWERUP_SIZE;
    }

publicint getX() {
	returnx;
    }

publicint getY() {
	returny;
    }

 public String getName() {
	returnname;
    }
}
```

6.1.2.3 : BOMB

```
package game;

publicclass Bomb
{
// Constants are static by definition.
privatefinalstaticintBOMBSIZE = 30;
privatefinalstaticintSTARTCOUNTDOWN = 100;
privateinttimeToExplosion = STARTCOUNTDOWN;
privatefinalintrowIndex;
privatefinalintcolIndex;
privateintexplosionRadius;
privatebooleanplayerLeft;

public Bomb(finalintrowIndex, finalintcolIndex, intexplosionRadius) {
```

```
this.rowIndex = rowIndex;
this.colIndex = colIndex; this.explosionRadius
= explosionRadius; playerLeft = false;
    }

publicint getRowIndex() {
returnrowIndex;
    }

publicint getColIndex() {
returncolIndex;
    }


publicstaticint getBOMBSIZE() {
returnBOMBSIZE;
    }

publicint getTimeToExplosion() {
returntimeToExplosion;
    }

publicvoid setTimeToExplosion(finalinttimeToExplosion) {
this.timeToExplosion = timeToExplosion;
    }

publicint getExplosionRadius() {
returnexplosionRadius;
    }

publicbooleanisPlayerLeft() {
returnplayerLeft;
    }

publicvoid setPlayerLeft(finalbooleanplayerLeft) {
this.playerLeft = playerLeft;
    }
}
```

6.1.2.4 : BOMB COUNTER PU

```
package game;

publicclass BombCounterPU extends AbstractPowerup
{

public BombCounterPU(introwIndex, intcolIndex) {
        super(colIndex, rowIndex);
    }

publicvoid addToPlayer(Player player) { intcurrentBombCount =
        player.getBombCount();
        player.setBombCount(currentBombCount + 1);
    }
public String getName() {
        final String name = "BombCounter";
        returnname;
    }
}
```

6.1.2.5   : BOMBERMAN COMPOUND

```
package game;


import javax.swing.*;

import java.awt.*;

import java.util.AbstractMap;

import java.util.EnumMap;



public class BombermanComponent extends JComponent implements FloorListener

{

    // Constants are static by definition.

private final static int SQUARE_SIZE = 40;

private final static int CHARACTER_ADJUSTMENT_FOR_PAINT = 15;

private final static int SQUARE_MIDDLE = SQUARE_SIZE/2;

private final static int BOMB_ADJUSTMENT_1 =5;

private final static int BOMB_ADJUSTMENT_2 =10;

    // Defining painting parameters

private final static int PAINT_PARAMETER_13 = 13;

private final static int PAINT_PARAMETER_15 = 15;

private final static int PAINT_PARAMETER_17 = 17;

private final static int PAINT_PARAMETER_18 = 18;

private final static int PAINT_PARAMETER_19 = 19;

private final static int PAINT_PARAMETER_20 = 20;

private final static int PAINT_PARAMETER_24 = 24;

private final Floor floor;

private final AbstractMap<FloorTile, Color> colorMap;
```

```
 public BombermanComponent(Floor floor) {
                     this.floor = floor;

                     colorMap = new EnumMap<>(FloorTile.class);
                     colorMap.put(FloorTile.FLOOR, Color.GREEN);
                     colorMap.put(FloorTile.UNBREAKABLEBLOCK, Color.BLACK);
                     colorMap.put(FloorTile.BREAKABLEBLOCK, Color.RED);
    }

    // This method is static since each square has the same size.
public static int getSquareSize() {
                     return SQUARE_SIZE;
    }

    // This method is static since each square has the same size.
public static int getSquareMiddle() {
                     return SQUARE_MIDDLE;
    }

public Dimension getPreferredSize() {
                     super.getPreferredSize();
                     return new Dimension(this.floor.getWidth() * SQUARE_SIZE,
this.floor.getHeight() * SQUARE_SIZE);
    }

public void floorChanged() {
                     repaint();
    }

    @Override
protected void paintComponent(Graphics g) {
                     super.paintComponent(g);
```

```
                    final Graphics2D g2d = (Graphics2D) g;


                    for (int rowIndex = 0; rowIndex < floor.getHeight();
rowIndex++) {
                    for (int colIndex = 0; colIndex < floor.getWidth(); colIndex++)
{

 g2d.setColor(colorMap.get(this.floor.getFloorTile(rowIndex, colIndex)));
                         if(floor.getFloorTile(rowIndex,
colIndex)==FloorTile.BREAKABLEBLOCK){
                          paintBreakableBlock(rowIndex, colIndex, g2d);
                         }
                         else if(floor.getFloorTile(rowIndex,
colIndex)==FloorTile.UNBREAKABLEBLOCK){
                         paintUnbreakableBlock(rowIndex, colIndex, g2d);
                         }
                         else{
                         paintFloor(rowIndex, colIndex, g2d);
                         }
                        }
                    }
                    // Paint player:
                    paintPlayer(floor.getPlayer(), g2d);


                    //Paint enemies
                    for (Enemy e: floor.getEnemyList()) {
                    paintEnemy(e, g2d);
                    }


                    //Paint powerups
                    for (AbstractPowerup p: floor.getPowerupList()) {
                    if (p.getName().equals("BombCounter")) {
                         g2d.setColor(Color.BLACK);
```

```
                            } else if (p.getName().equals("BombRadius")) {
                             g2d.setColor(Color.RED);
                            }
                        g2d.fillOval(p.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT,
p.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT, p.getPowerupSize(), p.getPowerupSize());
                        }


                        //Paint bombs
                        for (Bomb b: floor.getBombList()) {
                        g2d.setColor(Color.RED);
                        int bombX = floor.squareToPixel(b.getColIndex());
                        int bombY = floor.squareToPixel(b.getRowIndex());
                        g2d.fillOval(bombX + BOMB_ADJUSTMENT_1, bombY +
BOMB_ADJUSTMENT_1, Bomb.getBOMBSIZE(), Bomb.getBOMBSIZE());
                        g2d.setColor(Color.ORANGE);
                        g2d.fillOval(bombX + BOMB_ADJUSTMENT_2, bombY +
BOMB_ADJUSTMENT_1, BOMB_ADJUSTMENT_1, BOMB_ADJUSTMENT_2);
                        }


