

FACHBEREICH ELEKTROTECHNIK UND INFORMATIK



Projektarbeit

C++_Swing_DrawGUI

Ahmed Ahmed

Betreuer : Prof. Dr. Carsten Link

Inhalt

1. Beschreibung des Projekts	6
1.1 Ziele Des Projekts	6
1.2 Vorteile des Projekts	6
1.3 Nachteile des Projekts	6
1.4 Wichtige Begriffe	6
1.4.1 Java-Swing	6
1.4.2 JavaSwing-Jframe	7
1.4.3 JavaSwing-Jpanel	7
1.4.4 RGB	7
1.4.5 Prozess	7
1.4.6 Parentprozess	8
1.4.7 Subprozess	8
1.4.8 Pixel	9
1.4.9 Input/Output-Stream	9
1.4.10 Buffer	10
1.4.11 ActionListener	10
1.4.12 MouseMotionListener	10
1.4.13 Thread	11
1.4.14 Thread- oder Prozesssynchronisation	11
1.4.15 Rendering	11
1.4.16 Race-Condition	12
2. Funktionsweise des Projekts	12
2.1 Die Erzeugung des C++-Subprozesses	12
2.2 Das Killing des C++-Subprozesses	13
2.3 Control-GUI	13

2.3.1 pixellist	14
2.3.2 Schutz der Pixelliste beim Race-Condition	14
2.3.3 Pixellisttmp	14
2.3.4 BufferedImage	15
2.3.5 Double-Buffering	15
2.3.6 Rendering	16
2.3.7 RGB in einem Integer zusammenfügen	17
2.3.8 public String getPixel(int x, int y)	18
2.3.9 Ausfüllung der Pixelliste mit der Hintergrundfarbe	19
2.3.10 Die Zeichnung der Pixel im Jpanel vom frontBuffer	20
2.3.11 Zooming der Pixel im Jpanel	20
2.3.11.2 Zooming gemäß der Mouse-Position	22
2.3.12 Löschung der Pixel im Jpanel	23
2.3.13 Kopieren der Pixel von der Pixelliste zur temporären Pixelliste	23
2.3.14 Kopieren der Pixel zurück von der temporären Pixelliste zur originalen Pixelliste	24
2.3.15 Austausch der Pixel zwischen die originale und die temporäre Pixellisten während der Ausführung des Programms in der Main-Klasse	24
2.4 Übertragung der Pixeldaten in Input/Output-Streams zwischen dem Subprozess und dem Java- Parentprozess	25
2.4.1 Die geschickten Streams von dem Subprozess und dem Parentprozess	26
2.4.2 char* rgbDecToHex(int Value)	27
2.4.3 char* xyDecToHex(int Value)	28
2.4.4 string hexAll(int x,int y,int red,int green,int blue)	31
2.4.5 string hexXY(int x,int y)	31

2.4.6 void sendhexAllToSwing(int x,int y,int red,int green,int blue)	32
2.4.7 void sendhexXYToSwing(int x,int y)	32
2.4.8 void setPixel(int x,int y,int red,int green,int blue)	33
2.4.9 std::string getPixel(int x,int y)	33
2.4.10 int getJPanelWidth()	34
2.4.11 int getJPanelHeight()	35
2.4.12 Die Übernahme und die Sendung der Input/Output-Streams beim Java-Parentprozess	35
2.5 Initialisierung des Jframes	41
2.5.1 JFrame.setSize(800,600)	42
2.5.2 JFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)	42
2.5.3 JFrame.setTitle("Drawing GUI")	42
2.5.4 JFrame.setLocation(70,70)	42
2.5.5 JFrame.setResizable(true)	43
2.5.6 JFrame.getContentPane().add(cg)	43
2.5.7 JFrame.setVisible(true)	43
2.5.8 JFrame.setJMenuBar(jmbar)	43
2.5.9 JFrame.addMouseMotionListener(jfactlis)	43
2.6 Erstellung des Toolbars	44
2.6.1 JMenuBar.add(Component)	45
2.6.2 Jmenu.add(Component)	45
2.6.3 JButton.setContentAreaFilled(false)	45
2.6.4 JComponent.setBorderPainted(false)	45
2.6.5 JComponent.setFocusable(false)	45
2.6.6 JComponent.addActionListener(jfactlis)	46
2.6.7 JComponent.setActionCommand(String)	46

2.7 JFrameListener	46
2.7.1 Übernahme der Mouse-Position	47
2.7.2 Der Tastendruck auf “+” oder “-” auf die Tastatur	47
2.7.3 Der Tastendruck auf “1” ,“2” oder “3” im Toolbar	49
2.7.4 Löschung der Pixel im Jpanel	50
2.7.5 Die Auswahl einer Datei	50
2.7.6 Das Killing des Subprozesses	51
3. Auswertung des Endbenutzers	52
3.1 Testing von setPixel(int x,int y,int red,int green,int blue) in DrawGUI.cpp	52
3.2 Testing von getPixel(int x,int y) in DrawGUI.cpp	56
3.3 Testing von int getJPanelWidth() und int getJPanelHeight() in DrawGUI.cpp	59
3.4 Testing das Zentrum-Zooming	61
3.5 Testing das Zooming gemäß der Mouse-Position	64
3.6 Testing die Fähigkeit der mehrfachen Zeichnung.....	65
3.7 Testing die Löschung der Pixel im Jpanel	66
3.8 Testing die Eingabe von falschen X,Y-Koordinaten im Jpanel.....	66
3.9 Testing das Killing des Subprozesses	67
3.10 Allgemeine Fragen für die Bewertung der Anwendung	69
4. Quellen.....	70

1. Beschreibung des Projekts

1.1 Ziele Des Projekts

Das Projekt (C++_Swing_DrawGUI) geht hauptsächlich um die Zeichnung der Pixel über den C++- Subprozess im Java-Swing-Parentprozess. Der Nutzer kann durch das Programm die gewünschten Koordinaten und RGB-Werte eingeben, außerdem die ausgewählten Pixel vom Swing lesen und zeigen lassen.

1.2 Vorteile des Projekts

Der Nutzer vermeidet die komplexe Behandlung von den C++-GrafikBibliotheken, was für die kleine und unkomplizierte Projekte einen Sinn macht. Die Behandlung vom JavaSwing-Programm ist einfach und flexibel, wo der Nutzer die kompilierten Dateien für die Ausführung direkt auswählen kann, außerdem kann der Nutzer die Zeichnung löschen und zoomen, sowie in verschiedenen SwingJframe-Größen zeichnen. Es kommt auch noch dazu das Töten des Subprozesses , Wenn der Subprozess gehängt werden könnte.

1.3 Nachteile des Projekts

Die Zeichnung der Pixel im Java-Swing ist relativ langsam wegen der Datenübertragung zwischen dem C++- Subprozess und dem JavaSwing-Programm.

1.4 Wichtige Begriffe

1.4.1 Java-Swing

Swing ist ein Bestandteil der Java Foundation Classes (JFC), mit denen grafische Benutzeroberflächen (GUIs) erstellt werden können. Die Swing-Klassen befinden sich in dem Java-Paket `javax.swing`. (<https://www.java-tutorial.org/swing.html>)

1.4.2 JavaSwing-Jframe

Die `javax.swing.JFrame`-Klasse ist ein Behälter, die `java.awt.Frame`-Klasse vererbt. Es kann im `JFrame` Labels, Buttons, Textfields eingefügt werden um GUI (graphical user interface) zu erstellen. (<https://www.javatpoint.com/java-jframe>)

1.4.3 JavaSwing-Jpanel

Die Klasse `javax.swing.JPanel` bildet den Standardcontainer für alle Bedienelemente. Er wird häufig zur Gruppierung zusammengehöriger Bedienelemente verwendet. In der Regel bildet ein `JPanel` die Grundlage für die Grafikprogrammierung. Je nach Verwendungszweck werden entweder weitere Container hinzugefügt oder, bei eher kleineren Anwendungen, die Bedienelemente direkt darauf platziert. Ein `JPanel` ist nicht auf eine bestimmte Anzahl an Elementen beschränkt, die es aufnehmen kann. (<https://www.java-tutorial.org/jpanel.html>)

1.4.4 RGB

Es ist eine Zusammenfassung für (Red-Green-Blue) (<https://praxistipps.chip.de/was-ist-rgb-einfach-und-verstaendlich-erklaert-44407>)

1.4.5 Prozess

Ein Prozess ist ein aktives Programm in der Ausführung und enthält ein Zähler, Prozess-Stack, Register und Programm-Code. (<https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process>)

1.4.6 Parentprozess

Alle die Prozesse im Betriebssystem außer der Startup-Prozess werden erzeugt, wenn ein Prozess führt das `fork()`-Systemsaufruf aus. Der Prozess, der `fork()`-Systemsaufruf verwendet, wird als Parentprozess bezeichnet. Ein Parentprozess kann mehrere Subprozesse haben, aber ein Subprozess gehört nur zu einem Parentprozess. (<https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process>)

1.4.7 Subprozess

Ein Subprozess wird im Betriebssystem durch ein Parentprozess mit `fork()`-Systemsaufruf erzeugt. Der Subprozess ist eine Kopie vom Parentprozess und vererbt mehrere Eigenschaften von dem wie User- und Prozess-Group-IDs, Environment Settings, Signal Handling Settings, Attached shared Memory Segments, Memory mapped Segments, Working Directory ...etc , aber der Subprozess hat ein eigenes Prozess-ID. Wenn ein Subprozess kein Parentprozess hätte, dann wird es direkt aus dem Kernel erzeugt. (<https://www.mksssoftware.com/docs/man3/fork.3.asp>) (<https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process>)

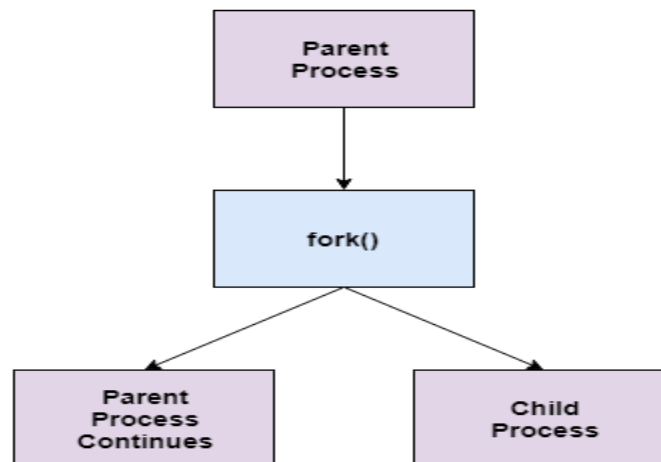


Bild 1. Ein Parentprozess Erzeugt ein Subprozess.

(<https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process>)

1.4.8 Pixel

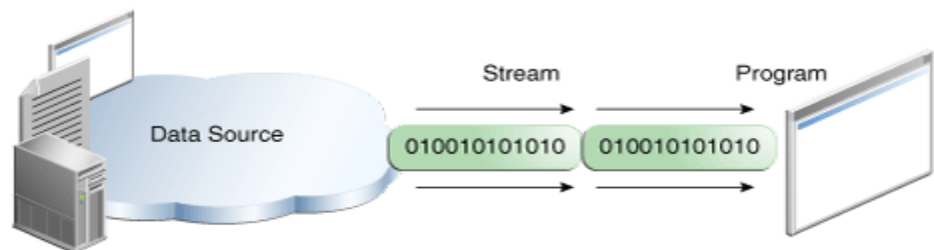
Als Pixel oder Bildpunkt bezeichnet man einzelne Farbwerte einer digitalen erzeugten Rastergrafik und jedes Pixel enthält ein RGB. (<https://www.it-business.de/was-sind-pixel-a-925513/>)

1.4.9 Input/Output-Stream

Ein I/O-Stream repräsentiert eine Eingabequelle oder Ausgabeziel und kann dadurch verschiedene Typen von den Quellen und Ziele repräsentiert werden, enthalten Disk-Files, Geräte, andere Programme und Memory-Arrays. Die Streams unterstützen unterschiedliche Datentypen wie Bytes, Primitive Datentypen und Objekte.

(<https://docs.oracle.com/javase/tutorial/essential/io/streams.html>)

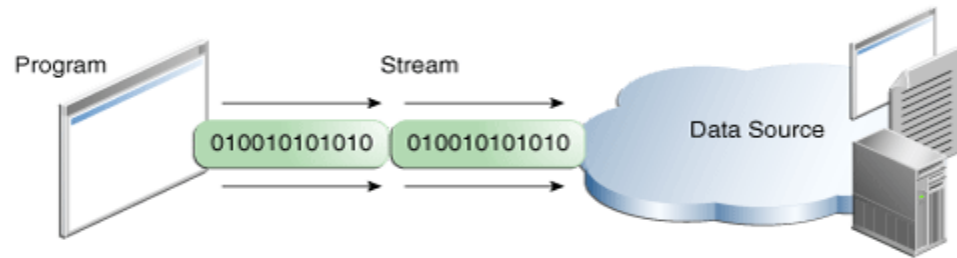
Ein Stream ist eine Reihenfolge von Daten, die von einem Programm gelesen werden. Ein Programm verwendet das Input-Stream um von den Datenquellen zu lesen.



Reading information into a program.

Bild 2. Ein Programm liest ein Input-Stream von den Datenquellen
(<https://docs.oracle.com/javase/tutorial/essential/io/streams.html>)

Ein Programm verwendet das Output-Stream um die Daten zum Ausgabeziel zu schreiben.



Writing information from a program.

Bild 3. Ein Programm schreibt ein Output-Stream zum Ausgabeziel.
(<https://docs.oracle.com/javase/tutorial/essential/io/streams.html>)

1.4.10 Buffer

Buffer ist in der Informatik und Telekommunikation ein Speicher für die Zwischenlagerung von Daten. (https://en.wikipedia.org/wiki/Data_buffer)

1.4.11 ActionListener

ActionListener in Java ist eine Klasse, die zuständig für die Bearbeitung aller Aktionenereignisse ist, wie z.B. ein Tastendruck. Eine Klasse, die diese Schnittstelle *implementiert*, muss eine `actionPerformed(ActionEvent e)` Methode enthalten. Der `ActionEvent`-Parameter ist ein Event-Objekt, das ein Ereignis (meistens einen Tastendruck) repräsentiert und enthält Informationen, die verwendet werden können um auf das Ereignis zu reagieren.

(<https://javapointers.com/java/java-se/actionlistener/>)

1.4.12 MouseMotionListener

MouseMotionListener in Java ist eine Klasse, die Mousebewegungsereignisse erhält und wird die zugehörige Methode im Listenerobjekt `mouseMoved(MouseEvent e)` aufgerufen. Wenn ein Mousebewegungsereignis auftritt, dann enthält `MouseEvent` die Informationen, die vom Mouse-Ereignis erstellt wurden.

(<https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseMotionListener.html>)

1.4.13 Thread

Ein Thread ist ein nebenläufiges Programm innerhalb eines Prozesses, das parallel durchgeführt wird.

1.4.14 Thread- oder Prozesssynchronisation

Die Thread- oder Prozesssynchronisation ist ein Mechanismus, der sicherstellt, dass zwei oder mehr nebenläufige Prozesse oder Threads nicht gleichzeitig ein bestimmtes Programmsegment ausführen.

[https://de.qaz.wiki/wiki/Synchronization_\(computer_science\)](https://de.qaz.wiki/wiki/Synchronization_(computer_science))

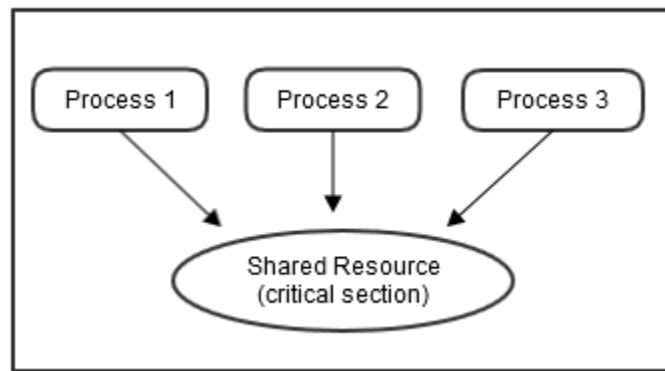


Bild 4. Mehrere Prozesse versuchen gleichzeitig geteilte Ressourcen zu zugreifen. [https://de.qaz.wiki/wiki/Synchronization_\(computer_science\)](https://de.qaz.wiki/wiki/Synchronization_(computer_science))

1.4.15 Rendering

Rendering ist die programmierten oder aufgenommenen Datensätze in zwei- oder dreidimensionale Bilder umgewandelt. In einem aufwendigen Rechenprozess werden auf die Rohdaten spezifische Eigenschaften übertragen. Am Ende entsteht ein auf Rechnern und anderen digitalen Geräten darstellbares Bildformat, das mit allen festgelegten Spezifikationen wiedergegeben wird.

