**Documentation for Missile Command Game**

**EECE 1610**

**Group: Marielle Billig, Peter Dobbs, Lauren Ernst**

**Due: April 30, 2015**

**Requirements**

The program should resemble a Mission Command game. By shooting missiles from the blue bases located at the bottom of the screen, the player will aim to destroy the bombs random falling from the sky before any bomb hits one of the bases. If the player shoots a bomb with their missile, an explosion will occur, the bomb and missile will disappear from the screen, and the score will be incremented by 100. Each base is given a specific number of missiles they can shoot. If a base is hit by a bomb, the base will disappear and will not be operational anymore. As the player continues in the game, they are given 10 missiles to shoot per level. When a player reaches a score of 500 they move on to level 2 where a spaceship becomes a flying object in the game. The orange spaceship can be destroyed by the player’s missiles for an increment of 200 to the score. As the player continues further in the game, the levels increase, and the pink spaceship enters the game screen. The pink spaceship can be destroyed by the player’s missiles for an increment of 300 to the score. Both spaceships also shoot bombs which can destroy the bases. The player loses the game when all of the bases are gone, or when the player has shot all of their missiles given the level they are at in the game. After the player has finished the game, if they have received a high score, they are able to type in their name and save their high score. This score will continue to be displayed each time the game is played.

**------------------------------------------------------------------------------------------------------------------------------------------**

**Description of Classes**

**Main Class**

Purpose: Set up window and visual components, extends JFrame

**Variables:**

* FRAME\_WIDTH = final int 800
* FRAME\_HEIGHT = final int 800
* GAME\_WIDTH = final int 700
* GAME\_HEIGHT = final int 700
* scorePanel: JPanel
* groundPanel: JPanel
* gamePanel: JPanel
* currentScoreLabel: JLabel
* missileCountLabel: JLabel
* highScoreLabel: JLabel
* gameVet1, gameVet2, gameVet3: JLabels
* flyingObjectComponent: JComponent
* score: int current score
* nMissiles: int
* bombX: int current x coordinate of bomb
* bombY: int current y coordinate of bomb

**Constructor:**

Sets preferences of JFrame (title, default close operation, size, resizable)

Note: Main extends JFrame

Sets preferences of gamePanel and adds it to JFrame

Sets preferences of groundPanel and add it to JFrame

Sets preferences of scorePanel, sets text for JLabels, adds JLabels to scorePanel, and adds scorePanel to JFrame

Packs and sets the JFrame visibile

**Methods:**

main()

Create new Main object

**------------------------------------------------------------------------------------------------------------------------------------------**

**Missile Class**

Purpose: class for missile objects are corresponding methods

**Variables:**

* xCoord: int x coordinate of missile at any point, original starting point
* yCoord: int y coordinate of missile at any point, original starting point
* missileX: int x coordinate of final destination
* missileY: int y corrdinate of final destination
* missileSpeed: double speed of missile during any level
* MISSILE\_SIZE = int 20

**Constructor:**

Takes three ints and set the first two to xCoord and yCoord respectively. The last int is missileX. missileY and missileSpeed are set to 625 and 0.15.

getHighScore()

calls the method to get the high score

**Methods:**

magentaMissile(Graphics g)

Defines two new doubles missileDX and missileDY that are decremented from missileX and missileY every time the method is executed. It also sets the color to magenta and places and circle (diameter MISSILE\_SIZE) at the point missileX, missileY.

getXCoord()

Returns xCoord

getYCoord()

Returns yCoord

**------------------------------------------------------------------------------------------------------------------------------------------**

**FlyingObjects Class**

**Purpose**: This class is intended to create the graphics of all the flying objects in the game. Different methods are created for each object, and separate objects have specific variable names to define coordinates of the graphics.