                        //Paint explosions
                        g2d.setColor(Color.ORANGE);
                        for (Explosion tup: floor.getExplosionCoords()) {
                        g2d.fillOval(floor.squareToPixel(tup.getColIndex()) +
BOMB_ADJUSTMENT_1, floor.squareToPixel(tup.getRowIndex()) +

BOMB_ADJUSTMENT_1, Bomb.getBOMBSIZE(), Bomb.getBOMBSIZE());
                        }
    }


private void paintBreakableBlock(int rowIndex, int colIndex, Graphics g2d){
                        g2d.setColor(Color.lightGray);
                        g2d.fillRect(colIndex * SQUARE_SIZE, rowIndex * SQUARE_SIZE,
SQUARE_SIZE, SQUARE_SIZE);
                        g2d.setColor(Color.BLUE);
```

```
                    g2d.drawLine(colIndex* SQUARE_SIZE+1, rowIndex*SQUARE_SIZE+10,
colIndex*SQUARE_SIZE+SQUARE_SIZE, rowIndex*SQUARE_SIZE+10);

                    g2d.drawLine(colIndex* SQUARE_SIZE+1,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE, colIndex*SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE);

                    g2d.drawLine(colIndex* SQUARE_SIZE+1,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10, colIndex*SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10);

                    g2d.drawLine(colIndex* SQUARE_SIZE+1,
rowIndex*SQUARE_SIZE+SQUARE_SIZE, colIndex*SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_SIZE);


                    g2d.drawLine(colIndex* SQUARE_SIZE+10, rowIndex*SQUARE_SIZE+1,
colIndex*SQUARE_SIZE+10, rowIndex*SQUARE_SIZE+10);

                    g2d.drawLine(colIndex* SQUARE_SIZE+SQUARE_MIDDLE+10,
rowIndex*SQUARE_SIZE+1, colIndex*SQUARE_SIZE+SQUARE_MIDDLE+10,
rowIndex*SQUARE_SIZE+10);


                    g2d.drawLine(colIndex* SQUARE_SIZE+1, rowIndex*SQUARE_SIZE+10,
colIndex*SQUARE_SIZE+1, rowIndex*SQUARE_SIZE+SQUARE_MIDDLE);

                    g2d.drawLine(colIndex* SQUARE_SIZE+SQUARE_MIDDLE+1,
rowIndex*SQUARE_SIZE+10, colIndex*SQUARE_SIZE+SQUARE_MIDDLE+1,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE);


                    g2d.drawLine(colIndex* SQUARE_SIZE+10,
rowIndex*SQUARE_SIZE+1+SQUARE_MIDDLE, colIndex*SQUARE_SIZE+10,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10);

                    g2d.drawLine(colIndex* SQUARE_SIZE+SQUARE_MIDDLE+10,
rowIndex*SQUARE_SIZE+1+SQUARE_MIDDLE, colIndex*SQUARE_SIZE+SQUARE_MIDDLE+10,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10);


                    g2d.drawLine(colIndex* SQUARE_SIZE+1,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10, colIndex*SQUARE_SIZE+1,
rowIndex*SQUARE_SIZE+SQUARE_SIZE);

                    g2d.drawLine(colIndex* SQUARE_SIZE+SQUARE_MIDDLE+1,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10, colIndex*SQUARE_SIZE+SQUARE_MIDDLE+1,
rowIndex*SQUARE_SIZE+SQUARE_SIZE);

    }
```

```
private void paintUnbreakableBlock(int rowIndex, int colIndex, Graphics g2d){

                     g2d.fillRect(colIndex * SQUARE_SIZE, rowIndex * SQUARE_SIZE,
SQUARE_SIZE, SQUARE_SIZE);

                     g2d.setColor(Color.DARK_GRAY);

                     g2d.drawLine(colIndex* SQUARE_SIZE, rowIndex*SQUARE_SIZE,
colIndex*SQUARE_SIZE+SQUARE_SIZE, rowIndex*SQUARE_SIZE);

                     g2d.drawLine(colIndex* SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_SIZE, colIndex*SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_SIZE);

                     g2d.drawLine(colIndex* SQUARE_SIZE, rowIndex*SQUARE_SIZE,
colIndex*SQUARE_SIZE, rowIndex*SQUARE_SIZE+SQUARE_SIZE);

                     g2d.drawLine(colIndex* SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE, colIndex*SQUARE_SIZE+SQUARE_SIZE,
rowIndex*SQUARE_SIZE+SQUARE_SIZE);

    }


private void paintFloor(int rowIndex, int colIndex, Graphics g2d){

                     g2d.setColor(Color.white);

                     g2d.fillRect(colIndex * SQUARE_SIZE, rowIndex * SQUARE_SIZE,
SQUARE_SIZE, SQUARE_SIZE);

                     g2d.setColor(Color.CYAN);

                     g2d.drawLine(colIndex* SQUARE_SIZE+5, rowIndex*SQUARE_SIZE+10,
colIndex * SQUARE_SIZE + 10, rowIndex * SQUARE_SIZE + 5);

                     g2d.drawLine(colIndex* SQUARE_SIZE+5,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE, colIndex * SQUARE_SIZE + SQUARE_MIDDLE, rowIndex
* SQUARE_SIZE + 5);

                     g2d.drawLine(colIndex* SQUARE_SIZE+5,
rowIndex*SQUARE_SIZE+SQUARE_MIDDLE+10, colIndex * SQUARE_SIZE + SQUARE_MIDDLE + 10,
rowIndex * SQUARE_SIZE + 5);

    }


private void paintEnemy(Enemy e, Graphics g2d){

                     // Paint body

                     g2d.setColor(Color.orange);

                     g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT, e.getSize(), e.getSize());

                     // Paint brows
```

```
                    g2d.setColor(Color.BLACK);
                    // Paint eyes
                    g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+4,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+9, 7, 7);

g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_19,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+9, 7, 7);
                    // Paint mouth
                    g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+5,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_20, PAINT_PARAMETER_20, 2);
                    // Fill eyes
                    g2d.setColor(Color.RED);
                    g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+5,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+10, 5, 5);

g2d.fillOval(e.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_20,
e.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+10, 5, 5);


    }


private void paintPlayer(Player player, Graphics g2d){
                    // Paint hat
                    g2d.setColor(Color.BLUE);

g2d.fillOval(player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_15,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT-2, PAINT_PARAMETER_15,
PAINT_PARAMETER_15);
                    // Paint body
                    g2d.setColor(Color.LIGHT_GRAY);
                    g2d.fillOval(player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT, player.getSize(), player.getSize());
                    // Paint face
                    g2d.setColor(Color.PINK);
                    g2d.fillOval(player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+3,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+3, player.getSize()-6,
player.getSize()-6);
```