[\(https://www.ionos.de/digitalguide/websites/webdesign/rendern/\)](https://www.ionos.de/digitalguide/websites/webdesign/rendern/)

1.4.16 Race-Condition

Race-Condition passiert, wenn es mehrere nebenläufige Prozesse oder Threads gibt und sie versuchen gleichzeitig die geteilten Ressourcen zu zugreifen um es zu schreiben oder zu aktualisieren.

2. Funktionsweise des Projekts

2.1 Die Erzeugung des C++-Subprozesses

Java-Parentprozess erzeugt ein Subprozess im Betriebssystem für C++-Programm, damit der Nutzer in dem die XY-Koordinaten und RGB-Werte eingeben kann, die im Java-Swing gezeichnet werden.

```
public Process childProcess;

public void createSubProcess(String path) throws IOException {

    ProcessBuilder processBuilder = new ProcessBuilder(path);

    processBuilder.redirectErrorStream(true); // ErrorStreams werden mit InputStreams zum Parentprozess geschickt
    childProcess = processBuilder.start(); // Erzeugung des Child Prozesses
}
```

Bild 5. Die Erzeugung des Subprozesses (SubProcess.java).
(<https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html>)

Durch ProcessBuilder kann Java ein Subprozess erzeugen und wird der Dateipfad als Parametet im ProcessBuilderklasse-Konstruktor eingegeben.

2.2 Das Killing des C++-Subprozesses

```
public void killSubProcess() throws InterruptedException { // Kindprozess eliminieren
    childProcess.destroy();

    new Thread(new Runnable() { // Kill das Subprozess im neuen Thread um die anderen Funktionen nicht zu blockieren
        @Override
        public void run() {
            try {
                TimeUnit.SECONDS.sleep(3); // Warten bis das Subprozess aufgeräumt wird
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            childProcess.destroyForcibly();

            JOptionPane.showMessageDialog(null, "The Subprocess has been killed...", "Message", 1);
        }
    }).start();
}
```

Bild 6. Das Killing des Subprozesses (SubProcess.java).

(<https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html>)

Manchmal ist es notwendig den Subprozess zu töten, wenn der Subprozess gehängt werden könnte oder wenn einen Fehler bei der Durchführung des Subprozesses auftreten würde.

Durch die Funktion `Process.destroy()` kann ein Subprozess getötet werden und das wird im eigenen Thread durchgeführt, damit andere Programmfunktionen während das Töten des Subprozesses nicht blockiert werden könnten.

Nach der Funktion `Process.destroy()` wird 3 Sekunden gewartet um sicherzustellen, dass die Bearbeitung des Subprozesses und alle die zugehörigen Funktionen schon aufgeräumt wurde. Dann kommt die Funktion `Process.destroyForcibly()` um das Töten des Subprozesses zu bestätigen, falls das Töten des Subprozesses durch die Funktion `Process.destroy()` scheitern würde.

Am Ende wird eine Nachricht zum Nutzer gezeigt, dass der Subprozess erfolgreich getötet wurde.

2.3 Control-GUI

ControlGUI-Klasse ist zuständig dafür ein Pixel im `JavaSwing-JPanel` zu erstellen, löschen, vergrößern oder zurückgeben.

2.3.1 pixellist

Es ist ein zwei dimensionales Integer-Array, das die RGB-Integers zu den entsprechenden X,Y-Koordinaten enthält.

2.3.2 Schutz der Pixelliste beim Race-Condition

Um das Race-Condition zu vermeiden soll der Zugriff der Threads oder der Prozesse auf die geteilten Ressourcen synchronisiert werden.

```
synchronized (lock) {  
    for (int x = 0; x < pixellist.length; x++) {  
        for (int y = 0; y < pixellist[x].length; y++) {  
            pixellist[x][y] = getIntRGB(defaultcol.getRed(), defaultcol.getGreen(), defaultcol.getBlue());  
        }  
    }  
}
```

Bild 7. Synchronisation der Pixelliste (ControlGUI.java).

Der Code im Synchronised-Block wird synchronisiert und kann nur gleichzeitig von einem Thread oder Prozess zugegriffen werden. Das Problem da stellt sich vor, wenn z.B zwei Threads gleichzeitig die Pixelliste zugreifen wollen und ein Thread aktualisiert die Pixelliste mit neuen Pixel, aber anderer Thread löscht die Pixel in der Pixelliste, was eine gemischte Pixel von den gelöschten und neuen Pixel erstellt werden könnte.

Deshalb ist es entscheidend, wenn nur ein Thread die Pixelliste verarbeiten kann und warten die andere Threads, bis der laufende Thread mit der Verarbeitung der Pixelliste fertig ist.

2.3.3 Pixellisttmp

Es ist eine Pixelliste für die temporäre Speicherung der Pixeldaten um die Pixeldaten später in der originalen Pixelliste zurückzuschreiben, damit die gezeichneten Pixel nicht verloren werden.

2.3.4 BufferedImage

Es ist eine Klasse, die die Image-Daten X,Y und RGB in einem zwei Dimensionalen Array enthält.

2.3.5 Double-Buffering

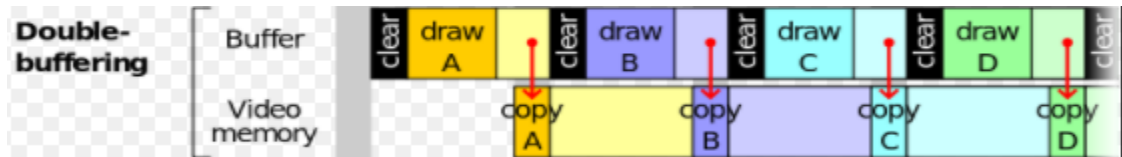


Bild 8. Double-Buffering. (https://en.wikipedia.org/wiki/Multiple_buffering)

Der Double-Buffering ist ein Mechanismus um das Schreiben und die Darstellung der Pixeldaten im BufferedImage separat voneinander zu halten.

Das Problem stellt sich da bei den nebenläufigen Schreiben und Darstellung der Pixeldaten vor, wenn der Pixelbuffer im BufferedImage während der Darstellung aktualisiert oder gelöscht werden könnte. Das könnte dazu führen, dass die dargestellten Pixeldaten eine Mischung von den alten und neuen Pixeldaten sein könnte.

Um dieses Problem zu vermeiden werden zwei BufferedImages backBuffer und frontBuffer initialisiert. Der backBuffer sorgt sich um das Schreiben bzw. Aktualisierung oder Löschung der Pixeldaten und der fronBuffer sorgt sich um die Darstellung der Pixeldaten.

```
public void fillPixellistZero(int width, int height) { // Pixelsfüllen mit der Hintergrundfarbe
    synchronized (lock) {
        for (int x = 0; x < pixellist.length; x++) {
            for (int y = 0; y < pixellist[x].length; y++) {
                pixellist[x][y] = getIntRGB(defaultcol.getRed(), defaultcol.getGreen(), defaultcol.getBlue());
            }
        }
    }
    backBuffer = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    frontBuffer = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
}
```

Bild 9. Instanzieren der Klasse BufferedImage (ControlGUI.java).

Bei der Ausfüllung der Pixelliste mit der Hintergrundfarbe werden zwei Objekte von der Klasse BufferedImage mit den Breite ,Höhe des Jpanels und RGB-Integertyp instanziiert. Die Breite entspricht die X- Koordinate und die Höhe entspricht Y-Koordinate.

```
if (frontBuffer == null) { // render was never called before
    frontBuffer = backBuffer;
} else { // else: front buffer holds old frame
    // swap front with back buffer
    BufferedImage temp = frontBuffer;
    frontBuffer = backBuffer;
    backBuffer = temp;
}
```

Bild 10. Austausch frontbuffer mit backBuffer (ControlGUI.java).

Nach dem Rendering werden frontBuffer und backBuffer ausgetauscht (Swapping), damit das alte Bild im frontBuffer mit dem neuen Bild im backBuffer ausgetauscht werden kann und bleibt der Inhalt von den beiden Buffers separat voneinander.

2.3.6 Rendering

```
public void render() { // Schreiben der Pixels in der Pixelliste
    synchronized (lock) {
        for (int x = 0; x < pixellist.length; x++) {
            for (int y = 0; y < pixellist[x].length; y++) {
                backBuffer.setRGB(x, y, pixellist[x][y]);
            }
        }
    }
}

BufferedImage temp = frontBuffer; // swap front with back buffer
frontBuffer = backBuffer;
backBuffer = temp;
}
```

Bild 11. Rendering der Pixel (ControlGUI.java).

Die Funktion ist zuständig die Pixeldaten X,Y und RGB im backBuffer zu schreiben und die Referenzen der front- und backBuffer auszutauschen, damit der frontBuffer die neuen geschriebenen Pixel im backBuffer darstellen kann.

2.3.7 RGB in einem Integer zusammenfügen

```
public int getIntRGB(int red, int green, int blue) { // Alle RGB-Werte in einem Integer zusammenfügen
    int val = 0;
    val |= red;
    val <<= 8; // 8 Bits nach links verschieben
    val |= green;
    val <<= 8; // 8 Bits nach links verschieben
    val |= blue;
    return val;
}
```

Bild 12. Zusammenfügen der RGB im Integer durch die Bitverschiebung (ControlGUI.java).

Die RGB sind verteilt auf drei Integer red, green und blue und durch die obige Funktion werden sie mittels der Bitverschiebung nach links nur in einem Integer zusammengefügt und zu den entsprechenden X,Y in der Pixelliste festgelegt. Das vorherige Verfahren hilft um die Pixeldaten einfacher zu behandeln und um Speicherplatz zu sparen.

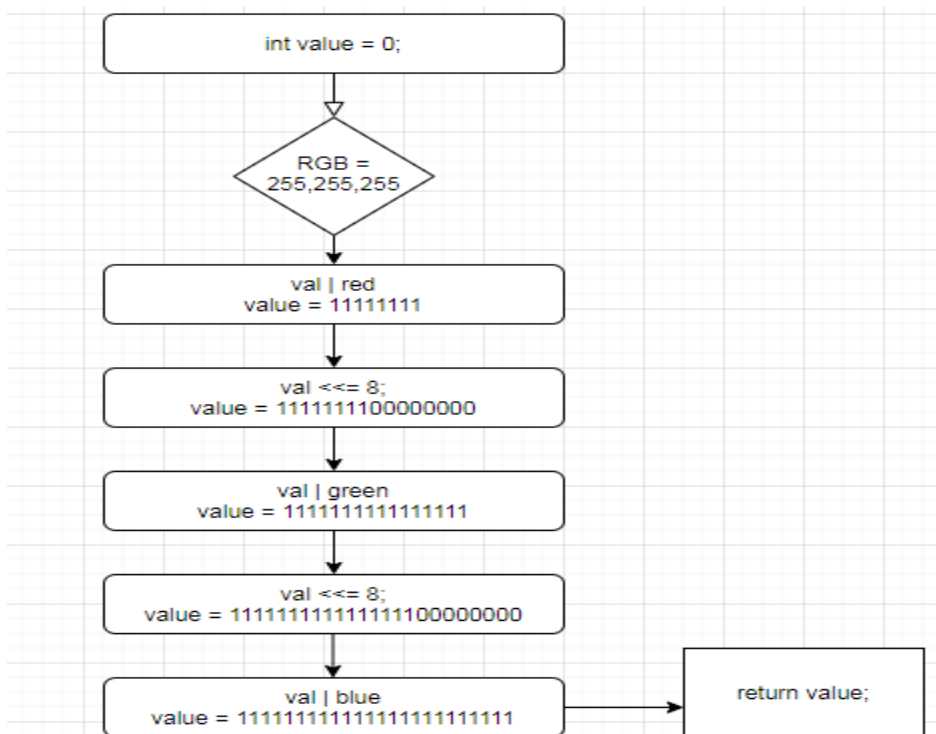


Bild 13. Das Algorithmusdiagramm für die Bitverschiebung nach links um mehrere Integer in einem Integer zusammenzufügen.

Der Dezimalwert der RGB ist jeweils maximal 255, was nur ein Byte entspricht, deshalb können die RGB-Integers zusammen in 3 Bytes durch die Bitverschiebung in einem einzelnen Integer eingefügt werden, dann wird dieser Integer enthalten RGB als Rückgabewert zurückgegeben.

2.3.8 public String getPixel(int x, int y)

```
public String getPixel(int x, int y) { // Pixelwert erhalten als String von X,Y Koordinaten

    String rgb;

    int rgbcopy = 0;

    int red = 255;
    int green = 255;
    int blue = 255;

    synchronized (lock) {

        rgbcopy = pixellist[x][y]; // erzeugung einer Kopie von einem Wert in der Pixelliste
    }

    blue &= rgbcopy;
    rgbcopy >>= 8; // 8 Bits nach links verschieben

    green &= rgbcopy;
    rgbcopy >>= 8; // 8 Bits nach links verschieben

    red &= rgbcopy;

    rgb = String.valueOf(red) + "," + String.valueOf(green) + "," + String.valueOf(blue); // Alle Farbestring zusammenfügen

    return rgb;
}
```

Bild 14. Ein RGB-String zu den eingegebenen X,Y zurückgeben (ControlGUI.java).

Der Java-Parentprozess erhält durch ein Input-Stream vom Subprozess die X,Y-Koordinaten um die entsprechenden Pixel-RGB im JPanel in einem Output-Stream zum Subprozess zurückzuschicken. Deswegen wird eine Kopie von der ursprünglichen Pixelliste erstellt um die originale Pixeldaten nicht manipulieren zu können.

Der Integer rgbcopy kopiert ein Feld von der Pixelliste mit dem gegebenen X,Y. Der zurückgegebene Integer von der Pixelliste beinhaltet die RGB zusammen, daher ist es notwendig die Bits des Integers zurück nach rechts zu schieben um die gewünschten RGB-Werten jeweils separat zurückzukehren. Am Ende werden alle RGB-Strings in einem String im folgend Format z.B "255,0,255" zusammengefügt und es als einen Rückgabewert für die obige Funktion zurückgegeben.

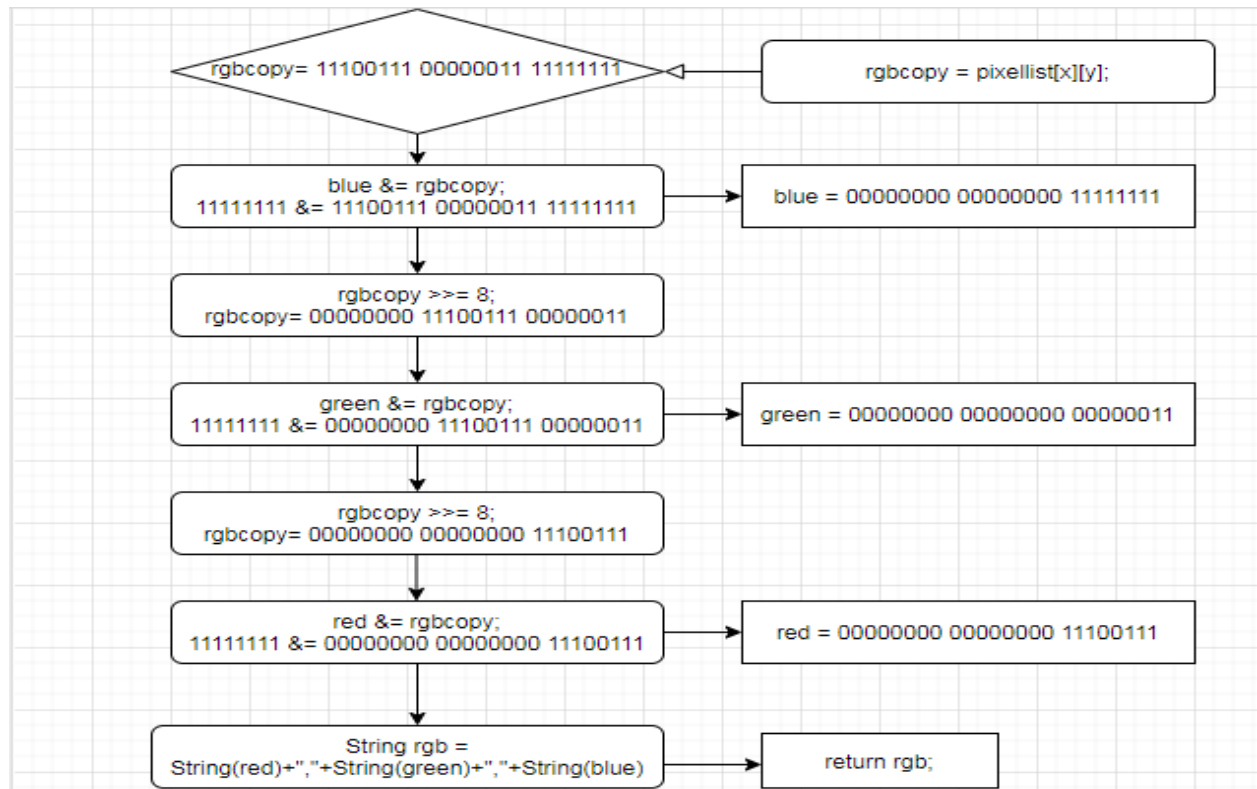


Bild 15. Das Algorithmusdiagramm für die Bitverschiebung nach rechts um die einzelnen entsprechenden RGB-Bytes für jede Farbe separat von einander rauszuholen und in einem String zusammenfügen.