**Variables:**

* lastX = 0, lastY = 0
* startX = 10, startY = 10
* baseX = 150, baseY = 625
* bombX = 0, bombY = 0
* bombSpeed
* baseNumber
* missileX = 150, missileY = 650
* xCoord, yCoord
* base1Exists, base2Exists, base3Exists
* missile1, missile2
* MListener mouseListener
* Thread animationThread
* Color1 : Color, Color2 : Color, Color3 : Color
* scoreColor, font, fontMetrics
* bases = 3; missileCount
* int lvl;
* BOMB\_SIZE=20

Initial variables are created for initial mouse click (xCoord and yCoord), base (baseX and baseY), bomb (bombX and bombY), and missile coordinates (missileX and missileY). 3 boolean variables are also created to represent if any of the bases exist after collisions with bombs (base1Exists, base2Exists, base3Exists). Corresponding color variables for each base are also defined (color1, color2, color3). A missile object, mouseListener, and animationThread are also created. Variables for Font, FontMetrics, missileCount, and score are also created.

**Methods:**

**FlyingObjects()**

setBound()

set bounds of flying objects

setOpaque()

run() new animationThread

use a try catch to implement a Thread.sleep(150)

new Missile(xCoord, yCoord, baseX)

method from the Missile class with MouseClicker coordinates and a base as parameters

OrangeSpaceShip()

Create new orange spaceship object

PinkSpaceship()

Create new pink spaceship object

Font()

set font as Gautami

new MListener() and addMouseListener()

create and add a new mouseListener to the program, so the X and Y coordinates of every mouse click can be obtained

**paintComponent**

createScoreBoard(g);

create a score board with method

g.setColor() and gfillRect()

method to set color and fill shape of the 3 bases with the boundaries of the desired shape

greyBomb()

create a greyBomb with method

if() and missile.magentaMissile()

create magenta missile with method to shoot from base on level 1

else if() and missile2.magentaMissile(g)

create another magenta missile with method to shoot from base on level 2 or higher

orangeSpaceShip() and if() and missileOrangeCollision()

create orange spaceship with method, use if statement to set events, and call collision method

pinkSpaceship() and if() and missilePinkCollision()

create pink spaceship with method, use if statement to set events, and call collision method

missileCollision()

create collision explosion when missile and bomb collide

baseCollision()

when bomb collides with base, make base disappear

groundCollision()

create flash with collision ground

gameProgress()

call method to stop game eventually

**createScoreBoard()**

setColor()

set color

setFont()

set font

drawString()

3 of these methods to label counts for score, missiles, and level

**createMessage()**

setColor()

set color

setFont

set font

drawstring()

draw a message that is a string

**greyBomb()**

variables: bombDX and bombDY define the delta X and delta Y between the missile initial coordinates and the mouse click coordinates

setColor()

set color of bomb to dark gray

fillOval()

set boundaries of bomb shape

increment bombX and bombY by adding bombDX and bombDY

if() and else if()

this if statement determines which random base to shoot from, depending on whether it exists or not

**missileCollision()**

setColor()

set explosion color at collision to red

fillOval()

set boundaries of the explosion shape

increment the score by 100 when the missile collides with a bomb

Random()

generates a random number to assign to a base

if() and else if()

statements help to determine which base is fired at if it exists

initialize bombX and bombY afterwards

**missileOrangeCollision()**

setColor()

sets the explosion to a yellow oval

fillOval()

fills the shape depending on set boundaries

increment the score

change initial xCoord

**missilePinkCollision()**

setColor()

sets the explosion to a yellow oval

fillOval()

fills the shape depending on set boundaries

increment the score

change initial xCoord

**baseCollision()**

if()

statements help to determine if collision at base occurs when the bomb coordinates are within the coordinates of the bombX width (within the if statements are also corresponding colors and boolean to show when the base

setColor()

within the if statements sets the color of the explosion when collision occurs

fillOval()

set boundaries of circle explosion shape when the collision occurs

**groundCollision()**

setColor()

set the ground collision color to orange

fillOval()

set boundaries of circle explosion shape when the collision occurs

**gameProgress()**

if()

if base1 lost all its missiles, declare base1 as non-operational

if()

if base2 lost all its missiles, declare base2 as non-operational

if()

if base2 lost all its missiles, declare base2 as non-operational

if()

if all the bases are gone, stop the animationThread of the game

if()

if the missile count is 0, stop the animationThread of the game

if()

if the score is between 700 and 2100 increment bombSpeed and missileCount to increase difficulty of the game

if()

if the score is greater than 2100, set missile count

**MListener class implements MouseListener**

mouseClicked()

creates a new missile to obtain the x and y coordinates

**------------------------------------------------------------------------------------------------------------------------------------------**