```
                        // Paint eyes

                        g2d.setColor(Color.BLACK);

                        g2d.drawLine(player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+10,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+10,
player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+10,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_18);


g2d.drawLine(player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_20,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+10,
player.getX()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_20,
player.getY()-CHARACTER_ADJUSTMENT_FOR_PAINT+PAINT_PARAMETER_18);

   }

}
```

6.1.2.6 : BOMBERMAN FRAME

```
package game;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

import java.awt.event.KeyEvent;
publicclassBombermanFrameextends JFrame
{
private Floor floor;
private BombermanComponent bombermanComponent;

public BombermanFrame(final String title, Floor floor) throws HeadlessException {
        super(title);
        this.floor = floor;
        this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        bombermanComponent = new BombermanComponent(floor);
        floor.createPlayer(bombermanComponent, floor);
        setKeyStrokes();

        this.setLayout(newBorderLayout());
        this.add(bombermanComponent, BorderLayout.CENTER);
        this.pack();
        this.setVisible(true);
    }

public BombermanComponent getBombermanComponent() {
        returnbombermanComponent;
    }

privatebooleanaskUser(String question) {
        return JOptionPane.showConfirmDialog(null, question, "",
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION;
    }
privatevoid setKeyStrokes() {

        KeyStroke stroke = KeyStroke.getKeyStroke(KeyEvent.VK_W,
Toolkit.getDefaultToolkit().~~getMenuShortcutKeyMask~~());
        bombermanComponent.getInputMap().put(stroke, "q");
        bombermanComponent.getActionMap().put("q", quit);
    }

privatefinal Action quit = newAbstractAction()
    {
        publicvoid actionPerformed(ActionEvent e) { dispose();

        }
    };
}
```

6.1.2.7 BOMB RADIUS PU

```
package game;

publicclass BombRadiusPU extends AbstractPowerup
{

public BombRadiusPU(introwIndex, intcolIndex) {
        super(colIndex, rowIndex);
    }

publicvoid addToPlayer(Player player) { intcurrentExplosionRadius
        = player.getExplosionRadius();
        player.setExplosionRadius(currentExplosionRadius + 1);
    }

public String getName() {
        final String name = "BombRadius";
        returnname;
    }
}
```

ENEMY

```
package game;

publicclass Enemy extends AbstractCharacter
{
private Move currentDirection;

public Enemy(intx, inty, booleanvertical) {
super(x, y, 1);
currentDirection = randomDirection(vertical);
    }

publicvoid changeDirection() {
if (currentDirection == Move.DOWN) {
currentDirection = Move.UP;
        } elseif (currentDirection == Move.UP) {
currentDirection = Move.DOWN;
```

```
        } elseif (currentDirection == Move.LEFT) {
currentDirection = Move.RIGHT;
        } else {
currentDirection = Move.LEFT;
        }
    }

public Move getCurrentDirection() {
returncurrentDirection;
    }

private Move randomDirection(booleanvertical) {
assert Move.values().length == 4;
intpick = (int) (Math.random() * (Move.values().length-2));
if(vertical) {
return Move.values()[pick];
        }
else{
return Move.values()[pick+2];
        }


    }
}
```

6.1.2.8 : ENGINE

```
package game;



import javax.swing.*;

import java.awt.event.ActionEvent;

import java.io.BufferedReader; import

java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collection;



public final class Engine

{

private static final int TIME_STEP = 30; private

static int width = 10;

private static int height = 10;

private static int nrOfEnemies = 5;
private static Timer clockTimer = null;
private Engine() {}

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

public static void startGame() {
                        Floor floor = new Floor(width, height, nrOfEnemies); BombermanFrame
                        frame = new BombermanFrame("Bomberman", floor);
                        frame.setLocationRelativeTo(null);
                        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                        floor.addFloorListener(frame.getBombermanComponent());

                        Action doOneStep = new AbstractAction()
                        {
                        public void actionPerformed(ActionEvent e) {
                             tick(frame, floor);
                            }
                        };
                        clockTimer = new Timer(TIME_STEP, doOneStep);
                        clockTimer.setCoalesce(true); clockTimer.start();
    }

private static void gameOver(BombermanFrame frame, Floor floor) {
                        clockTimer.stop();
                        frame.dispose();
                        startGame();
    }
```

```
private static void tick(BombermanFrame frame, Floor floor) {

                        if (floor.getIsGameOver()) {

                        gameOver(frame, floor);

                        } else { floor.moveEnemies();

                        floor.bombCountdown();

                        floor.explosionHandler();

                        floor.characterInExplosion();

                        floor.notifyListeners();

                        }

    }

}
```

6.1.2.9 : EXPLOSION

```
package game;

publicclass Explosion
{
privateintrowIndex;
privateintcolIndex;
privateintduration = 5;

public Explosion(introwIndex, intcolIndex)
    {
this.rowIndex = rowIndex;
this.colIndex = colIndex;
    }

publicint getRowIndex() {
returnrowIndex;
    }

publicint getColIndex() {
returncolIndex;
    }

publicint getDuration() {
returnduration;
    }

publicvoid setDuration(finalintduration) {
this.duration = duration;
    }
```

```
}
```

6.1.2.10 : FLOOR

```
package game;

import game.AbstractCharacter.Move; import

java.util.*;



public class Floor {

    // Constants are static by definition.

private final static double CHANCE_FOR_BREAKABLE_BLOCK = 0.4; private

final static double CHANCE_FOR_RADIUS_POWERUP = 0.2; private final

static double CHANCE_FOR_COUNTER_POWERUP = 0.8; private final

FloorTile[][] tiles;

private int width; private

int height;

private Collection<FloorListener> floorListeners = new ArrayList<>(); private

Player player = null;

private Collection<Enemy> enemyList = new ArrayList<>(); private

List<Bomb> bombList= new ArrayList<>();

private Collection<AbstractPowerup> powerupList = new ArrayList<>(); private

Collection<Bomb> explosionList= new ArrayList<>();

private Collection<Explosion> explosionCoords= new ArrayList<>();

private boolean isGameOver = false;


public Floor(int width, int height, int nrOfEnemies) {

                        this.width = width;

                        this.height = height;