2.3.9 Ausfüllung der Pixelliste mit der Hintergrundfarbe

```

public void fillPixellistZero(int width, int height) { // Pixelsfüllen mit der Hintergrundfarbe
    synchronized (lock) {
        for (int x = 0; x < pixellist.length; x++) {
            for (int y = 0; y < pixellist[x].length; y++) {
                pixellist[x][y] = getIntRGB(defaultcol.getRed(), defaultcol.getGreen(), defaultcol.getBlue());
            }
        }
    }

    backBuffer = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    frontBuffer = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
}

```

Bild 16. Ausfüllung der Pixelliste mit der Hintergrundfarbe (ControlGUI.java).

Die obige Funktion Initialisiert alle die Pixel in der Pixelliste mit der Hintergrundfarbe, dann werden die front- und backBuffer-BufferedImage mit den gegebenen Breite und Höhe des Jpanels instanziiert.

2.3.10 Die Zeichnung der Pixel im Jpanel vom frontBuffer

```
if(centerbool == true) {  
    if (centerzoomlevel == 1) {  
        g2d.drawImage(frontBuffer, 0, 0, frontBuffer.getWidth(), frontBuffer.getHeight(), null); // Bufferimage zeichnen  
        g2d.dispose(); // Alle Grafikobjekte werden freigeschaltet
```

Bild 17. Die Zeichnung des frontBuffers (ControlGUI.java).

Das Objekt g2d wurde von der Klasse Graphics2D erstellt. Anhand dieser Klasse können in zwei Dimensionen X und Y gezeichnet werden.

Die Funktion

g2d.drawImage(frontBuffer,0,0,frontBuffer.getWidth(),frontBuffer.getHeight(),null) zeichnet der frontBuffer-BufferedImage im Jpanel, dann über g2d.dispose() werden alle Grafikobjekte freigeschaltet.

2.3.11 Zooming der Pixel im Jpanel

Die gezeichneten Pixel im Jpanel können bis zum Faktor 3 vergrößert werden, aber muss trotz die Position der Pixel beim Zooming nicht verändert werden.

Das JavaSwing-Programm bietet den Nutzer zwei Vergrößerungsmöglichkeiten an, eine ist die Vergrößerung des Jpanelzentrums, aber die andere ist die Vergrößerung des JPanels gemäß der Mouse-Position im JFrame.

2.3.11.1 Zooming das Zentrum des Jpanels

```
} else if (centerzoomlevel == 2) {  
  
    zoomFactor = 2;  
    zoomWidth = width * zoomFactor;  
    zoomHeight = height * zoomFactor;  
    centerx = (width - zoomWidth) / 2; // Das Zentrum von x wird berechnet  
    centery = (height - zoomHeight) / 2; // Das Zentrum von y wird berechnet  
  
    AffineTransform at = new AffineTransform();  
    at.translate(centerx, centery); // Move the origin to the center of the JPanel  
    at.scale(zoomFactor, zoomFactor); // Um Faktor 2 vergrößern  
  
    g2d.setTransform(at);  
    g2d.drawImage(frontBuffer, 0, 0, frontBuffer.getWidth(), frontBuffer.getHeight(), null); // BufferedImage zeichnen  
  
    g2d.dispose(); // Alle Grafikobjekte werden freigeschaltet
```

Bild 18. Zooming das Zentrum des Jpanels um den Faktor 2 (ControlGUI.java).

zoomWidth= width * zoomFactor; ergibt sich die neue Breite des Jpanels nach dem Zooming, aber zoomHeight= height * zoomFactor; ergibt sich die neue Höhe des Jpanels nach dem Zooming.

Dann kommen centerx = (width - zoomWidth) / 2 und centery= (height - zoomHeight) / 2 um das X,Y-Zentrum im Jpanels zu berechnen.

at.translate(centerx,centery); sorgt sich dafür die originale X,Y-Position für die Transformation zu setzen, dann wird das Jpanel durch at.scale(zoomFactor,zoomFactor); mit dem vorgegebenen Zooming-Faktor neu skaliert. Nachher fügt die Funktion g2d.setTransform(at); die neue Transformation zum Jpanel ein und schließlich wird mittels der Funktion drawImage() der frontBuffer-BufferedImage im Jpanel gezeichnet.

2.3.11.2 Zooming gemäß der Mouse-Position

```
} else if(xyzoomlevel == 2) {  
  
    zoomFactor = 2;  
    zoom_x= mouseX * zoomFactor;  
    zoom_y = mouseY * zoomFactor;  
    x_position = (mouseX - zoom_x); // Die Position von Mouse-Xkoordinate wird berechnet  
    y_position = (mouseY - zoom_y); // Die Position von Mouse-Ykoordinate wird berechnet  
  
    AffineTransform at = new AffineTransform();  
    at.translate(x_position, y_position); // Move the origin to the center of the JPanel  
    at.scale(zoomFactor, zoomFactor);    // Um Faktor 2 vergrößern  
  
    g2d.setTransform(at);  
    g2d.drawImage(frontBuffer, 0, 0, frontBuffer.getWidth(), frontBuffer.getHeight(), null); // Bufferimage zeichnen  
  
    g2d.dispose(); // Alle Grafikobjekte werden freigeschaltet
```

Bild 19. Zooming des Jpanels um den Faktor 2 gemäß der Mouse-Position im JFrame (ControlGUL.java).

Im Zomming gemäß der Mouse-Position kann der Nutzer beim Drücken der Taste “+“ die Pixel im Jpanel zoomen und beim Drücken der Taste “-“ die Pixel im Jpanel zurück auszoomen.

zoom_x = mouseX * zoomFactor; ergibt sich die neue X-Position nach dem Zooming, aber zoom_y= mouseY * zoomFactor; ergibt sich die neue Y-Position nach dem Zooming.

Dann kommen x_position = (mouseX – zoom_x) und y_position = (mouseY – zoom_y) um die X,Y-Positionen vor dem Zooming zu berechnen.

at.translate(x_position, y_position); sorgt sich dafür die originale X,Y-Position für die Transformation zu setzen, dann wird das Jpanel durch at.scale(zoomFactor,zoomFactor); mit dem vorgegebenen Zooming-Faktor neu skaliert. Nachher fügt die Funktion g2d.setTransform(at); die neue Transformation zum Jpanel ein und schließlich wird mittels der Funktion drawImage() der frontBuffer-BufferedImage im Jpanel gezeichnet.

2.3.12 Löschung der Pixel im JPanel

```
public void clearGUI() {  
  
    bufferbool = true;  
    pixellist = new int[getWidth()][getHeight()];  
    fillPixellistZero(getWidth(), getHeight());  
    render();  
    repaint();  
  
}
```

Bild 20. Löschung die Pixel des Jpanels (ControlGUI.java).

Wenn der Nutzer die Pixel des jPanels löschen will, dann ruft er die obige Funktion auf. Erstens wird eine neue Pixelliste mit den aktuellen Breite und Höhe des Jpanels erstellt, zweitens wird die Pixelliste mit der Hintergrundfarbe initialisiert, danach werden über render() die Pixel in der Pixelliste im backBuffer geschrieben und die Referenzen der front- und backBuffer ausgetauscht, schließlich zeichnet repaint() die Pixel des frontBuffers im JPanel.

2.3.13 Kopieren der Pixel von der Pixelliste zur temporären Pixelliste

```
public void copyPixelsinPixellisttmp(int[][] pixellist, int[][] pixellisttmp) { // Copying Pixels tempora  
    synchronized (lock) {  
        if (pixellisttmp.length <= pixellist.length && pixellisttmp[0].length <= pixellist[0].length) {  
            for (int i = 0; i < pixellisttmp.length; i++) {  
                for (int i2 = 0; i2 < pixellisttmp[i].length; i2++) {  
                    pixellisttmp[i][i2] = pixellist[i][i2];  
                }  
            }  
        } else {  
            for (int i = 0; i < pixellist.length; i++) {  
                for (int i2 = 0; i2 < pixellist[i].length; i2++) {  
                    pixellisttmp[i][i2] = pixellist[i][i2];  
                }  
            }  
        }  
    }  
}
```

Bild 21. Kopieren die Pixel von der Pixelliste zur temporären Pixelliste (ControlGUI.java).

Die Obige Funktion ist zuständig darüber die Pixel von der originalen Pixelliste zur temporären Pixelliste zu kopieren. Falls die Breite und die Höhe der

originalen Pixelliste größer als die in der temporären Pixelliste sind, dann laufen die for-Schleifen über die originale Pixelliste durch und umgekehrt, wenn die Breite und die Höhe der temporären Pixelliste größer als die in der originalen Pixelliste sind, dann laufen die for-Schleifen über die temporäre Pixelliste durch. Das vorherige Verfahren wird verwendet um ein Index-out-of-Bounds-Exception zu vermeiden.

2.3.14 Kopieren der Pixel zurück von der temporären Pixelliste zur originalen Pixelliste

```
public void copyPixelsinPixellist(int[][] pixellist, int[][] pixellisttmp) { // Copying Pixels back from Pixellisttmp in Pixellist
    synchronized (lock) {
        for (int i = 0; i < pixellist.length; i++) {
            for (int i2 = 0; i2 < pixellist[i].length; i2++) {
                pixellist[i][i2] = pixellisttmp[i][i2];
            }
        }
    }
}
```

Bild 22. Kopieren die Pixel zurück von der temporären Pixelliste zur originalen Pixelliste (ControlGUI.java).

Die Pixel werden zurück in der originalen Pixelliste kopiert.

2.3.15 Austausch der Pixel zwischen die originale und die temporäre Pixellisten während der Ausführung des Programms in der Main-Klasse

```
public static void exec() throws IOException {
    cg.pixellisttmp = new int[cg.getWidth()][cg.getHeight()];
    cg.fillPixellisttmpZero(cg.getWidth(), cg.getHeight()); // Filling the Pixellisttmp Array with the Background Color
    cg.copyPixelsinPixellisttmp(cg.pixellist, cg.pixellisttmp); // Copying Pixels temporary from Pixellist in Pixellisttmp
    cg.pixellist = new int[cg.getWidth()][cg.getHeight()];
    cg.fillPixellistZero(cg.getWidth(), cg.getHeight()); // Filling the Pixellist Array with the Background Color
    cg.copyPixelsinPixellist(cg.pixellist, cg.pixellisttmp); // Copying Pixels back from Pixellisttmp in Pixellist
}
```

Bild 23. Austausch der Pixel zwischen die originale und die temporäre Pixellisten in der Ausführung des Programms in der Main-Klasse (Main.java).

Das Problem ergibt sich da, wenn die Größe des JPanels geändert wird, dann muss eine neue Pixelliste mit den neuen Breite und Höhe des Jpanels erstellt werden, deswegen damit der Nutzer mehrere Bilder nacheinander zeichnen kann, ist es wesentlich die vorherigen Pixel vorläufig in einer temporären Pixelliste zu speichern, dann später die Pixel in der originalen Pixelliste zurückzuschreiben.

Erstes wird ein Objekt für die temporäre Pixelliste mit den gegebenen Breite und Höhe des Jpanels instanziiert, zweitens wird die temporäre Pixelliste mit der Hintergrundfarbe ausgefüllt, drittens werden die Pixel von der originalen Pixelliste in der temporären Pixelliste kopiert, viertens wird ein neues Objekt für die originale Pixelliste mit den gegebenen Breite und Höhe des Jpanels instanziiert, fünftens wird die originale Pixelliste mit der Hintergrundfarbe ausgefüllt, schließlich werden die Pixel von der temporären Pixelliste zur originalen Pixelliste zurückgeschrieben.

2.4 Übertragung der Pixeldaten in Input/Output-Streams zwischen dem Subprozess und dem Java-Parentprozess

Die Pixeldaten X, Y und RGB werden vom Nutzer im Dezimalformat in der Funktion `setPixel(int x,int y,int red,int green,int blue)` im C++-Subprozess eingegeben, aber sollen nachher im Hexdezimalformat konvertiert und vom Java- Parentprozess als Input-Stream in einem String erhalten werden.

Der Nutzer kann auch nur die X,Y-Werte im Dezimalformat in der Funktion `std::string getPixel(int x,int y)` im C++-Subprozess eingeben, dann werden die X,Y-Werte auch im Hexdezimalformat konvertiert und vom Java- Parentprozess als Input-Stream in einem String erhalten, danach erhält den C++-Subprozess ein Output-Stream vom Java- Parentprozess in einem String mit den gewünschten RGB-Werten für das zugehörige Pixel im JavaSwing-JPanel mittels der gegebenen X,Y-Werten. Die Pixel-RGB wird als String-Rückgabewert für die Funktion `std::string getPixel(int x,int y)` zurückgegeben.

Die Größe der X, Y und RGB-Werte ist zusammen 6 Bytes, die im Hexdezimalformat vom C++-Subprozess zum Java- Parentprozess im Input-Stream übertragen werden sollen. Die einzelne Größe von X und Y jeweils beträgt 12 Bits und kann so damit auf

dem Koordinatensystem bis zur Zahl 4095 ($2^{12}-1$) dargestellt werden. Die einzelne Größe der RGB-Werte ist ein Byte, weil sie die maximale Zahl 255 haben, was bedeutet, dass nur ein Byte dafür schon reicht.

Es gibt auch noch die Möglichkeit für den Nutzer im C++-Subprozess die Breite und die Höhe des JPanels im Java-Swing in einem Output-Stream vom Java-Parentprozess zu erhalten.

2.4.1 Die geschickten Streams von dem Subprozess und dem Parentprozess

Der Subprozess schickt die folgenden Streams zum Parentprozess:

X,Y,RGB zusammen über `setPixel(int x,int y,int red,int green,int blue);`.

X,Y zusammen über `getPixel(int x,int y);` um ein Pixel-RGB zu den gegebenen X,Y abzuholen.

“W” über `getJPanelWidth()` um die Breite des Jpanels abzuholen.

“H” über `getJPanelHeight()` um die Höhe des Jpanels abzuholen.

Der Parentprozess schickt die folgenden Streams zum Subprozess:

Die RGB eines Pixels im folgenden Format z.B “255,0,255”.

Die Breite des Jpanels.

Die Höhe des Jpanels.

2.4.2 char* rgbDecToHex(int Value)

```
char* rgbDecToHex(int Value)
{
    char *CharRes = new (char);
    sprintf(CharRes,"%X", Value);

    char *CharRes2 = new (char);

    if(Value > -1 && Value < 16){

        *CharRes2 = *CharRes;
        *CharRes = 48;
        CharRes++;
        *CharRes = *CharRes2;
        CharRes++;
        *CharRes='\0';
        CharRes -= 2;

    }

    return CharRes;
}
```

Bild 24. Die Umwandlung der RGB-Werte vom Dezimalformat zum Hexadezimalformat (DrawGUI.cpp).

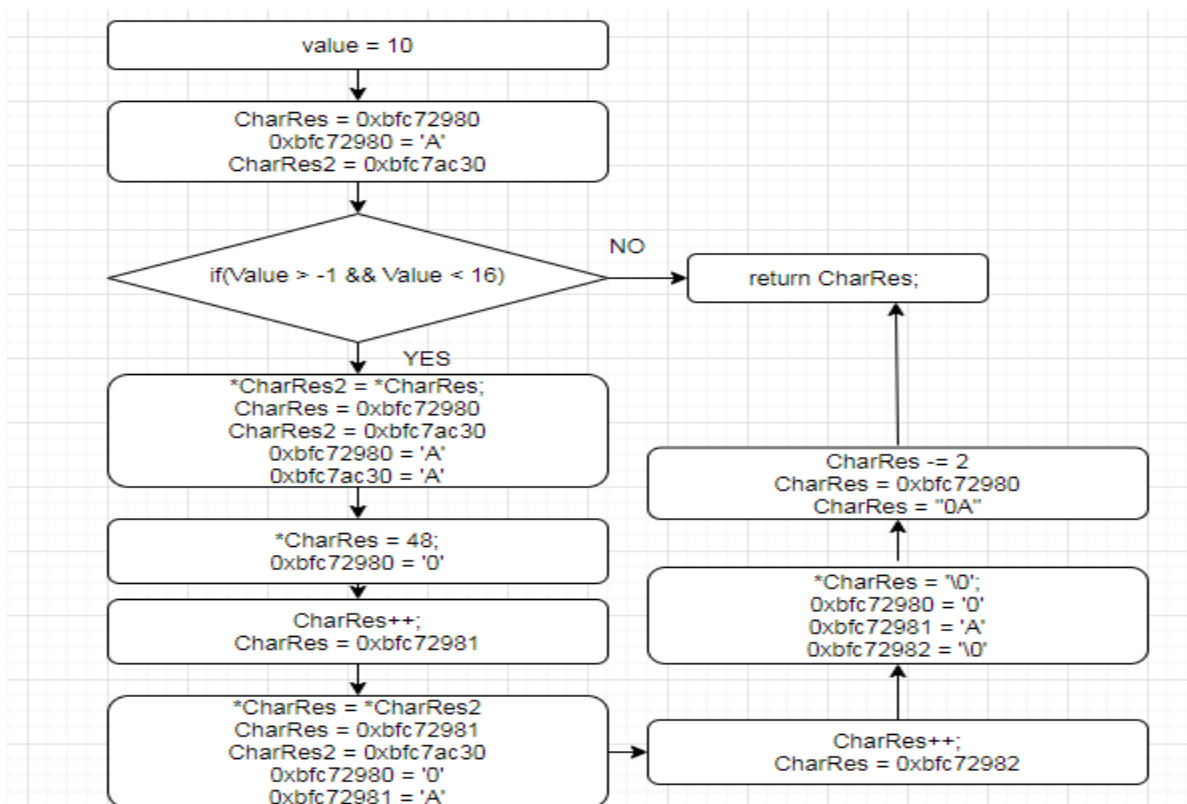


Bild 25. Das Algorithmusdiagramm für Umwandlung der RGB-Werte vom Dezimalformat zum Hexadezimalformat.