**OrangeSpaceShip Class**

**Purpose:** Create an orange spaceship object

**Variables:**

* xCoord = 0; yCoord = 300;
* orangeSpeed = 5;
* orangeWidth = 50; orangeHeight = 20;

**Methods:**

**orangeSpaceShip()**

setColor()

sets the color of the spaceship to orange

fillOval()

fills the boundaries of the shape

increment the x coordinate by the speed of the orange spaceship each time

greyBomb()

shoots bombs from the spaceship

baseCollision

base is destroyed when the bomb from the spaceship collides with it

**------------------------------------------------------------------------------------------------------------------------------------------**

**PinkSpaceship Class**

**Purpose:** Create a pink spaceship object

**Variables:**

* xCoord = 0; yCoord = 450;
* pinkW = 10; pinkH = 75;
* pinkSpeed = 10;

**Methods:**

**pinkSpaceship()**

setColor()

sets the color of the spaceship to pink

fillOval

fills the boundaries of the shape

increment the x coordinate by the speed of the pink spaceship each time

**Score Class**

**Purpose:** Converts the score of the game to a string with the player’s name and number of points

**Variables:**

* int points
* String name

**Methods:**

**Score()**

Integer.parseInt()

Coverts the points from a string to an integer

**Score2String()**

Creates output string with name and points

**------------------------------------------------------------------------------------------------------------------------------------------**

**HighScore Class**

**Purpose:**  Displays high score to the player and saves the score in permanent list for the game

**Variables:**

* String playerName;
* JFrame nameWindow
* JPanel panel
* JLabel instructions
* JTextField nameField
* JButton addButton
* ArrayList

**Methods:**

**createNameWindow()**

addActionListener

adds action listener to button

setTitle()

sets title for score window

setPreferredSize()

sets dimension of the window

setBackground()

sets color to cyan

setBounds()

sets boundaries of the window

setBorder()

sets border

setLayout()

sets a grid layout

setOpaque()

sets the window as opaque

add()

add the instructions, text fields, buttons, and panels to the frame

**getHighScores()**

add()

call score class to create new score

Scanner()

Scan a new file

While(has.next())

Continue if the array has another value

Score()

Create new score

Size()

Obtains the size of the array

Get()

Gets the value at a specific index to be used elsewhere

Remove()

Remove the intital comparison score

**actionPerformed()**

addScoreToLog()

call the method to add the score to the log

**addScoreToLog()**

new ArrayLIst<>()

creates a new array list for the scores

Scanner()

Scans a file

PrintWriter

Prints to a new file

Close()

Use to close the PrintWriter and the scanner

Delete()

Use to delete the first file path

File()

Creates a new file

**------------------------------------------------------------------------------------------------------------------------------------------**

**TESTING**

**Test #1**

**Description of Test:** Test missile to move slowly from initial missile point to mouse clicked point

**Input:**

bombSpeed = 10;

missile = new Missile(xCoord, yCoord, baseX);

public Missile(int x, int y, int baseX)

{

xCoord = x;

yCoord = y;

missileX=baseX;

missileY=625;

missileSpeed = .15;

}

public void magentaMissile(Graphics g){

double missileDX = (missileX-xCoord)\*(missileSpeed);

double missileDY = (missileY-yCoord)\*(missileSpeed);

g.setColor(Color.MAGENTA);

g.fillOval(missileX, missileY, BOMB\_SIZE, BOMB\_SIZE);

missileX+=missileDX;

missileY+=missileDY;}

**Expected Result:** a missile that moves slowly across the screen to where the mouse click event occured

**Actual Result:** a missile moved fast in the opposite direction of the mouse click

**Cause:** Change increments to -= not += for missileX and missleY

missileX+=missileDX;

missleY+=missileDY;