                        this.tiles = new FloorTile[height][width];
```

```
placeBreakable(); placeUnbreakableAndGrass();
```

```
			spawnEnemies(nrOfEnemies);
    }

public static int pixelToSquare(int pixelCoord){
			return ((pixelCoord + BombermanComponent.getSquareSize()-1) /
BombermanComponent.getSquareSize())-1;
    }

public FloorTile getFloorTile(int rowIndex, int colIndex) {
			return tiles[rowIndex][colIndex];
    }

public int getWidth() {
			return width;
    }

public int getHeight() {
			return height;
    }

public Player getPlayer() {
			return player;
    }

public Collection<Enemy> getEnemyList() {
			return enemyList;
    }

public Iterable<Bomb> getBombList() {
			return bombList;
    }
```

```
public int getBombListSize() {
                        return bombList.size();
    }

public Iterable<AbstractPowerup> getPowerupList() {
                        return powerupList;
    }

public Iterable<Explosion> getExplosionCoords() {
                        return explosionCoords;
    }

public boolean getIsGameOver() {
                        return isGameOver;
    }

public void setIsGameOver(boolean value) {
                        isGameOver = value;
    }

public void addToBombList(Bomb bomb) {
                        bombList.add(bomb);
    }

public void createPlayer(BombermanComponent bombermanComponent, Floor floor){ player =
                        new Player(bombermanComponent, floor);
    }

public int squareToPixel(int squareCoord){
                        return squareCoord * BombermanComponent.getSquareSize();
}
```

```
public void moveEnemies() {
                        if (enemyList.isEmpty()) {
                        isGameOver = true;
                        }
                        for (Enemy e: enemyList){
                            Move currentDirection = e.getCurrentDirection();

                        if (currentDirection == Move.DOWN) {
                             e.move(Move.DOWN);
                            } else if (currentDirection == Move.UP) { e.move(Move.UP);
                            } else if (currentDirection == Move.LEFT) {
                             e.move(Move.LEFT);
                            } else {
                             e.move(Move.RIGHT);
                            }

                        if (collisionWithBlock(e)) {
                             e.changeDirection();
                            }

                        if (collisionWithBombs(e)) {
                             e.changeDirection();
                            }

                        if (collisionWithEnemies()) { isGameOver
                             = true;
                            }
                        }
    }
```

```
public void addFloorListener(FloorListener bl) {
                        floorListeners.add(bl);
    }

public void notifyListeners() {
                        for (FloorListener b : floorListeners) {
                        b.floorChanged();
                        }
    }



public void bombCountdown(){
                        Collection<Integer> bombIndexesToBeRemoved = new ArrayList<>();
                        explosionList.clear();
                        int index = 0;
                        for (Bomb b: bombList) { b.setTimeToExplosion(b.getTimeToExplosion() -
                        1);
                        if(b.getTimeToExplosion() == 0){
                             bombIndexesToBeRemoved.add(index);
                             explosionList.add(b);
                            }
                        index++;
                        }
                        for (int i: bombIndexesToBeRemoved){bombList.remove(i);}
    }

public void explosionHandler(){
                        Collection<Explosion> explosionsToBeRemoved = new
ArrayList<>();
```

```
for (Explosion e:explosionCoords) {

e.setDuration(e.getDuration()-1);
```

```
                    if(e.getDuration()==0){
                         explosionsToBeRemoved.add(e);
                        }
                    }
                    for (Explosion e: explosionsToBeRemoved){explosionCoords.remove(e);}


                    for (Bomb e: explosionList) {
                    int eRow = e.getRowIndex(); int
                    eCol = e.getColIndex(); boolean
                    northOpen = true; boolean
                    southOpen = true; boolean
                    westOpen = true; boolean
                    eastOpen = true;
                    explosionCoords.add(new Explosion(eRow, eCol));
                    for (int i = 1; i < e.getExplosionRadius()+1; i++) { if
                          (eRow - i >= 0 && northOpen) {
                         northOpen = bombCoordinateCheck(eRow-i, eCol, northOpen);
                         }
                          if (eRow - i <= height && southOpen) {
                         southOpen = bombCoordinateCheck(eRow+i, eCol, southOpen);
                         }
                          if (eCol - i >= 0 && westOpen) {
                         westOpen = bombCoordinateCheck(eRow, eCol-i, westOpen);
                         }
                          if (eCol + i <= width && eastOpen) {
                         eastOpen = bombCoordinateCheck(eRow, eCol+i, eastOpen);
                         }
                        }
                    }
          }
```

```
public void playerInExplosion(){
                        for (Explosion tup:explosionCoords) { if(collidingCircles(player,
                        squareToPixel(tup.getColIndex()),
squareToPixel(tup.getRowIndex()))){
                              isGameOver = true;
                            }
                        }
    }


public void enemyInExplosion(){
                        for (Explosion tup:explosionCoords) {
                            Collection<Enemy> enemiesToBeRemoved = new ArrayList<>(); for
                        (Enemy e : enemyList) {
                              if(collidingCircles(e, squareToPixel(tup.getColIndex()),
squareToPixel(tup.getRowIndex()))){
                              enemiesToBeRemoved.add(e);
                              }
                            }
                        for (Enemy e: enemiesToBeRemoved ) {
                              enemyList.remove(e);
                            }
                        }
    }


public void characterInExplosion(){
                        playerInExplosion();
                        enemyInExplosion();
    }


private void placeBreakable () {
                        for (int i = 0; i < height; i++) { for
```

```
(int j = 0; j < width; j++) {
```

```
                              double r = Math.random();
                              if (r < CHANCE_FOR_BREAKABLE_BLOCK) {
                              tiles[i][j] = FloorTile.BREAKABLEBLOCK;
                              }
                             }
                        }
                        clearSpawn();
    }

private void clearSpawn () {
                        tiles[1][1]  =  FloorTile.FLOOR;
                        tiles[1][2]  =  FloorTile.FLOOR;
                        tiles[2][1] = FloorTile.FLOOR;
    }

private void spawnPowerup (int rowIndex, int colIndex) {
                        double r = Math.random();
                        if (r < CHANCE_FOR_RADIUS_POWERUP) {
                        powerupList.add(new BombRadiusPU(squareToPixel(rowIndex) +
BombermanComponent.getSquareMiddle(), squareToPixel(colIndex) +
BombermanComponent.getSquareMiddle()));
                        } else if (r > CHANCE_FOR_COUNTER_POWERUP) {
                        powerupList.add(new BombCounterPU(squareToPixel(rowIndex) +
BombermanComponent.getSquareMiddle(), squareToPixel(colIndex) +
BombermanComponent.getSquareMiddle()));
                        }
    }

private void placeUnbreakableAndGrass () {
                        for (int i = 0; i < height; i++) { for
                        (int j = 0; j < width; j++) {
                              //Makes frame of unbreakable
```