Der gegebene Int-Parameter(Dezimal) wird im Hexadezimalformat umgewandelt und als char Pointer zurückgegeben. Die Funktion `Sprintf(char* buffer,const char* format,...)` sorgt dafür ein Stringformat zum Char-Buffer zu schreiben. (<https://www.programiz.com/cpp-programming/library-function/cstdio/sprintf>)

In der Funktion `Sprintf(CharRes,"%X",Value)` wird der gegebene Int-Parameter zum Hexadezimal im Stringformat umgewandelt und dann im Char-Buffer gespeichert.

Die Größe des zurückgegebenen RGB-Stringwerts (CharRes) nach der Umwandlung zum Hexadezimal-String soll immer im Hexadezimalformat ein Byte entsprechen. Was bedeutet, dass der RGB-Stringwert (CharRes) immer zwei Hex-Chars haben muss z.B "FF".

Wenn der Wert des eingegebenen Integers im Parameter der Funktion weniger als 16 ist, dann hat CharRes nur ein Hex-Stringwert z.B 'A', aber soll in diesem Fall der Rest mit Nullen gefüllt werden z.B "0A".

Um dieses Problem zu lösen, wird einen anderen Char-Pointer CharRes2 zur temporären Speicherung des originalen Werts vom CharRes erstellt und wird im CharRes das erste char mit '0' (48 Dezimal) initialisiert, dann wird CharRes – Pointer um eins inkrementiert um seinen originalen Wert vom CharRes2 zurückzuholen. Nachher wird CharRes –Pointer nochmal um eins inkrementiert um ein Null-terminated-String am Ende des Strings zu setzen, dies deutet darauf hin, dass der String zu Ende ist. Schließlich wird CharRes –Pointer um zwei dekrementiert um vom Anfang des Strings zu beginnen.

2.4.3 char* xyDecToHex(int Value)

```
char* xyDecToHex(int Value) // Wandeln X, Y
{
    char *CharRes = new (char);
    sprintf(CharRes, "%X", Value);

    char *CharRes2 = new (char);

    if(Value > -1 && Value < 16){
        *CharRes2 = *CharRes;
        *CharRes = 48;
        CharRes++;
        *CharRes = 48;
        CharRes++;
        *CharRes = *CharRes2;
        CharRes++;
        *CharRes = '\0';
        CharRes -= 3;
    }
}
```

```

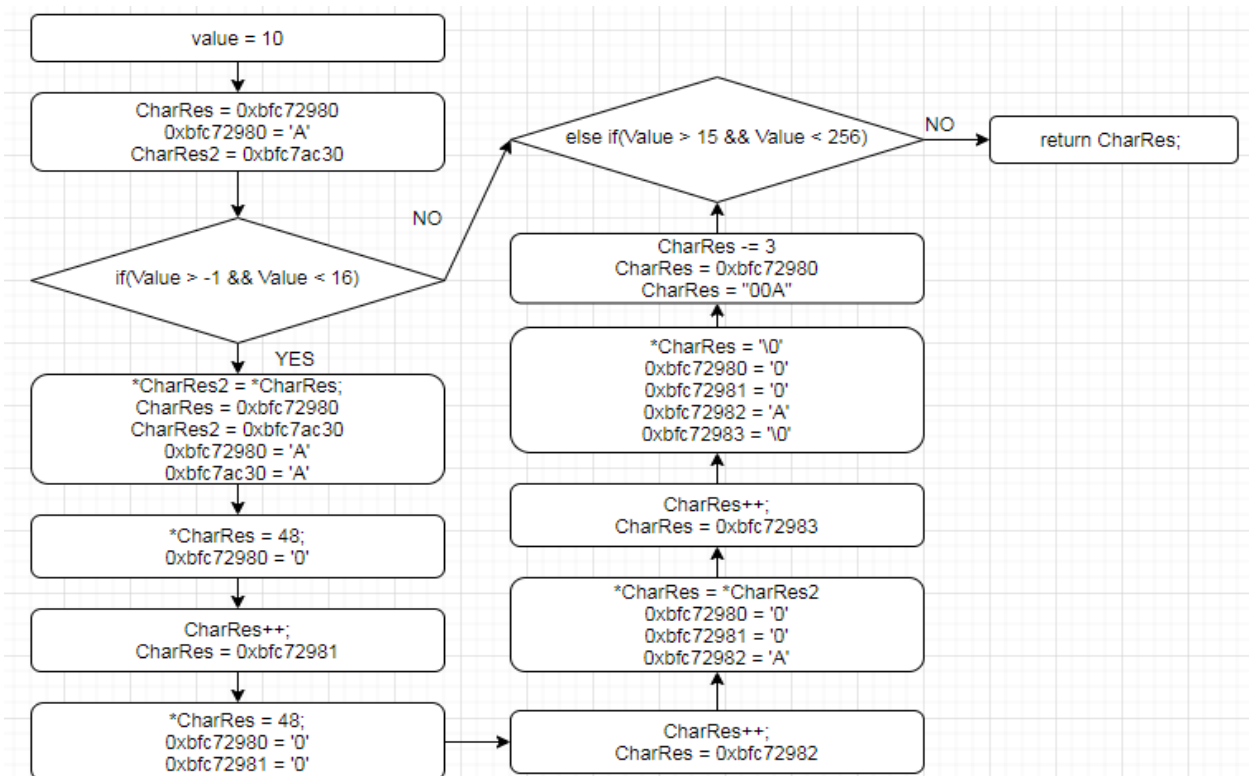
    }else if(Value > 15 && Value < 256){

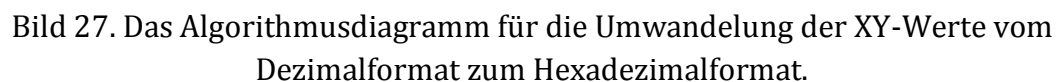
        *CharRes2 = *CharRes;
        CharRes++;
        CharRes2++;
        *CharRes2 = *CharRes;
        CharRes--;
        CharRes2--;
        *CharRes = 48;
        CharRes++;
        *CharRes = *CharRes2;
        CharRes++;
        CharRes2++;
        *CharRes = *CharRes2;
        CharRes++;
        *CharRes='\0';
        CharRes -= 3;

    }
    return CharRes;
}

```

Bild 26. Die Umwandlung der XY-Werte vom Dezimalformat zum Hexadezimalformat (DrawGUI.cpp).





Die Größe des zurückgegebenen X,Y-Stringwerts (CharRes) nach der Umwandlung zum Hexadezimal-String soll immer im Hexadezimalformat 12 Bits entsprechen. Was bedeutet, dass der X,Y-Stringwert (CharRes) immer drei Hex-Chars haben muss z.B. "FFF".

Wenn der Wert des eingegebenen Integers im Parameter der Funktion weniger als 16 ist, dann hat CharRes nur ein Hex-Char z.B 'A', und wenn der Wert des eingegebenen Integers zwischen 15 und 256 ist, dann hat CharRes nur zwei Hex-Chars z.B "C8", aber soll in den beiden Fällen der Rest mit Nullen gefüllt werden z.B "00A", "0C8".

Um dieses Problem zu lösen, wird einen anderen Char-Pointer CharRes2 zur temporären Speicherung des originalen Werts vom CharRes erstellt. Falls der gegebenen Integerwert weniger als 16 ist, werden im CharRes das erste und zweite chars mit '0' (48 Dezimal) initialisiert z.B "00A", dann wird CharRes – Pointer um eins inkrementiert um seinen originalen Wert vom CharRes2 zurückzuholen. Danach wird CharRes –Pointer nochmal um eins inkrementiert um Null-terminated-String am Ende des Strings zu setzen, dies deutet darauf hin, dass der String zu Ende ist. Schließlich wird CharRes –Pointer um drei dekrementiert um vom Anfang des Strings zu beginnen.

Falls der gegebenen Integerwert zwischen 15 und 256 ist, werden die selben Verfahren durchgeführt, aber wird in diesem Fall nur das erste Char im CharRes mit '0' (48 Dezimal) initialisiert z.B "0C8".

2.4.4 string hexAll(int x,int y,int red,int green,int blue)

```
string hexAll(int x,int y,int red,int green,int blue){ // X,Y,R,G,B-Werte zusammenfügen

    string hexall= "";

    string strx = xyDecToHex(x);
    string stry = xyDecToHex(y);
    string strred = rgbDecToHex(red);
    string strgreen = rgbDecToHex(green);
    string strblue = rgbDecToHex(blue);

    hexall = strx+stry+strred+strgreen+strblue;

    return hexall;
}
```

Bild 28. XY-RGB-Stringwerte zusammenfügen (DrawGUI.cpp).

Die Strings der Pixeldaten X, Y und RGB im Hexadezimalformat werden dadurch zusammengefügt und als einen einzigen String in einem Rückgabewert für die obige Funktion zurückgegeben.

2.4.5 string hexXY(int x,int y)

```
string hexXY(int x,int y){ // X,Y-Werte zusammenfügen

    string hexxy= "";

    string strx = xyDecToHex(x);
    string stry = xyDecToHex(y);

    hexxy = strx+stry;

    return hexxy;
}
```

Bild 29. XY-Stringwerte zusammenfügen (DrawGUI.cpp).

Die Strings der Pixeldaten X, Y im Hexadezimalformat werden dadurch zusammengefügt und als einen einzigen String in einem Rückgabewert für die obige Funktion zurückgegeben.

2.4.6 void sendhexAllToSwing(int x,int y,int red,int green,int blue)

```
void sendhexAllToSwing(int x,int y,int red,int green,int blue) {  
    cout<<hexAll(x,y,red,green,blue)<<"\n";  
    cout.flush();  
}
```

Bild 30. Setzen des XY-RGB-Stringwerts im Output-Stream (DrawGUI.cpp).

Der String der Pixeldaten X, Y und RGB im Hexadezimalformat wird über `cout<<hexXY(x,y,red,green,blue)<<"\n";` in einer Zeile in einem Input-Stream vom C++-Subprozess zum Java-ParentProzess geschickt.

Durch `Cout.flush();` wird der Buffer des Output-Streams vom C++-Subprozess in jeder Operation geleert um sicherzustellen, dass das Output-Stream nicht ganz voll sein könnte.

2.4.7 void sendhexXYToSwing(int x,int y)

```
void sendhexXYToSwing(int x,int y) {  
    cout<<hexXY(x,y)<<"\n";  
    cout.flush();  
}
```

Bild 31. Setzen des XY-Stringwerts im Output-Stream (DrawGUI.cpp).

Der String der Pixeldaten X,Y im Hexadezimalformat wird über `cout<<hexAll(x,y)<<"\n";` in einer Zeile in einem Input-Stream vom C++-Subprozess zum Java-ParentProzess geschickt.

Durch `Cout.flush();` wird der Buffer des Output-Streams vom C++-Subprozess in jeder Operation geleert.

2.4.8 void setPixel(int x,int y,int red,int green,int blue)

```
void setPixel(int x,int y,int red,int green,int blue){  
    sendhexAllToSwing(x,y,red,green,blue);  
}
```

Bild 32. Erstellung eines Pixels mit den XY-RGBwerten (DrawGUI.cpp).

Durch die Funktion `setPixel(int x,int y,int red,int green,int blue)` soll ein Pixel mit den Pixeldaten-XY,RGB im `JavaSwing-Jpanel` gezeichnet werden.

2.4.9 std::string getPixel(int x,int y)

```
std::string getPixel(int x,int y){  
    sendhexXYToSwing(x,y);  
    string rgb;  
    cin>>rgb;  
    cout<<rgb<<"\n";  
    return rgb;  
}
```

Bild 33. Die RGB-Werte eines Pixels über X,Y erhalten (DrawGUI.cpp).

Durch die Funktion `std::string getPixel(int x,int y)` wird ein String in einer Zeile mit den Pixeldaten-XY im Hexadezimalformat in einem Input-Stream vom C++-Subprozess zum Java-Parentprozess geschickt, dann antwortet den Java-Parentprozess in einem Output-Stream mit den zugehörigen RGB-Werten im folgenden Format z.B "255,0,255". Danach wird über `cout<<rgb<<"\n";` ein String in einer Zeile mit den RGB-Werten im folgenden Format z.B "255,0,255" in einem Input-Stream zum Java-Parentprozess wieder zurückgeschickt. Schließlich wird der RGB-String als Funktionsrückgabewert zurückgegeben.

2.4.10 int getJPanelWidth()

```
int getJPanelWidth() {  
    int width = 0;  
    cout<<"W"<<"\n";  
    string line;  
    cin>>line;  
    width = stoi(line);  
    return width;  
}
```

Bild 34. Die Breite des JavaSwing-JPanels erhalten (DrawGUI.cpp).

Der String "W" wird in einem Input-Stream in einer Zeile zum Java-Parentprozess geschickt, dann erhält die Funktion in einem Output-Stream vom Java-Parentprozess ein String in einer Zeile mit der Breite des JavaSwing-JPanels zurück.

`width = stoi(line)` ist zuständig darüber der Breite-String des JavaSwing-JPanels zum Integer `width` zu konvertieren, dann wird der Integer `width` als die Funktionsrückgabewert gesetzt.

2.4.11 int getJPanelHeight()

```
int getJPanelHeight () {  
    int height = 0;  
    cout<<"H"<<"\n";  
    string line;  
    cin>>line;  
    height = stoi(line);  
    return height;  
}
```

Bild 35. Die Höhe des JavaSwing-JPanels erhalten (DrawGUI.cpp).

Der String "H" wird in einem Input-Stream in einer Zeile zum Java-Parentprozess geschickt, dann erhält die Funktion in einem Output-Stream vom Java-Parentprozess ein String in einer Zeile mit der Breite des JavaSwing-JPanels zurück.

height = stoi(line) ist zuständig darüber der Höhe-String des JavaSwing-Jpanels zum Integer height zu konvertieren, dann wird der Integer height als die Funktionsrückgabewert gesetzt.

2.4.12 Die Übernahme und die Sendung der Input/Output-Streams beim Java-Parentprozess

```
,  
  
public void setInputStream(InputStream inputStream) {  
    this.br = new BufferedReader(new InputStreamReader(inputStream));  
}  
  
public BufferedReader getInputStream() {  
    return br;  
}  
  
public void setOutputStream(OutputStream outputStream) {  
    this.bw = new BufferedWriter(new OutputStreamWriter(outputStream));  
}  
  
public BufferedWriter getOutputStream() {  
    return bw;  
}
```

Bild 36. Set- und get Input/Outputstream (InputOutputStream.java).

setInputStream(InputStream inputStream) instantiiert ein Objekt von der Klasse BufferedReader um Input-Streams lesen zu können.

setOutputStream(OutputStream outputStream) instantiiert ein Objekt von der Klasse BufferedWriter um Output-Streams schreiben zu können.

```
public void readChildProcessStream(String path) throws IOException, InterruptedException {
    sp.createSubProcess(path);

    inout.setInputStream(sp.childProcess.getInputStream()); // lesen die Inputstreams des Child-Prozesses
    //
    // while (sp.childProcess.isAlive()) {
    //     System.out.println("aktiv");
    // }
    String line;
    while ((line = inout.getInputStream().readLine()) != null) {
        try {
            inout.controlInputStream(line, initial.frame);
        }
        catch (XYCoordinatesException e) {
            // X,y-Koordinaten werden überschritten
            SwingUtilities.invokeLater(new Runnable() { // Warten bis Swing fertig ist
```

Bild 37. Das Lesen des Input-Streams (readChildProcess.java).

Nach der Erzeugung eines neuen Subprozesses über die Funktion `sp.createSubProcess(path);` wird das Input-Stream vom erzeugten Subprozess als Parameter in der Funktion `inout.setInputStream(sp.childProcess.getInputStream());` gesetzt um das Input-Stream vom Subprozess zum Java-Parentprozess lesen zu können. Danach läuft eine endlose Schleife innerhalb des erhaltenen Input-Streams durch, solange es keinen Nullwert besitzt.