Change bomb speed to 0.005 instead of 10

**------------------------------------------------------------------------------------------------------------------------------------------**

**Test #2**

**Description of Test:** Test collision between the missile and the bomb to occur when the two Y coordinates are close enough to each other than is less than the explosion radius

**Input:**

// missile collision with bomb

int r = 45; // explosion radius

// if the distance between missile and bomb is less that the explosion radius...

if(Math.sqrt(Math.pow(missile.missileX-bombX,2)+Math.pow(missile.missileY-bombY,2)) <= r)

missileCollision(g, r);

**Expected Result:** collision occurs when the bomb and missile get very close to each other

**Actual Result:** collision with radius of 45 occurs when the bomb and missile are close to each other

**Cause:** Test was successful!

**------------------------------------------------------------------------------------------------------------------------------------------**

**Test #3**

**Description of Test:** Create a pink spaceship to move across the screen horizontally

**Input:**

public void pinkSpaceship(Graphics g){

int w = getWidth();

int h = getHeight();

int spaceship1W = 100;

int spaceship1H = 10;

int spaceship1Speed = 3;

// if the spaceship goes out of bounds, repaint it

if (lastX > w + spaceship1W) {

lastX = -spaceship1W;

}

g.setColor(Color.PINK);

g.fillRect(lastX, h/2 + spaceship1H, spaceship1W, spaceship1H);

lastX += spaceship1Speed;

}

**Expected Result:** pink spaceship moves horizontally across the screen at a constant speed

**Actual Result:** collision with radius of 45 occurs when the bomb and missile are close to each other

**Cause:** Test was successful!

**------------------------------------------------------------------------------------------------------------------------------------------**

**Test #4**

**Description of Test:** Create an orange spaceship to move diagonally across the screen

**Input:**

**public** **void** **orangeSpaceship**(**Graphics** g){

**int** **w2** = getWidth();

**int** **h2** = getHeight();

**int** **spaceship2W** = 50;

**int** **spaceship2H** = 20;

**int** **spaceship2Speed** = 2;

// if the spaceship goes out of bounds, repaint it

**if**(lastX > h2 + spaceship2H){

lastY = -spaceship1W;

}

g.setColor(**Color**.***ORANGE***);

g.fillOval(lastX, lastY, spaceship2W, spaceship2H);

lastY += spaceship2Speed;}

**Expected Result:** an orange spaceship that moves diagonally from the point (0, 0) on the screen

**Actual Result:** the orange spaceship does not correctly move across the screen as we intended

**Cause:** within the orangeSpaceship graphics class, the if statement should read like following to correctly create an orange spaceship that constantly moves across the screen diagonally

**if**(**lastY** > h2 + spaceship2H)

{lastY = -**spaceshipW**;}

g.setColor(**Color**.***ORANGE***);

g.fillOval(lastX, lastY, spaceship2W, spaceship2H);

lastY += spaceship2Speed;

**------------------------------------------------------------------------------------------------------------------------------------------**

**Relationship of Classes Map**

**JFrame**

**JComponent**

**Main Class**

main()

getHighScores()

**FlyingObjects Class**

FlyingObjects()

new Missile(xCoord, yCoord, baseX)

new OrangeSpaceShip()

new PinkSpaceship()

paintComponent

missile.magentaMissile()

orangeSS.orangeSpaceShip()

pinkSS.pinkSpaceship()

createScoreBoard()

createMessage()

greyBomb()

missileCollision()

missileOrangeCollision()

missilePinkCollision()

baseCollision()

groundCollision()

gameProgress()

MListener class implements MouseListener

**Missile Class**

**Missile()**

**magentaMissile(Graphics g)**

getXCoord()

getYCoord()

**PinkSpaceship Class**

**pinkSpaceship(Graphics g)**

setColor()

fillOval()

**OrangeSpaceShip Class**

**orangeSpaceShip(Graphics g)**

setColor()

fillOval()

greyBomb()

baseCollision()

FlyingObjects flying Objects

**HighScore Class**

createNameWindow()

getHighScores()

actionPerformed()

addScoreToLog()

**Score Class**

Score()

Score2String()