```
                              if ((i == 0) || (j == 0) || (i == height - 1) || (j ==
width - 1) || i % 2 == 0 && j % 2 == 0) {
                              tiles[i][j] = FloorTile.UNBREAKABLEBLOCK;
                                    //Every-other unbreakable
                              } else if (tiles[i][j] != FloorTile.BREAKABLEBLOCK) { tiles[i][j]
                              = FloorTile.FLOOR;
                              }
                             }
                        }
    }

private void spawnEnemies (int nrOfEnemies) {
                        for (int e = 0; e < nrOfEnemies; e++){
                        while(true) {
                              int randRowIndex = 1 + (int) (Math.random() * (height -
2));
                              int randColIndex = 1 + (int) (Math.random() * (width -
2));
                              if(getFloorTile(randRowIndex, randColIndex) !=
FloorTile.FLOOR){
                              continue;
                              }

if(randRowIndex==1&&randColIndex==1||randRowIndex==1&&randColIndex==2||randRowIndex==
2&&randColIndex==1){
                              continue;
                              }
                              if((randRowIndex % 2)==0){
                              enemyList.add(new Enemy(squareToPixel(randColIndex) +
BombermanComponent.getSquareMiddle(), squareToPixel(randRowIndex) +
BombermanComponent.getSquareMiddle(), true));
                              }
                              else{
```

```
                              enemyList.add(new Enemy(squareToPixel(randColIndex) +
BombermanComponent.getSquareMiddle(), squareToPixel(randRowIndex) +
BombermanComponent.getSquareMiddle(), false));

                              }

                              break;

                             }

                        }

    }





public boolean collisionWithEnemies(){
                        for (Enemy enemy : enemyList) {
                        if(collidingCircles(player,
enemy.getX()-BombermanComponent.getSquareMiddle(),
enemy.getY()-BombermanComponent.getSquareMiddle())){

                              return true;

                             }

                        }

                        return false;

    }


public boolean collisionWithBombs(AbstractCharacter abstractCharacter) { boolean

                        playerLeftBomb = true;


                        for (Bomb bomb : bombList) {

                        if (abstractCharacter instanceof Player) {

                              playerLeftBomb = bomb.isPlayerLeft();

                             }

                        if(playerLeftBomb && collidingCircles(abstractCharacter,
squareToPixel(bomb.getColIndex()), squareToPixel(bomb.getRowIndex()))){

                              return true;
```

```
}
```

```
                        }
                        return false;
    }



public boolean collisionWithBlock(AbstractCharacter abstractCharacter){
                        //Maybe create if statements to only check nearby squares for
                        (int i = 0; i < height; i++) {
                        for (int j = 0; j < width; j++) { if(getFloorTile(i,
                             j) != FloorTile.FLOOR){
                             boolean isIntersecting = squareCircleInstersect(i, j,
abstractCharacter);
                             if (isIntersecting) {
                                    return true;
                                 }
                             }
                            }
                        }
                        return false;
    }


public void collisionWithPowerup() {
                        for (AbstractPowerup powerup : powerupList) {
                        if(collidingCircles(player,
powerup.getX()-BombermanComponent.getSquareMiddle(),
powerup.getY()-BombermanComponent.getSquareMiddle())){
                             powerup.addToPlayer(player);
                             powerupList.remove(powerup);
                             break;
                            }
                        }
```

```
}
```

```
public boolean squareHasBomb(int rowIndex, int colIndex){
                        for (Bomb b: bombList) {
                        if(b.getRowIndex()==rowIndex && b.getColIndex()==colIndex){
                             return true;
                            }
                        }
                        return false;
    }



public void checkIfPlayerLeftBomb(){
                        for (Bomb bomb: bombList) {
                        if(!bomb.isPlayerLeft()){
                             if(!collidingCircles(player, squareToPixel(bomb.getColIndex()),
squareToPixel(bomb.getRowIndex()))){
                             bomb.setPlayerLeft(true);
                             }
                            }
                        }
    }


private boolean bombCoordinateCheck(int eRow, int eCol, boolean open){
                        if(tiles[eRow][eCol] != FloorTile.FLOOR){open = false;}
                        if(tiles[eRow][eCol] == FloorTile.BREAKABLEBLOCK){ tiles[eRow][eCol] =
                        FloorTile.FLOOR;
                        spawnPowerup(eRow, eCol);
                        }
                        if(tiles[eRow][eCol] !=
FloorTile.UNBREAKABLEBLOCK){explosionCoords.add(new Explosion(eRow, eCol));}
```

```
                        return open;
     }


private boolean collidingCircles(AbstractCharacter abstractCharacter, int x, int y){ int a =
                        abstractCharacter.getX() - x -
BombermanComponent.getSquareMiddle();
                        int b = abstractCharacter.getY() - y -
BombermanComponent.getSquareMiddle();
                        int a2 = a * a; int
                        b2 = b * b;
                        double c = Math.sqrt(a2 + b2);
                        return(abstractCharacter.getSize() > c);
     }


private boolean squareCircleInstersect(int row, int col, AbstractCharacter abstractCharacter)
{

                        int characterX = abstractCharacter.getX(); int
                        characterY = abstractCharacter.getY();

                        int circleRadius = abstractCharacter.getSize() / 2; int
                        squareSize = BombermanComponent.getSquareSize(); int
                        squareCenterX = (col*squareSize)+(squareSize/2); int
                        squareCenterY = (row*squareSize)+(squareSize/2);

                        int circleDistanceX = Math.abs(characterX - squareCenterX); int
                        circleDistanceY = Math.abs(characterY - squareCenterY);

                        if (circleDistanceX > (squareSize/2 + circleRadius)) { return
false; }
                        if (circleDistanceY > (squareSize/2 + circleRadius)) { return
false; }
```

```
                      if (circleDistanceX <= (squareSize/2)) { return true; } if

                      (circleDistanceY <= (squareSize/2)) { return true; }


                      int cornerDistance = (circleDistanceX - squareSize/2)^2 +

                                                                              (circleDistanceY -
squareSize/2)^2;


                        return (cornerDistance <= (circleRadius^2));