Die Funktion `inout.getInputStream().readLine()` liest das erhaltene Stream in einer Zeile für Zeile und gibt ein String damit zurück, dann wird der String Line mit der zurückgegebenen String-Zeile initialisiert und als Parameter in der Funktion `inout.controlInputStream(line,initial.frame);` gegeben.

```

public void controlInputStream(String value, JFrame jf) throws IOException, XYCoordinatesException {
    setOutputStream(sp.childProcess.getOutputStream()); // Schicken das Stream zum Kindprozess
    if (value.length() == 12) { // Stringvalue enthält X,Y,R,G,B Werte in HEX
        int x = Integer.parseInt(value.substring(0, 3), 16); // x ist 12 Bits
        int y = Integer.parseInt(value.substring(3, 6), 16); // y ist 12 Bits
        int red = Integer.parseInt(value.substring(6, 8), 16); // red ist 8 Bits bis 255
        int green = Integer.parseInt(value.substring(8, 10), 16); // green ist 8 Bits bis 255
        int blue = Integer.parseInt(value.substring(10, 12), 16); // blue ist 8 Bits bis 255

        if (x > cg.getWidth() - 1 || y > cg.getHeight() - 1 || x < 0
            || y < 0) { /*
                        * Exception werfen, wenn x und/oder y Koordinaten überschritten werden
                        */
            throw new XYCoordinatesException();
        } else {
            cg.pixellist[x][y] = cg.getIntRGB(red, green, blue); // Alle RGB-Werte in einem Integer zusammenfügen
        }
    }
}

```

Bild 38. Festlegung eines Output-Streams und die Übernahme von der X,Y,RGB-Werte (InputStream.java).

Teil 1 : In der Funktion setOutputStream(sp.childProcess.getOutputStream); wird das Output-Stream zum Subprozess gesetzt um das Output-Stream vom Java-Parentprozess zum Subprozess schreiben zu können.

Teil 2: Der String value erhält eine Zeile vom Input-Stream vom Subprozess.

Teil 3 : Wenn die Länge des erhaltenen Stringwerts in der String-Variable value 12 ist, deutet dies darauf hin, dass der String value im Hexadezimalformat aus 12 Hex-Chars mit einer entsprechenden Größe von 48 Bits bzw. 6 Bytes besteht und enthält die X,Y,RGB-Pixeln um ein Pixel im Java-Swing zu zeichnen.

Durch die Funktion Integer.parseInt(String,16) kann das erhaltene Input-Stream vom Subprozess in einem Hexadezimalformat-String zu einem Integer konvertiert werden und dann die Pixeln-Integers X,Y und RGB werden dadurch initialisiert.

Die X,Y-Integers werden initialisiert und lesen jeweils 3 Hex-Chars vom String value , was 12 Bits entspricht, aber die RGB-Integers werden initialisiert und lesen im Gegensatz nur 2 Hex-Chars vom String value, was 8 Bits bzw ein Byte entspricht.

Teil 4: Wenn die erhaltenen X- oder Y-Integers vom Subprozess weniger als null , X größer als die Breite des Java-JPanels oder Y größer als die Höhe des Java-JPanels ist, wird dann das Exception XYCoordinatesException geworfen um das Exception später beim Aufruf der obigen Funktion behandelt werden zu können.

Teil 5:

```
while ((line = inout.getInputStream().readLine()) != null) {
    try {
        inout.controlInputStream(line, initial.frame);
    }
    catch (XYCoordinatesException e) {
        // X,y-Koordinaten werden überschritten
        SwingUtilities.invokeLater(new Runnable() { // Warten bis Swing fertig ist
            @Override
            public void run() {
                // TODO Auto-generated method stub
                JOptionPane.showMessageDialog(null, "Mögliche X oder Y Koordinate wurde die Größe ihres JPanels überschri
                "Warning", 2);
            }
        });
        sp.killSubProcess(); // Kindprozess eliminieren
        cg.fillPixelListZero(cg.getWidth(), cg.getHeight()); // Filling the Pixellist Array with the Background Color
        return;
    }
}
```

Bild 39. Die Behandlung vom XYCoordinatesException (readChildProcess.java).

Durch `SwingUtilities.invokeLater(new Runnable())` wird die Behandlung vom `XYCoordinatesException` in einer Warteschlange im eigenen Thread festgelegt und wartet, bis alle Swing-Ereignisse verarbeitet werden, damit die laufenden Swing-Ereignissen nicht unterbrochen werden könnten. Nachher wird der erzeugte Subprozess über die Funktion `sp.killSubProcess()` getötet, weil es nicht mehr für die Verarbeitung gültig ist, danach werden alle Pixel im JPanel mit der Hintergrundfarbe über die Funktion `cg.fillPixelListZero(cg.getWidth(), cg.getHeight())` gefüllt um die gezeichneten Pixel wieder zu löschen.

Teil 6: Nach der Überprüfung die Länge des Strings `value` und der Richtigkeit der `X,Y-Integers` wird schließlich ein zwei dimensionaler Array-List über die Funktion `cg.pixellist[x][y] = getIntRGB(red, green, blue)` mit den gegebenen `X,Y-Integers` und den zugehörigen RGB-Pixeln initialisiert.

```

} else if (value.length() == 6) { // Stringvalue enthält nur X,Y-Werte in HEX
    int x = Integer.parseInt(value.substring(0, 3), 16);
    int y = Integer.parseInt(value.substring(3, 6), 16);

    if (x > cg.getWidth() - 1 || y > cg.getHeight() - 1) {
        throw new XYCoordinatesException();
    } else {
        getOutputStream().write(cg.getPixel(x, y) + "\n");
        getOutputStream().flush();
    }
} else if (value.indexOf(',') != -1) { // Stringvalue enthält RGB-Werte R,G,B in Decimal
    System.out.println(value);
} else if (value.equals("H")) { // Stringvalue enthält Height-Wert in Decimal
    getOutputStream().write(cg.getHeight() + "\n");
    getOutputStream().flush();
} else if (value.equals("W")) { // Stringvalue enthält Width-Wert in Decimal
    getOutputStream().write(cg.getWidth() + "\n");
    getOutputStream().flush();
}
}

```

Bild 40. Übernahme verschiedener Input-Streams vom Subprozess und die Überprüfung des Stringwertes value (InputStream.java).

Teil 1: Wenn die Länge des Strings value 6 ist, deutet dies hin, dass das erhaltene Input-Stream vom Subprozess nur die X,Y-Werte enthält um die entsprechenden RGB-Pixelaten in einem Output-Stream zum Subprozess zurückzuschicken.

In diesem falls werden nur die X,Y-Integers initialisiert und sie lesen wie vorher auch 3 Hex-Chars vom String value.

Bei der falschen Eingabe von X,Y-Integers im Subprozess wird auch das XYCoordinatesException geworfen und später es beim Aufruf der obigen Funktion behandelt. Ansonsten schickt Java-Parentprozess ein Output-Stream zum C++-Subprozess mit den entsprechenden RGB-Pixelaten in einem Stringzeile durch die obige Funktion `getOutputStream().write(cg.getPixel(x,y)+'\n');` zurück.

Nachher wird durch die Funktion `getOutputStream().flush();` der Buffer des Output-Streams geleert um sicherzustellen, dass das Output-Stream nicht ganz voll sein könnte.

Teil 2: Die Funktion `value.indexOf(',') != -1` überprüft, ob das String `value` den Char `' '` enthält, wenn das so ist, bedeutet das, dass das erhaltene String-Input-Stream vom Subprozess die RGB-Werte besitzt, weil sie im folgenden Format `"r,g,b"` durch die Funktion `cg.getPixel(x,y)` dargestellt werden sollten.

Teil 3: Wenn String `value` den Char `"H"` oder `"W"` enthält, dann fragt der Subprozess um die Breite bzw. die Höhe des JPanels nach.

Java-Parentprozess schickt daher die Breite oder die Höhe des Java-JPanels in einem Output-Stream zum Subprozess zurück, danach wird das Output-Stream durch `flush()` wieder geleert.

```
Thread t = new Thread(new Runnable() { // Lesen des Kindprozesses im neuen Thread

    @Override
    public void run() {

        try {

            rchproc.readChildProcessStream(jfactlis.getPath());

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Ungültige Datei wurde ausgewählt...", "Error", 0);

        } catch (NumberFormatException e) {

            JOptionPane.showMessageDialog(null, "Ungültige Datei wurde ausgewählt...", "Error", 0);

        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

});

t.start(); // Start des Threads
t.interrupt();
/* Interrupt-Flag wird gesetzt*/

}
```

Bild 41. Die Sendung und die Übernahme der Input/Output-Streams geht im eigenen Thread (Main.java).

Teil 1: Es wird einen eigenen Thread für die Sendung und die Übernahme der Input/Output-Streams erstellt, damit andere Programmfunktionen nicht blockiert werden, falls die Übertragung der Streams zwischen dem Subprozess und dem Parentprozess blockiert werden könnte.

Teil 2: IOException oder NumberFormatException werden geworfen, wenn eine ungültige Datei ausgewählt wird und werden dementsprechend falsche Input-Streams vom Subprozess gelesen.

Teil 3: t.interrupt(); setzt Interrupt-Flag für den erzeugten Thread um sicherzustellen, dass die Ausführung des Threads ohne eine Block-Ursache erfolgreich beendet wird. Falls der Thread blockiert werden könnte, wird ein InterruptedException geworfen.

2.5 Initialisierung des Jframes

```
public class JFrameInitialisation {  
  
    private JMenuBar jmbar;  
    public JFrame frame;  
    private JFrameListener jfactlis;  
  
    public JFrameInitialisation(ControlGui cg, JFrameListener jfactlis) {  
  
        this.jfactlis = jfactlis;  
        frame = new JFrame();  
        frame.setSize(800, 600);  
        createToolbar();  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setTitle("Drawing GUI");  
        frame.setLocation(70, 70);  
        frame.setResizable(true);  
        frame.getContentPane().add(cg);  
        frame.setVisible(true);  
        frame.setJMenuBar(jmbar);  
        frame.addMouseMotionListener(jfactlis);  
  
    }  
}
```

Bild 42. Initialisierung des Jframes im Konstruktor (JFrameInitialization.java).

(<https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>)

2.5.1 JFrame.setSize(800,600)

Die Größe des Jframes wird mit der Breite 800 und die Höhe 600 erstellt.

2.5.2 JFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)



Bild 43. X-Zeichen drücken (Java-Swing).

Die Funktion sorgt dafür, wenn X-Zeichen im JFrame gedrückt wird, wird dann auch das Java programm dadurch beendet.

2.5.3 JFrame.setTitle("Drawing GUI")



Bild 44. Der Titel des Jframes (Java-Swing).

Die Funktion erstellt ein Titel für das JFrame.

2.5.4 JFrame.setLocation(70,70)

Die Funktion setzt die Position des Jframes auf dem Bildschirm mit der gegebenen X,Y-Position.

2.5.5 JFrame.setResizable(true)

Wenn der gegebene Parameter true ist, dann kann der Nutzer die Größe des Jframes nach der Auslösung verändern.

2.5.6 JFrame.getContentPane().add(cg)

Es wird dadurch ein Jpanel zum JFrame eingefügt.

```
@SuppressWarnings("serial")  
public class ControlGui extends JPanel {
```

Bild 45. Die Klasse ControlGui vererbt die Klasse Jpanel (ControlGUI.java).

cg ist ein Objekt von der Klasse ControlGui, die von der Klasse Jpanel vererbt.

2.5.7 JFrame.setVisible(true)

Das lässt das Fenster des Jframes sichtbar sein.

2.5.8 JFrame.setJMenuBar(jmbar)

Es wird ein JMenuBar zum JFrame eingefügt.

2.5.9 JFrame.addMouseListener(jfactlis)

Der JFrame kann dadurch eine MouseMotionlistener -Klasse einfügen. jfactlis ist ein Objekt von der Klasse JFrameListener, die den Interface MouseMotionlistener implementiert.

2.6 Erstellung des Toolbars

```
protected void createToolbar() {  
  
    JMenuItem one, two, three;  
    JMenu zoomjmenu;  
    JButton clearbutton;  
    JButton choosebutton;  
    JButton killprocessbutton;  
  
    jmbar = new JMenuBar();  
  
    zoomjmenu = new JMenu("Zoom Level");  
  
    clearbutton = new JButton("Clear");  
    clearbutton.setContentAreaFilled(false);  
    clearbutton.setBorderPainted(false);  
    clearbutton.setFocusable(false);  
  
    choosebutton = new JButton("Choose compiled File");  
    choosebutton.setContentAreaFilled(false);  
    choosebutton.setBorderPainted(false);  
    choosebutton.setFocusable(false);  
  
    killprocessbutton = new JButton("Kill Subprocess");  
    killprocessbutton.setContentAreaFilled(false);  
    killprocessbutton.setBorderPainted(false);  
    killprocessbutton.setFocusable(false);  
  
    jmbar.add(zoomjmenu);  
    jmbar.add(clearbutton);  
    jmbar.add(choosebutton);  
    jmbar.add(killprocessbutton);  
  
    one = new JMenuItem("1");  
    one.addActionListener(jfactlis);  
    one.setActionCommand("one");  
    zoomjmenu.add(one);  
  
    two = new JMenuItem("2");  
    two.addActionListener(jfactlis);  
    two.setActionCommand("two");  
    zoomjmenu.add(two);  
  
    three = new JMenuItem("3");  
    three.addActionListener(jfactlis);  
    three.setActionCommand("three");  
    zoomjmenu.add(three);  
  
    clearbutton.addActionListener(jfactlis);  
    clearbutton.setActionCommand("clear");  
  
    choosebutton.addActionListener(jfactlis);  
    choosebutton.setActionCommand("choose");  
  
    killprocessbutton.addActionListener(jfactlis);  
    killprocessbutton.setActionCommand("killsubprocess");  
}
```



Bild 46. Erstellung des Toolbars (JFrameInitialization.java).

Durch die Funktion `createToolBar()` wird ein `ToolBar` bzw. ein `JMenuBar` erzeugt und dann im `JFrame` eingefügt. Der `ToolBar` enthält ein `JMenu` für die Zoomfunktion und drei `JButton` für das Löschen der Pixel, Auswahl der kompilierten Datei und das Töten des Subprozesses. Der `JMenu` enthält drei `JMenuItem`s für 3 verschiedene Zoomstufen, die vom Nutzer ausgewählt werden können.

2.6.1 JMenuBar.add(Component)

Eine Komponente wie `JButton`, `JMenu` oder `JLabel` kann zum `JMenuBar` eingefügt werden.

2.6.2 JMenu.add(Component)

Eine Komponente wie in diesem Fall `JMenuItem` kann zum `JMenuBar` eingefügt werden.

2.6.3 JButton.setContentAreaFilled(false)

Wenn der gegebene Parameter `true` ist, kann jedes Pixel innerhalb der Buttongrenzen gezeichnet werden, ansonst wird das `JButton` Transparent sein.

2.6.4 JComponent.setBorderPainted(false)

Wenn der gegebene Parameter `true` ist, können dann die Grenzen des Buttons gezeichnet werden.

2.6.5 JComponent.setFocusable(false)

`setFocusable` deutet darauf hin, ob es einen aktuellen Fokus für eine Komponente gibt. Mit dem gegebenen Parameter `false` gibt es keinen Fokus.

2.6.6 JComponent.addActionListener(jfactlis)

Es kann eine ausgewählte Komponente dadurch eine ActionListener-Klasse einfügen. jfactlis ist ein Objekt von der Klasse JFrameListener, die den Interface java.awt.event.ActionListener implementiert.

2.6.7 JComponent.setActionCommand(String)

Die obige Funktion setzt einen bestimmten Befehlsstringwert, der über ActionEvent.getActionCommand(); erhalten wird.

2.7 JFrameListener

```
public class JFrameListener implements java.awt.event.ActionListener, MouseMotionListener {
```

Bild 47. Implementierung der Interfaces ActionListener und MouseMotionListener.

Die Klasse JFrameListener implementiert die Interfaces ActionListener und MouseMotionListener und werden daher die folgenden Methoden aufgerufen und überschrieben.

```
@Override  
public void actionPerformed(ActionEvent e)  
@Override  
public void mouseDragged(MouseEvent e)  
@Override  
public void mouseMoved(MouseEvent e)
```

Der ActionEvent-Parameter ist ein Event-Objekt, das ein Ereignis (meistens einen Tastendruck) repräsentiert und enthält Informationen, die verwendet werden können um auf dieses Ereignis zu reagieren, aber der MouseEvent enthält die Informationen, die von den Mouse-Ereignissen erstellt wurden.

2.7.1 Übernahme der Mouse-Position

```
@Override
public void mouseMoved(MouseEvent e) {

    cg.mouseX = e.getX(); // X Koordinate des Jpanels ermitteln
    cg.mouseY = e.getY(); // Y Koordinate des Jpanels ermitteln

}
```

Bild 48. Die Übernahme der Mouseposition in den X,Y-Koordinaten (JframeListener.java).

Die Mouse-Position in den X,Y-Koordinaten wird über MouseEvent erhalten, damit es beim Zoomen und Auszoomen des Jpanels verwendet werden kann.

2.7.2 Der Tastendruck auf “+” oder “-” auf die Tastatur

```
public JFrameListener(ControlGUI cg, SubProcess sp) {

    this.cg = cg;
    this.sp = sp;

    cg.getInputMap().put(KeyStroke.getKeyStroke('+'), "Zoomin");
    cg.getActionMap().put("Zoomin", new AbstractAction() {

        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {

            if(cg.xyzoomlevel < cg.maxzoomlevel) {

                cg.xyzoomlevel++;
                cg.centerbool = false;
                cg.repaint();

            }

        }

    });

}
```

Bild 49. Warten das Drücken auf die “+” Taste auf die Tastatur.

```

cg.getInputMap().put(KeyStroke.getKeyStroke('-'), "Zoomout");
cg.getActionMap().put("Zoomout", new AbstractAction() {

    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {

        if(cg.xyzoomlevel > cg.minzoomlevel) {

            cg.xyzoomlevel--;
            cg.centerbool = false;
            cg.repaint();

        }

    }

});

```

Bild 50. Warten das Drücken auf die "-" Taste auf die Tastatur (JframeListener.java).

Über `cg.getInputMap().put(keyStroke.getKeyStroke('+'), "Zoomin");` wird die gewünschte Taste und der zugehörige Schlüssel dafür erstellt und über `cg.getActionMap().put()` wird den Schlüssel der Taste gegeben, auf dessen Ereigniss ActionListener warten soll.

Falls die Tasten "+" oder "-" gedrückt werden, wird der Integer `cg.xyzoomlevel` inkrementiert oder dekrementiert. Damit der Integer `cg.xyzoomlevel` den Zoom-Faktor 3 nicht überschreiten kann, wurde der Integer `cg.maxzoomlevel` mit dem Wert 3 initialisiert, solange der `cg.xyzoomlevel` kleiner als `cg.maxzoomlevel` ist, kann es inkrementiert werden. Im Gegensatz muss auch der `cg.xyzoomlevel` nicht kleiner als null sein, deswegen wurde der Integer `cg.minzoomlevel` mit dem Wert 1 initialisiert, solange der `cg.xyzoomlevel` größer als `cg.minzoomlevel`, kann es dekrementiert werden.

Der boolean `cg.centerbool` wird auf `false` initialisiert um sicher zu stellen, dass Zentrum-Zooming deaktiviert wird, sondern das Zooming gemäß der Mouse-Position. Schließlich wird das Jpanel durch `cg.repaint()` neu gezeichnet um die Pixel zu zoomen oder auszuzoomen.

2.7.3 Der Tastendruck auf “1”, “2” oder “3” im Toolbar

```
@Override
public void actionPerformed(ActionEvent e) { // Warten eine Aktion von dem Nutzer

    String command = e.getActionCommand();

    if (command.equals("one")) { // Zoomlevel auswählen

        cg.centerzoomlevel = 1;
        cg.xyzzoomlevel = 1;
        cg.centerbool = true;
        cg.repaint();

    } else if (command.equals("two")) { // Zoomlevel auswählen

        cg.centerzoomlevel = 2;
        cg.xyzzoomlevel = 2;
        cg.centerbool = true;
        cg.repaint();

    } else if (command.equals("three")) { // Zoomlevel auswählen

        cg.centerzoomlevel = 3;
        cg.xyzzoomlevel = 3;
        cg.centerbool = true;
        cg.repaint();

    }

}
```

Bild 51. Warten das Drücken auf die “1”, “2” oder “3” Tasten im Toolbar (JframeListener.java).

Der ActionEvent wartet auf ein Ereignis, dessen Wert als String von der Funktion e.getActionCommand(); zurückgegeben wird.

Falls die Tasten 1”, “2” oder “3” im Toolbar gedrückt werden, wird der Integer cg.centerzoomlevel mit dem Wert der ausgewählten Taste initialisiert. Der Integer cg.xyzzoomlevel wird auch hier mit dem ausgewählten Wert initialisiert um eine Kompatibilität zwischen die beiden Zoomingfunktionen (Zentrum-Zomming, Zooming gemäß der Mouse-Position) zu erstellen.

Der boolean cg.centerbool wird auf dem Wert true initialisiert um Zentrum-Zooming zu aktivieren, schließlich wird der Jpanel durch cg.repaint() neu gezeichnet um die Pixel zu zommen oder auszuzommen.

2.7.4 Löschung der Pixel im Jpanel

```
} else if (command.equals("clear")) { // Zeichnung löschen  
    cg.clearGUI();
```

Bild 52. Warten das Drücken auf die “Clear” Taste im Toollbar (JframeListener.java).

2.7.5 Die Auswahl einer Datei

```
} else if (command.equals("choose")) { // Eine kompilierte Datei auswählen  
    String currentDir = System.getProperty("user.dir"); // Working-Directory abholen  
    JFileChooser jfc = new JFileChooser(currentDir);  
    int returnValue = jfc.showOpenDialog(null);  
    if (returnValue == JFileChooser.APPROVE_OPTION) {  
        File selectedFile = jfc.getSelectedFile();  
        setPath(selectedFile.getAbsolutePath());  
        try {  
            if(childprocessalivebool == true) { // Falls einen laufend Subprozess gibt, wird es getötet  
                try {  
                    if(sp.childProcess.isAlive()) {  
                        sp.killSubProcess();  
                    }  
                } catch (InterruptedException e1) {  
                    // TODO Auto-generated catch block  
                    e1.printStackTrace();  
                }  
            }  
            childprocessalivebool = true;  
            Main.exec();  
        }
```

Bild 53. Warten das Drücken auf “Choose compiled File” Taste im Toollbar (JframeListener.java).

Beim Drücken auf die Taste “Choose ” im Toolbar wird den Working-Directory in einem Fenster geöffnet.

Die Funktion `jfc.showOpenDialog(null);` gibt den Zustand der ausgewählten Dateien in einem Enum-Integer zurück und der zurückgegebene Zustand kann wie folgt `JFileChooser.CANCEL_OPTION`, `JFileChooser.APPROVE_OPTION` oder `JFileChooser.ERROR_OPTION` (Wenn einen Fehler auftritt) sein.

(<https://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html>)

Falls der zurückgegebene Zustand `JFileChooser.APPROVE_OPTION` entspricht, wird dann die Datei über `jfc.getSelectedFile()` ausgewählt und der Pfad der ausgewählten Datei wird als Parameter in `setPath(selectedFile.getAbsolutePath());` gesetzt, damit der Pfad der ausgewählten Datei beim Lesen des Input-Streams des Subprozesses über `readChildProcessStream(getPath());` erhalten werden kann.

Soll auch vor dem Beginn eines neuen Prozesses sichergestellt werden, dass es keinen anderen Prozess laufend ist, um die Konflikte zu vermeiden, daher wird über `sp.childProcess.isAlive()` geprüft, ob es einen aktuellen laufenden Prozess gibt. Falls ein Prozess noch läuft, wird es über `sp.killSubProcess()` getötet.

Schließlich wird die Funktion in der Main-Klasse `Main.exec()` aufgerufen um der Input-Stream vom Subprozess zu übernehmen, verarbeiten und im Jpanel zeichnen.

2.7.6 Das Killing des Subprozesses

```
} else if (command.equals("killsubprocess")) { // Kindprozess eliminieren
    try {
        sp.killSubProcess();
    }
    catch (NullPointerException e1) {
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
}
```

Bild 54. Warten das Drücken auf "Kill Subprocess" Taste im Toolllbar (JframeListener.java).

Beim Drücken auf die Taste "Kill Subprocess" im Toolbar wird der erzeugte Subprozess im eigenen Thread getötet.

3. Auswertung des Endbenutzers

3.1 Testing von setPixel(int x,int y,int red,int green,int blue) in DrawGUI.cpp

```
class Rectangle : public Shape {
protected:
    int _width;
    int _height;
public:
    Rectangle(int x=0, int y=0, int width=0,int height=0,int red=0,int green=0,int blue=0);
    void draw();
    //double area();
    std::string shapeType();
};

Rectangle::Rectangle(int x, int y, int width,int height,int red,int green,int blue){
    _position = Position(x,y);
    _width = width;
    _height = height;
    _red = red;
    _green = green;
    _blue = blue;
}

void Rectangle::draw() {
    int i = _position.x;
    int w = _width;
    while(w >= 0){
        setPixel(i,_position.y,_red,_green,_blue);
        setPixel(i,_position.y+_height,_red,_green,_blue);
        i++;
        w--;
    }
    int i2 = _position.y;
    int h = _height;
    while(h >= 0){
        setPixel(_position.x,i2,_red,_green,_blue);
        setPixel(_position.x+_width,i2,_red,_green,_blue);
        i2++;
        h--;
    }
}

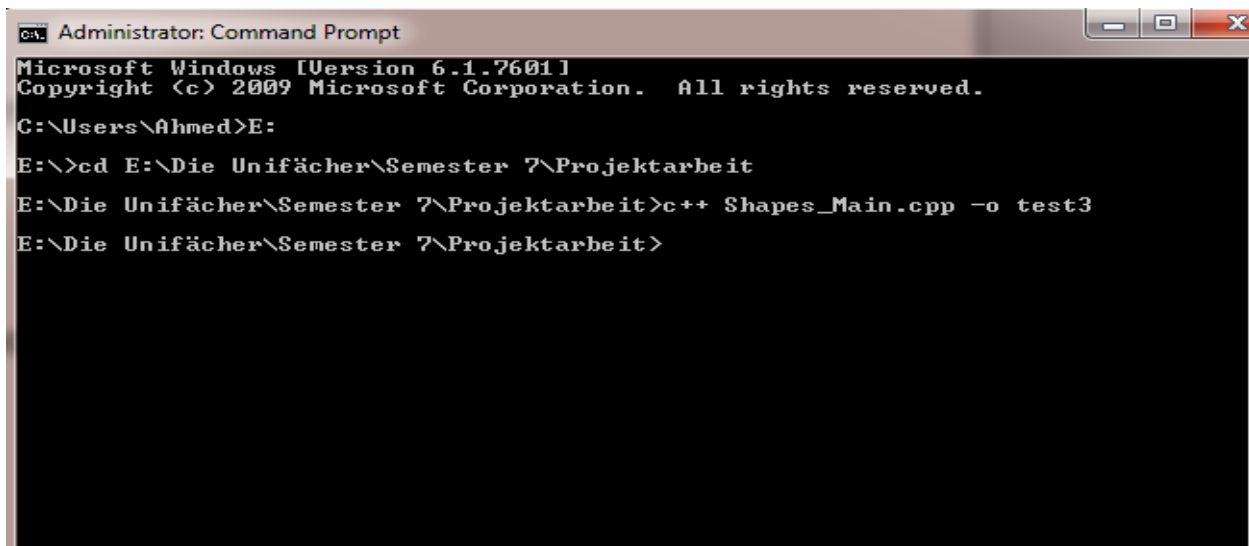
int main() {

    Rectangle* r = new Rectangle(200,200,250,250,100,0,255);

    r->draw();
}
```

Bild 55. Die Zeichnung eines Rechtecks über setPixel(int x,int y,int red,int green,int blue)(Shapes_Main.cpp).

Schritt 1: Die Kompilierung von Shapes_Main.cpp.



```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Ahmed>E:
E:\>cd E:\Die Unifächer\Semester 7\Projektarbeit
E:\Die Unifächer\Semester 7\Projektarbeit>c++ Shapes_Main.cpp -o test3
E:\Die Unifächer\Semester 7\Projektarbeit>
```

Bild 56. Kompilierung von Shapes_Main.cpp(Command Prompt).

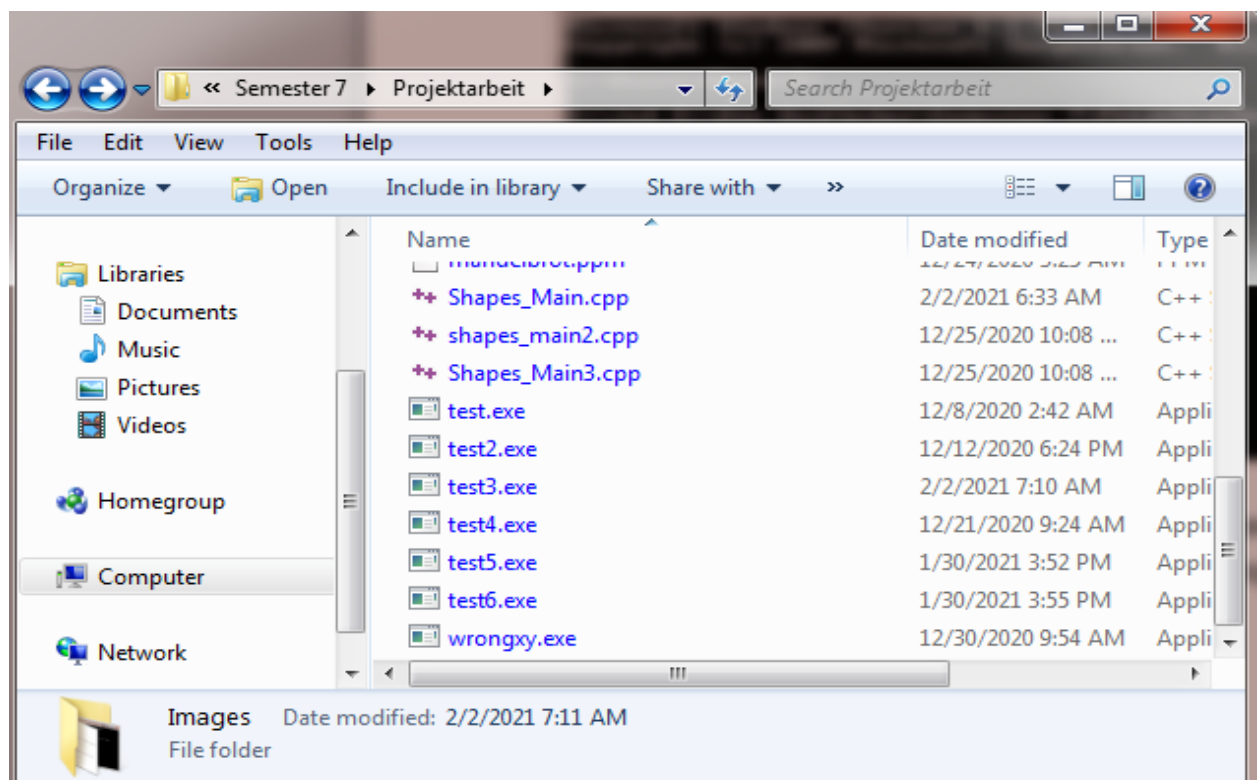


Bild 57. test3.exe-Datei wurde erfolgreich kompiliert.

Schritt 2: JavaSwing-Programm starten.

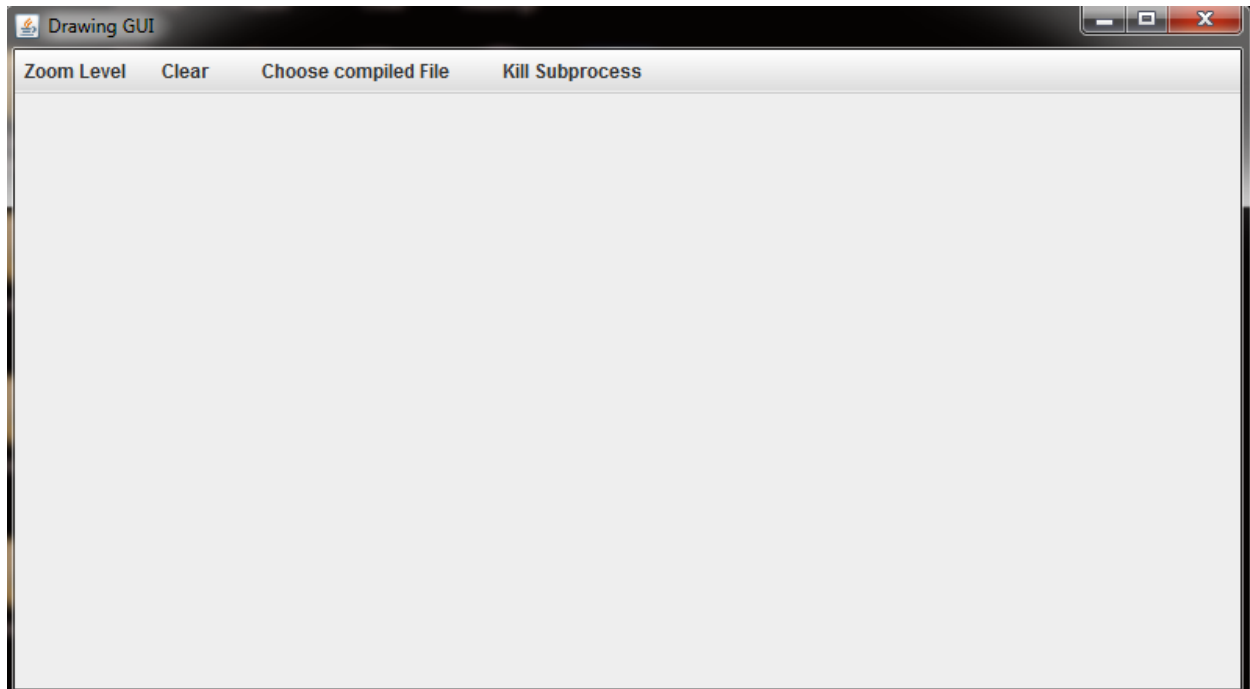


Bild 58. JavaSwing-Programm Starten (Java-Swing).