    }

}
```

6.1.2.11 : FLOOR LISTENER

```
package game;

publicinterface FloorListener
{
void floorChanged();
}package game;

publicenum FloorTile        {

FLOOR,

UNBREAKABLEBLOCK,

BREAKABLEBLOCK
}
```

6.1.2.12 : FLOOR TILE

```
package game;

publicenum FloorTile
       {

FLOOR,

UNBREAKABLEBLOCK,

BREAKABLEBLOCK
}
```

6.1.2.13 : PLAYER

```
package game;


import javax.swing.*;

import java.awt.event.ActionEvent;




public class Player extends AbstractCharacter

{


private final static int PLAYER_START_X = 60; private

final static int PLAYER_START_Y = 60; private final

static int PLAYER_PIXELS_BY_STEP = 4; private int

explosionRadius;

private int bombCount;

private Floor floor;


public Action up = new AbstractAction() {

                        public void actionPerformed(ActionEvent e) { movePlayer(Move.UP);


                        }

    };


public Action right = new AbstractAction() {

                        public void actionPerformed(ActionEvent e) { movePlayer(Move.RIGHT);


                        }

    };
```

```
public Action down = new AbstractAction() {
                    public void actionPerformed(ActionEvent e) { movePlayer(Move.DOWN);

                    }
    };

public Action left = new AbstractAction() {
                    public void actionPerformed(ActionEvent e) { movePlayer(Move.LEFT);

                    }
    };


public Action dropBomb = new AbstractAction()
    {
                    public void actionPerformed(ActionEvent e) {
                    if(!floor.squareHasBomb(getRowIndex(), getColIndex()) &&
floor.getBombListSize() < getBombCount()){
                         floor.addToBombList(new Bomb(getRowIndex(), getColIndex(),
getExplosionRadius()));
                        }
                    floor.notifyListeners();
                    }
    };

public Player(BombermanComponent bombermanComponent, Floor floor) {
                    super(PLAYER_START_X, PLAYER_START_Y, PLAYER_PIXELS_BY_STEP);
                    explosionRadius = 1;
                    bombCount = 1;
```

```
                        this.floor = floor;
                        setPlayerButtons(bombermanComponent);
    }

public void setPlayerButtons(BombermanComponent bombermanComponent){

bombermanComponent.getInputMap().put(KeyStroke.getKeyStroke("RIGHT"), "moveRight");

bombermanComponent.getInputMap().put(KeyStroke.getKeyStroke("LEFT"), "moveLeft");

bombermanComponent.getInputMap().put(KeyStroke.getKeyStroke("UP"), "moveUp");

bombermanComponent.getInputMap().put(KeyStroke.getKeyStroke("DOWN"), "moveDown");

bombermanComponent.getInputMap().put(KeyStroke.getKeyStroke("SPACE"), "dropBomb");
                        bombermanComponent.getActionMap().put("moveRight", right);
                        bombermanComponent.getActionMap().put("moveLeft", left);
                        bombermanComponent.getActionMap().put("moveUp", up);
                        bombermanComponent.getActionMap().put("moveDown", down);
                        bombermanComponent.getActionMap().put("dropBomb", dropBomb);
    }

public int getBombCount() {
                        return bombCount;
    }

public void setBombCount(int bombCount) {
                        this.bombCount = bombCount;
    }

public int getExplosionRadius() {
```

```
                return explosionRadius;

    }
```

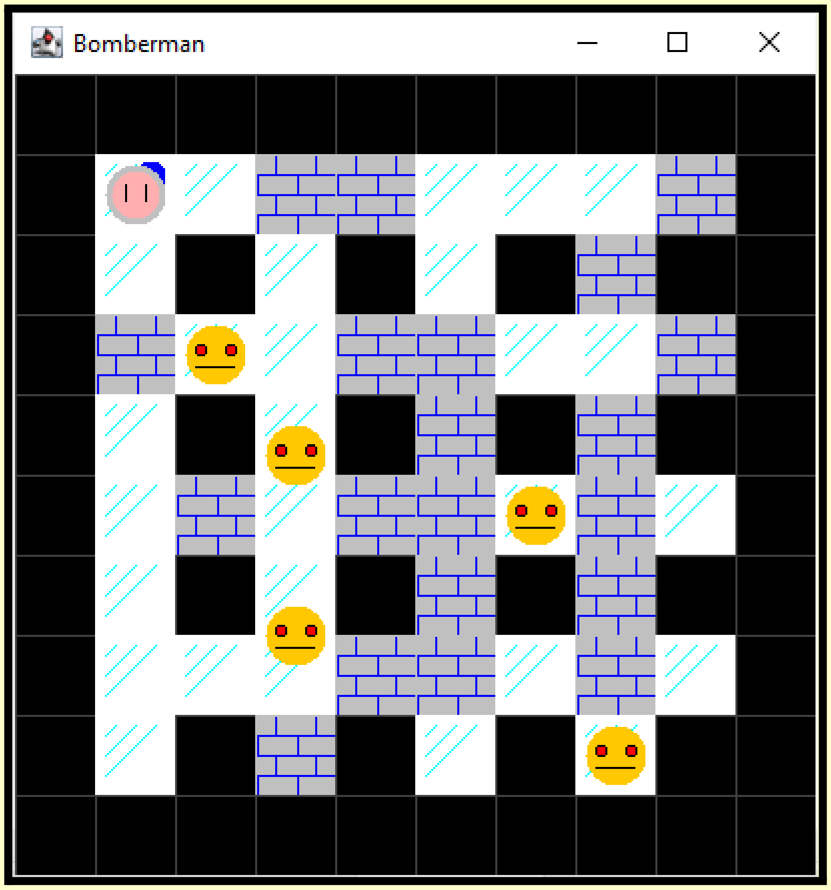
```
public void setExplosionRadius(int explosionRadius) {
			this.explosionRadius = explosionRadius;
    }


private void movePlayer(Move move) {
			move(move);
			if(floor.collisionWithBlock(this)){
			moveBack(move);
			}
			if(floor.collisionWithBombs(this)){
			moveBack(move);
			}
			if(floor.collisionWithEnemies()){
			floor.setIsGameOver(true);
			}


			floor.checkIfPlayerLeftBomb();
			floor.collisionWithPowerup();
			floor.notifyListeners();
    }