Schritt 3: Die Taste "Choose compiled File" drücken.

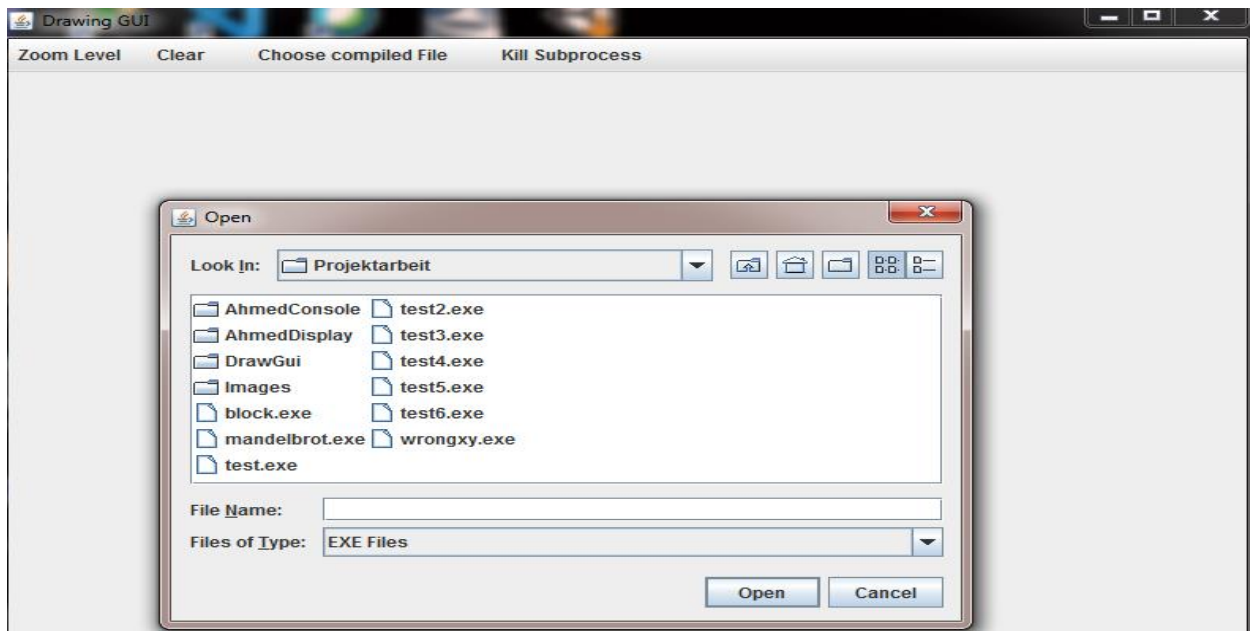


Bild 59. File Chooser (Java-Swing).

Schritt 4: Die kompilierte Datei test3 auswählen.

Ergebnis: Ein Rechteck wurde erfolgreich im Jpanel gezeichnet.

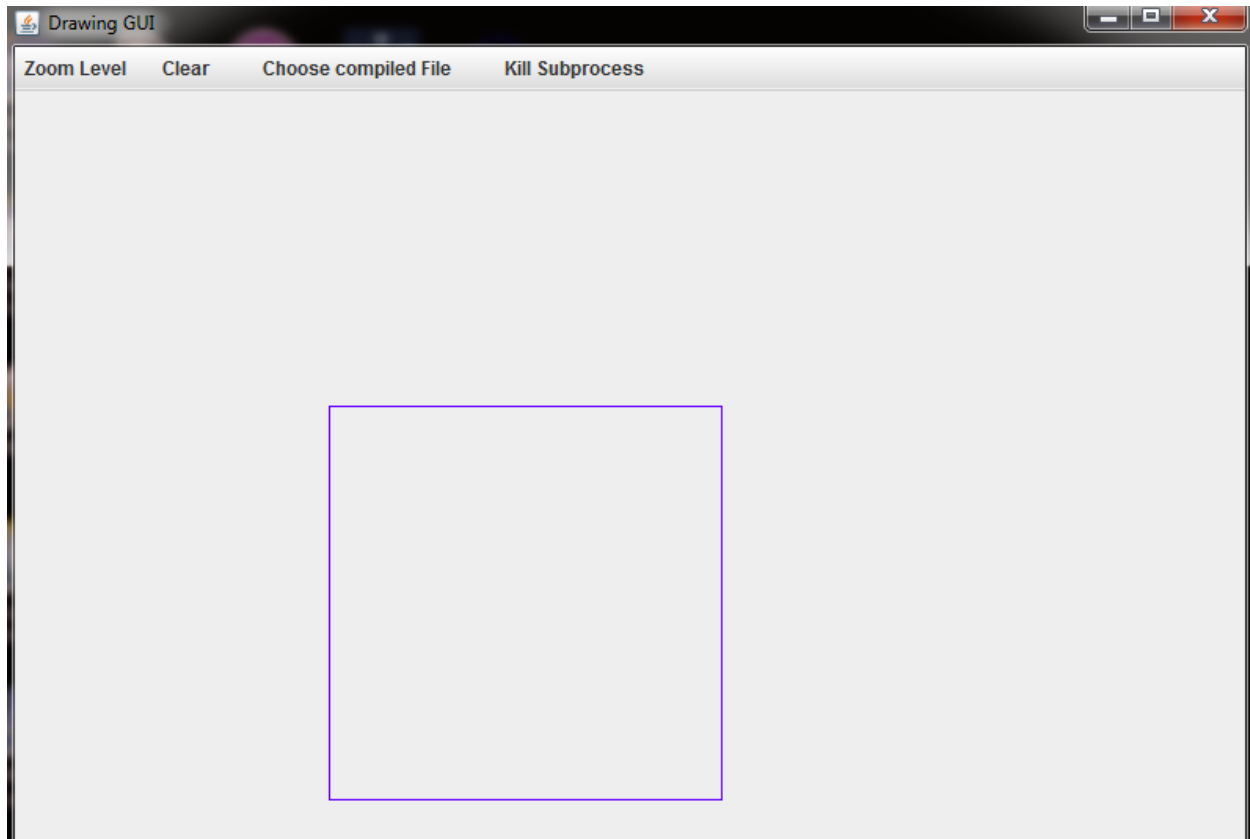


Bild 60. Die Zeichnung eines Rechtecks (Java-Swing).

Bemerkungen:

Die eingegebenen X,Y-Positionen in `setPixel(int x,int y,int red,int green,int blue)` wurden richtig im `JavaSwing-Jpanel` positioniert, sowie die richtige RGB-Farbe in `setPixel()` wurde genauso im `JavaSwing-Jpanel` gezeichnet.

Ich geheh davon aus, dass die Funktion `setPixel(int x,int y,int red,int green,int blue)` erfolgreich ausgeführt hat.

3.2 Testing von getPixel(int x,int y) in DrawGUI.cpp

```
void Shape::floodFill2(int x,int y,int red,int green,int blue,int jframewidth,int jframeheight){  
    set<Position*> Points;  
    set<Position*>::iterator itr;  
    Position* p = new Position(x,y);  
    Points.insert(p);  
  
    while(!Points.empty()){  
        itr = Points.begin();  
        Points.erase(itr);  
  
        Position* pbegin = *itr;  
  
        if((pbegin->x < jframewidth) && (pbegin->x > 0)&&  
            (pbegin->y < jframeheight) && (pbegin->y > 0) &&  
            (getPixel(pbegin->x,pbegin->y).compare("238,238,238") == 0)){  
  
            setPixel(pbegin->x,pbegin->y,red,green,blue);  
  
            Position* pright = new Position(pbegin->x+1,pbegin->y);  
            Points.insert(pright);  
            Position* pleft = new Position(pbegin->x-1,pbegin->y);  
            Points.insert(pleft);  
            Position* psouth = new Position(pbegin->x,pbegin->y+1);  
            Points.insert(psouth);  
            Position* pnorth = new Position(pbegin->x,pbegin->y-1);  
            Points.insert(pnorth);  
        }  
    }  
}  
  
int main() {  
  
    Rectangle* r = new Rectangle(200,200,250,250,100,0,255);  
  
    r->draw();  
  
    r->floodFill2(201,201,255,0,255,getJPanelWidth(),getJPanelHeight());  
}
```

Bild 61. Testen von getPixel(int x,int y) über FloodFill-Algorithmus (Shapes_Main.cpp).

Um getPixel(int x,int y) zu überprüfen habe ich das Algorithmus Flood-Fill implementiert. Das Algorithmus geht um die Pixel-Ausfüllung mit den ausgewählten RGB innerhalb der gezeichneten Grenzen.

Der Rückgabewert von getPixel(pbegin->x,pbegin->y); wird in einer While-Schleife mit dem String "238,238,238" verglichen um zu erkennen, ob die zurückgegebene RGB-Pixelfarbe eine Hintergrundfarbe ist. Falls eine Hintergrundfarbe erkannt wird, wird die vom Nutzer gegebenen X,Y-Position über setPixel(x, y,red,green,blue) mit den eingegebenen RGB ausgefüllt,

dann werden die Pixel in vier Richtungen (Ost,West,Nord,Süd) um das ausgewählte Pixel in der Positionen-Setliste eingefügt, aber wenn der Rückgabewert von `getPixel(pbegin->x,pbegin->y);` nicht eine Hintergrundfarbe ist, wird dann die nächste X,Y-Position in der Positionen-Setliste direkt abgeholt und so weiter bis zum Ende des Algorithmus.

Um das FloodFill-Algorithmus besser aufklären zu können, habe ich das folgende Diagramm vorbereitet.

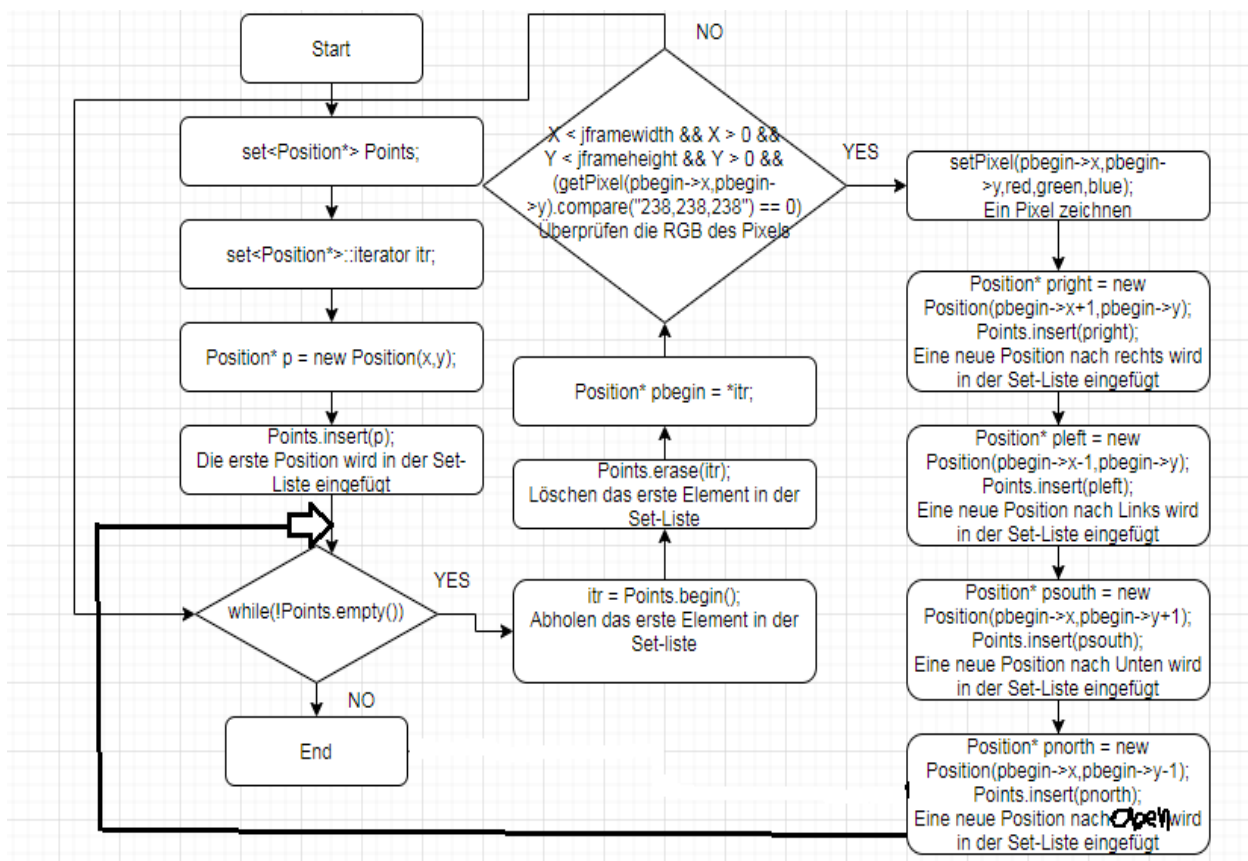


Bild 62. FloodFill-Algorithmusdiagramm.

Testing des Algorithmus:

Schritt 1: Die Zeichnung eines Rechtecks über `Rectangle* r = new Rectangle(200,200,250,250,100,0,255); r->draw();`.

Schritt 2: Die Ausfüllung des Rechtecks mit den gewünschten RGB-Pixelfarbe bei der Eingabe von X,Y-Position innerhalb des Rechtecks in der folgenden Funktion r->floodFill2(201,201,255,0,255,getJPanelWidth(),getJPanelHeight());.

Schritt 3: Ich habe die Datei test5.exe erfolgreich kompiliert.

Schritt 4: Ich habe dann die kompilierte Datei test5.exe über File-Chooser ausgewählt.

Ergebnis: Die inneren Flächen des Rechtecks wurden erfolgreich mit den eingegebenen RGB-Pixelfarbe ausgefüllt.

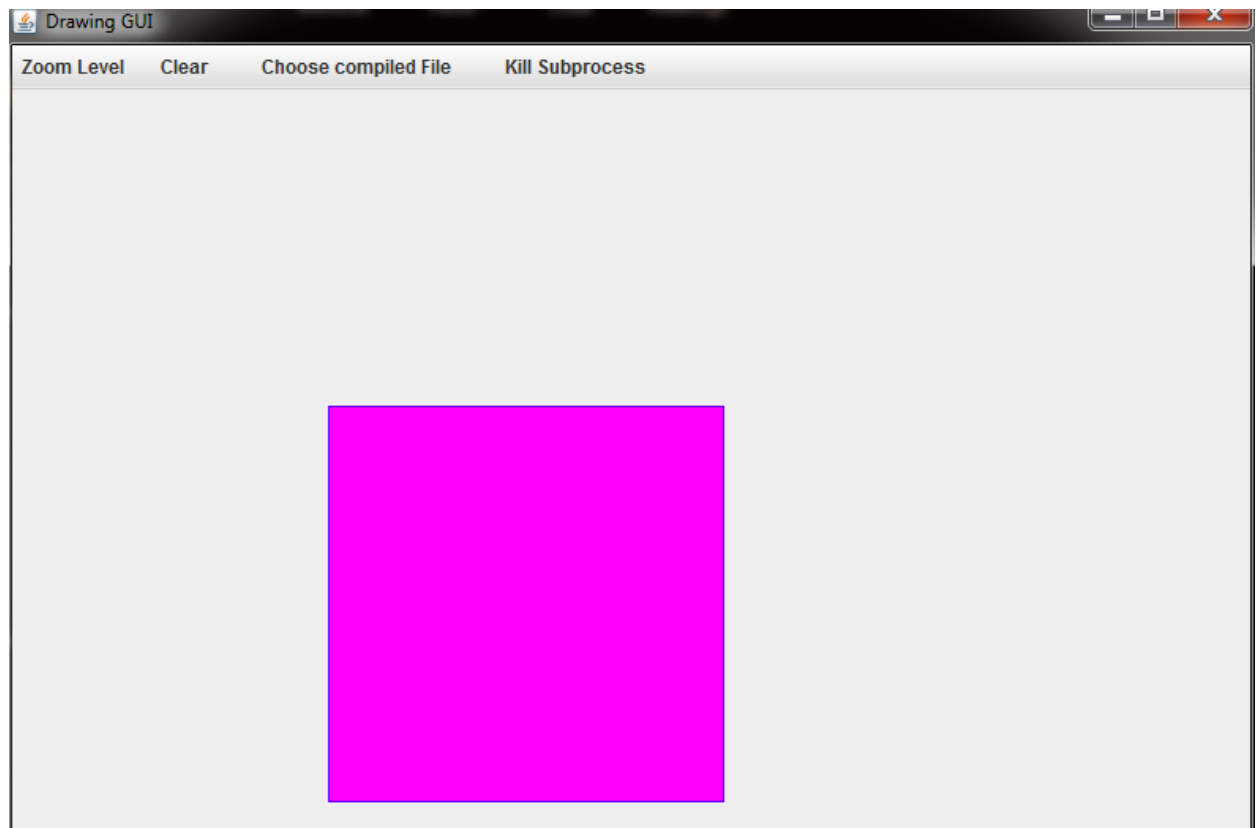


Bild 63. Die Zeichnung durch das Floodfill-Algorithmus (Java-Swing).