}
```

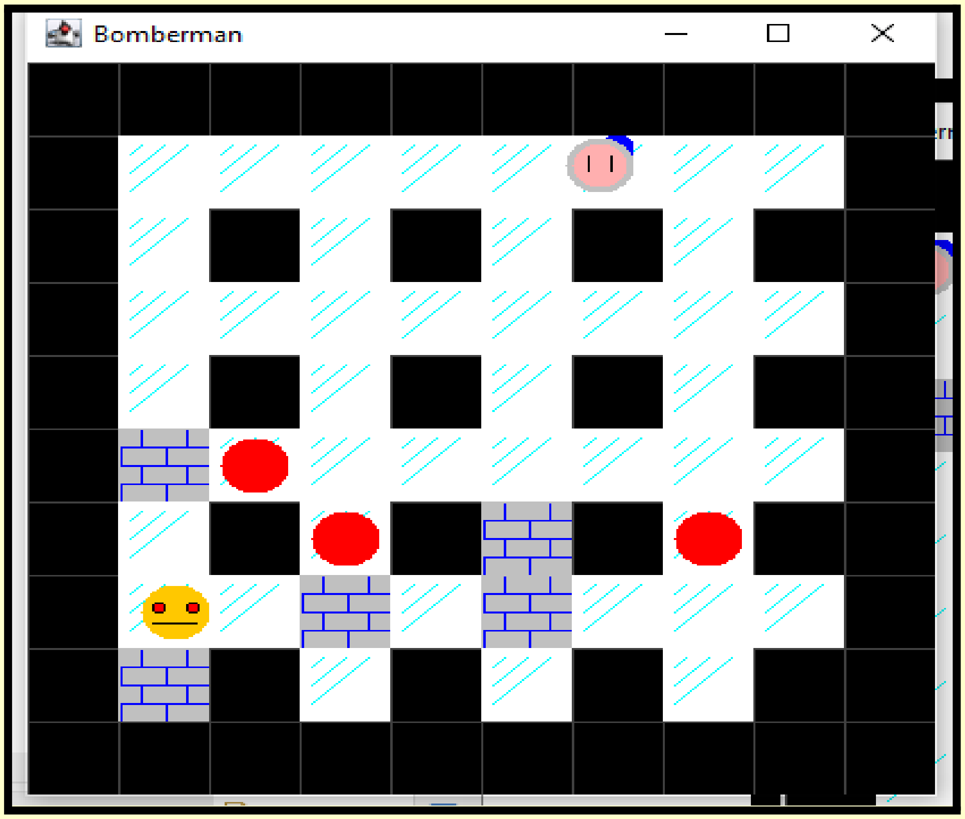
6.1.3 : MODULE

6.1.3.1 : MODULE INFO JAVA

```
module bomberman
{
	requires java.desktop;
}
```

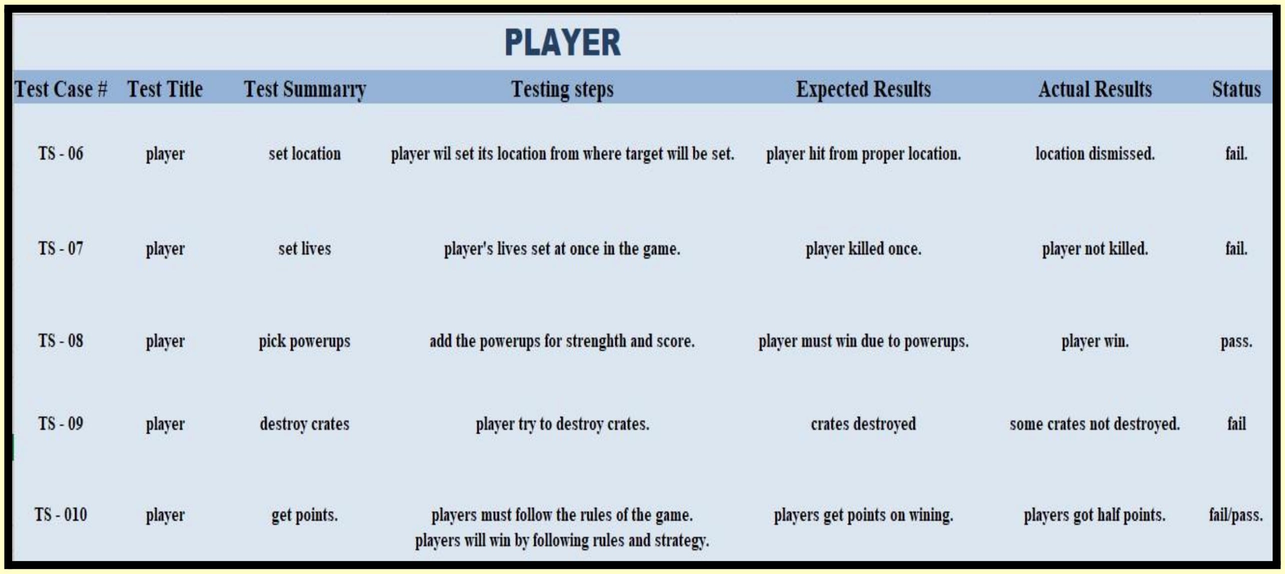
6.1.4 : OUTPUT SCREEN





7. TESTING

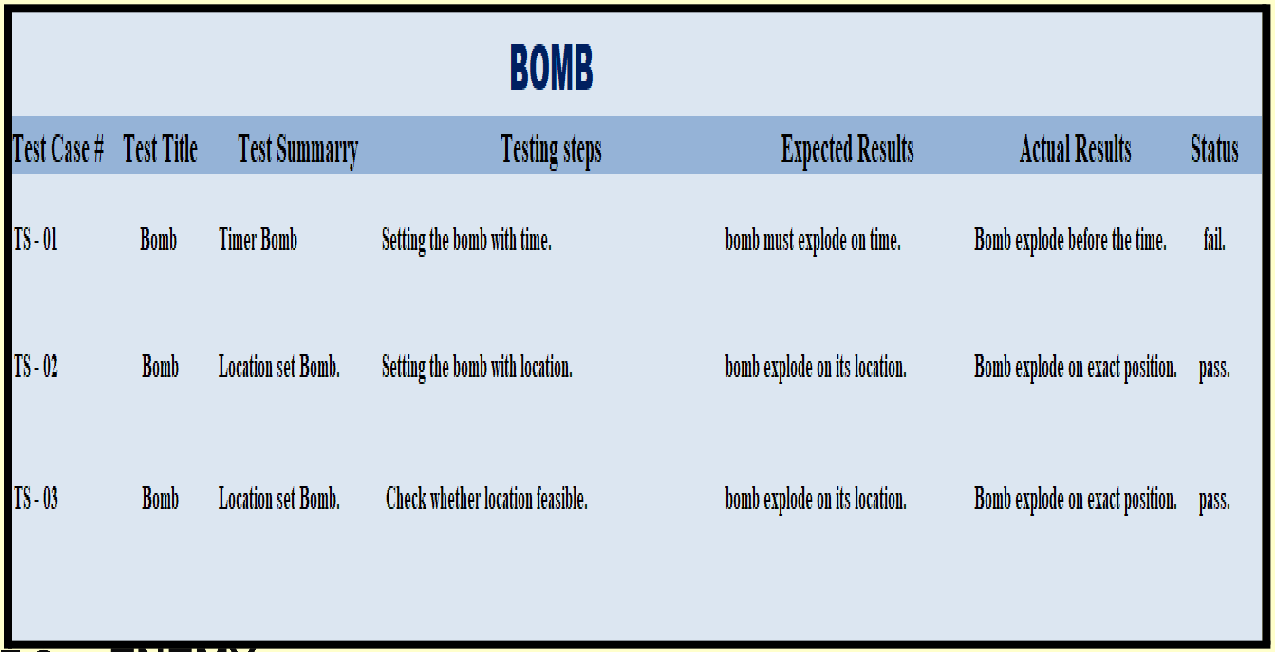
7.1 : PLAYER



PLAYER

Test Case #	Test Title	Test Summary	Testing steps	Expected Results	Actual Results	Status
TS - 06	player	set location	player wil set its location from where target will be set.	player hit from proper location.	location dismissed.	fail.
TS - 07	player	set lives	player's lives set at once in the game.	player killed once.	player not killed.	fail.
TS - 08	player	pick powerups	add the powerups for strenghth and score.	player must win due to powerups.	player win.	pass.
TS - 09	player	destroy crates	player try to destroy crates.	crates destroyed	some crates not destroyed.	fail
TS - 010	player	get points.	players must follow the rules of the game. players will win by following rules and strategy.	players get points on wining.	players got half points.	fail/pass.

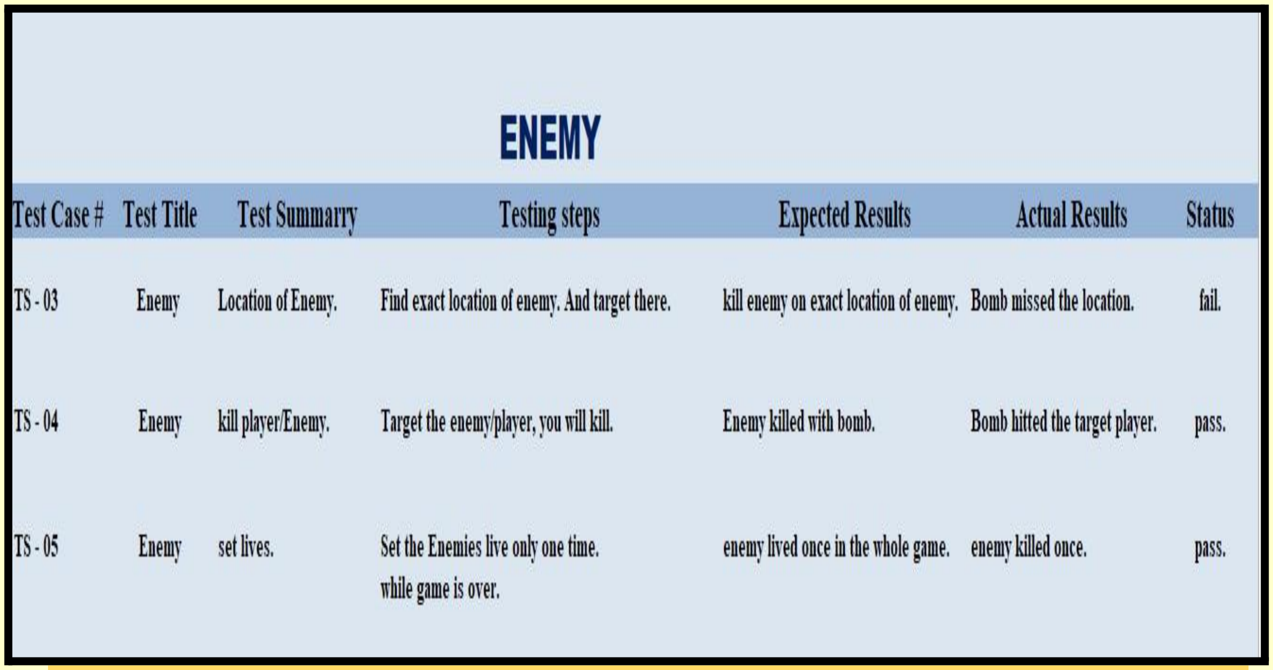
7.2 : BOMB



BOMB

Test Case #	Test Title	Test Summarry	Testing steps	Expected Results	Actual Results	Status
TS - 01	Bomb	Timer Bomb	Setting the bomb with time.	bomb must explode on time.	Bomb explode before the time.	fail.
TS - 02	Bomb	Location set Bomb.	Setting the bomb with location.	bomb explode on its location.	Bomb explode on exact position.	pass.
TS - 03	Bomb	Location set Bomb.	Check whether location feasible.	bomb explode on its location.	Bomb explode on exact position.	pass.

7.3 : ENEMY



ENEMY

Test Case #	Test Title	Test Summarry	Testing steps	Expected Results	Actual Results	Status
TS - 03	Enemy	Location of Enemy.	Find exact location of enemy. And target there.	kill enemy on exact location of enemy.	Bomb missed the location.	fail.
TS - 04	Enemy	kill player/Enemy.	Target the enemy/player, you will kill.	Enemy killed with bomb.	Bomb hitted the target player.	pass.
TS - 05	Enemy	set lives.	Set the Enemies live only one time. while game is over.	enemy lived once in the whole game.	enemy killed once.	pass.

8. APPENDIX

8.1 : GLOSSARY

- **Abstract Character:**

  We can hide pixels values by using protected function.

- **Delta**

  To provide the way to add modify or remove java code.

- **Public Enum**

  Special datatype that enable for a variable to set pre define condition.

- **Power up:**

  To calculate a number raised to the power of some other number.

- **Counter:**

  To count the repetitions or to know about in which repetition we are in.

- **Time Stamp:**

  Adds the ability to hold the SQL TIMESTAMP fractional seconds value.

- **Breakable Block:**

  Block which can be break by the bomb explosion.

- **Unbreakable Block:**

  Block or crates which are fixed and cannot be explode by bomb explosion.

9. REFERANCES

1. https://en.wikipedia.org/wiki/Bomberman
2. https://bomberman.fandom.com/wiki/Bomberman_(series)