Bemerkungen:

Die inneren Flächen des Rechtecks wurden vollständig mit der eingegebenen RGB-Farbe ausgefüllt. Man kann davon erschließen, dass die Funktion `getPixel(int x,int y)` richtig funktioniert hat.

3.3 Testing von int getJPanelWidth() und int getJPanelHeight() in DrawGUI.cpp

```
class MandelBrot:public Shape {
public:
    MandelBrot();
    void draw(int,int,int);
    std::string shapeType();
    int value(int,int,int,int,int);
};

MandelBrot::MandelBrot() {

}

std::string MandelBrot::shapeType() {
    std::string s = "Das ist ein Mandelbrot";
    return s;
}

int MandelBrot::value (int x, int y,int width,int height,int iteration) {
    complex<float> point((float)x/width-1.5, (float)y/height-0.5);
    complex<float> z(0, 0);
    int nb_iter = 0;
    while (abs (z) < 2 && nb_iter <= iteration) {
        z = z * z + point;
        nb_iter++;
    }
    if (nb_iter < iteration)
        return (255*nb_iter)/20;
    else
        return 0;
}

void MandelBrot::draw(int width,int height,int iteration){

    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            int val = value(i, j,width,height,iteration);
            setPixel(i,j,val,0,0);
        }
    }
}
```

```

int main() {
    //Rectangle* r = new Rectangle(200,200,250,250,100,0,255);
    //r->draw();
    //r->floodFill12(201,201,255,0,255,getJPanelWidth(),getJPanelHeight());
    MandelBrot* mb = new MandelBrot();
    mb->draw(getJPanelWidth(),getJPanelHeight(),1000);
    return 0;
}

```

Bild 64. Die Zeichnung von einem Mandelbrot mit der Breite und der Höhe des Jpanels (Shapes_Main.cpp).

Schritt 1: Ich habe die Datei test6.exe erfolgreich kompiliert.

Schritt 2: Ich habe dann die kompilierte Datei test6.exe über File-Chooser ausgewählt.

Ergebnis: Ein Mandelbrot wurde erfolgreich vollständig mit der Breite und der Höhe des Jpanels im Jpanel gezeichnet.

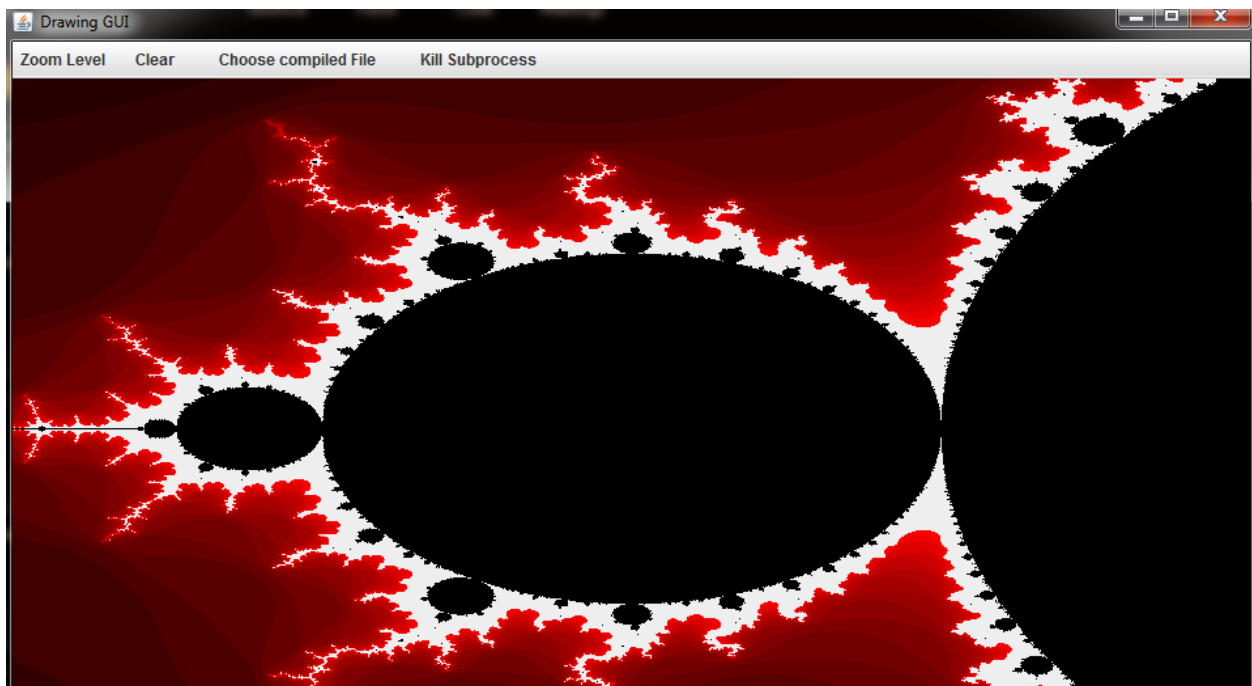


Bild 65. Die Zeichnung von einem Mandelbrot (Java-Swing).

3.4 Testing das Zentrum-Zooming

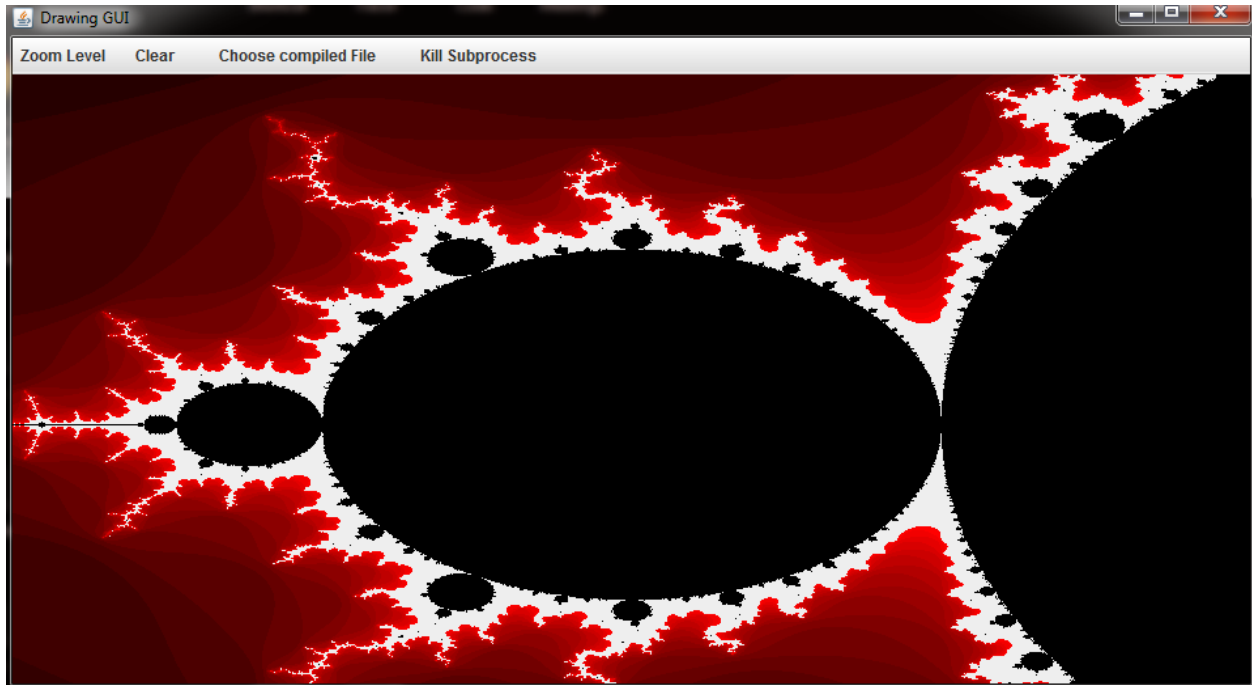


Bild 66. Jpanel ohne Zooming (Java-Swing).

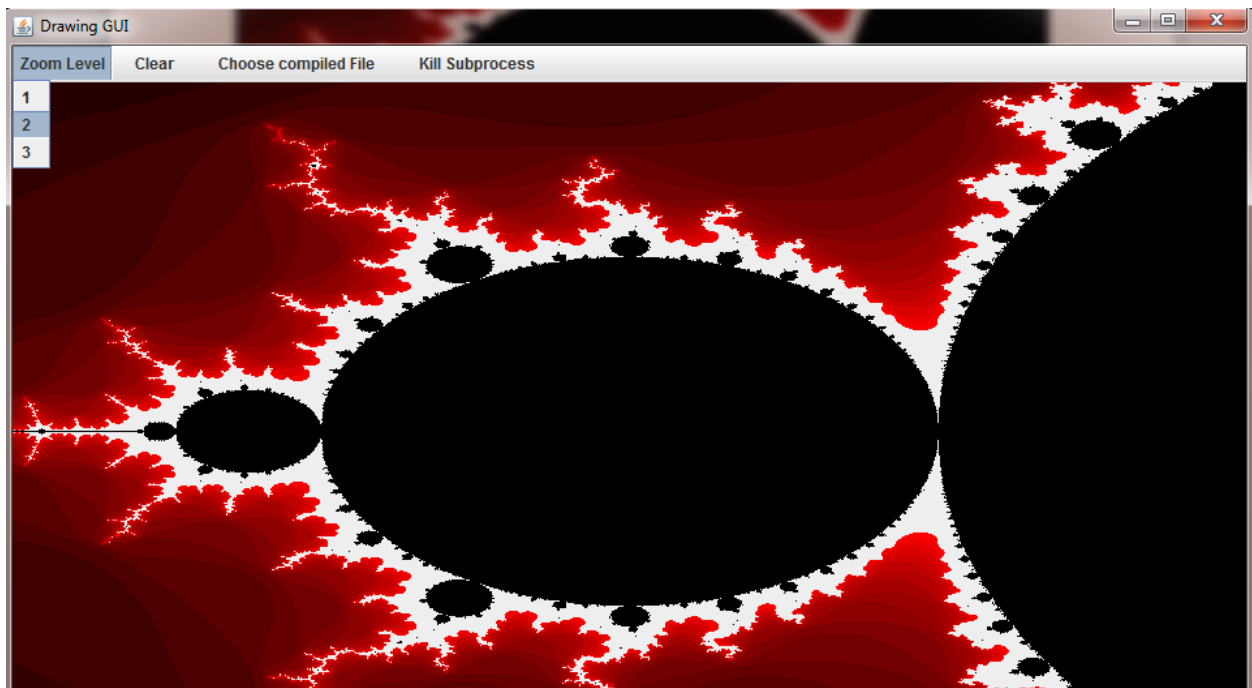


Bild 67. Zoom-Level 2 auswählen (Java-Swing).

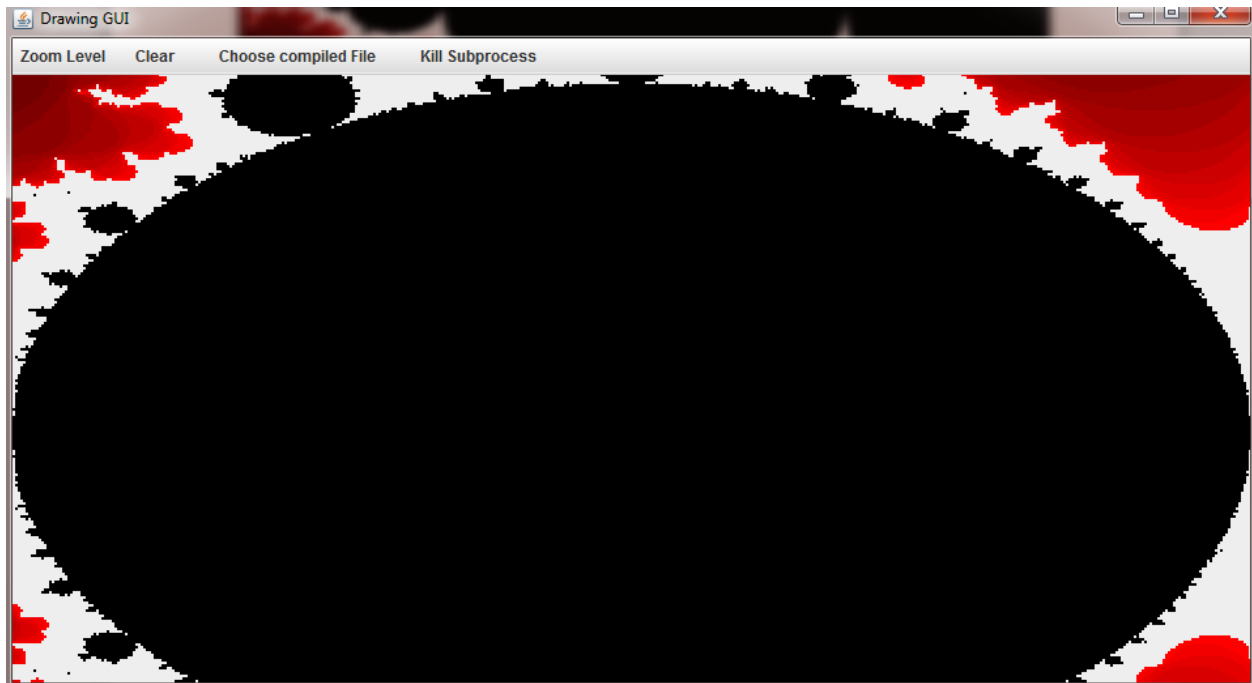


Bild 68. Jpanel mit einem Zooming-Faktor 2 (Java-Swing).

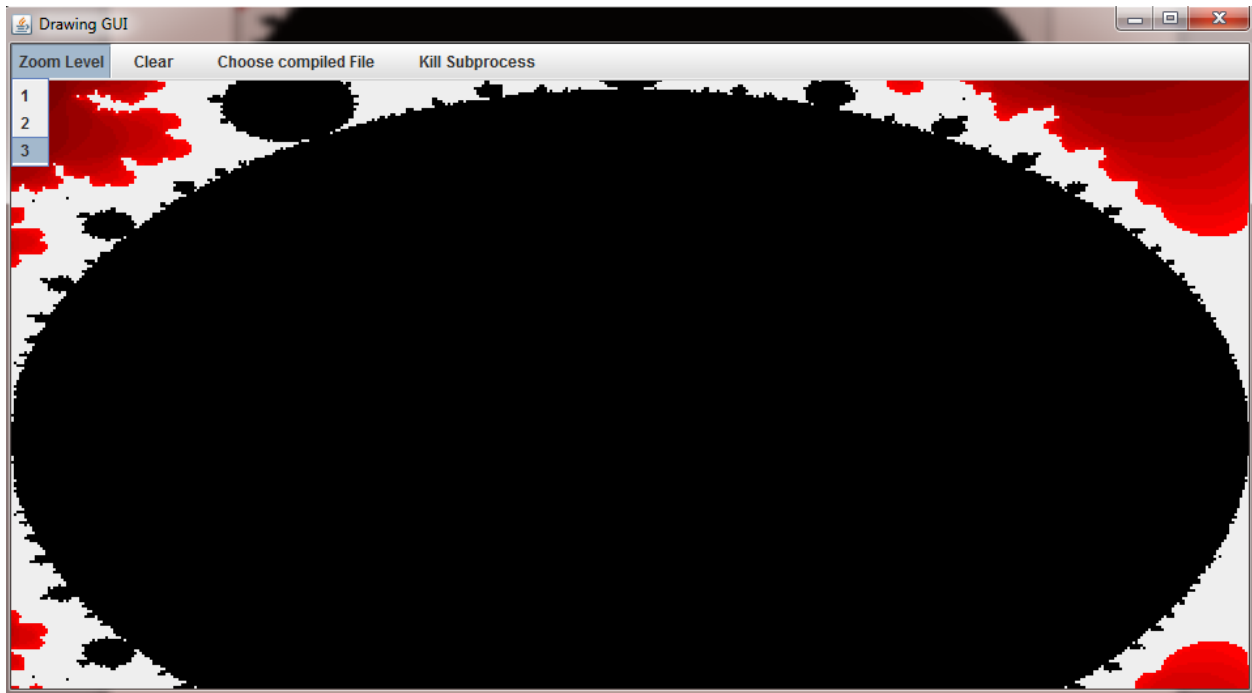


Bild 69. Zoom-Level 3 auswählen (Java-Swing).

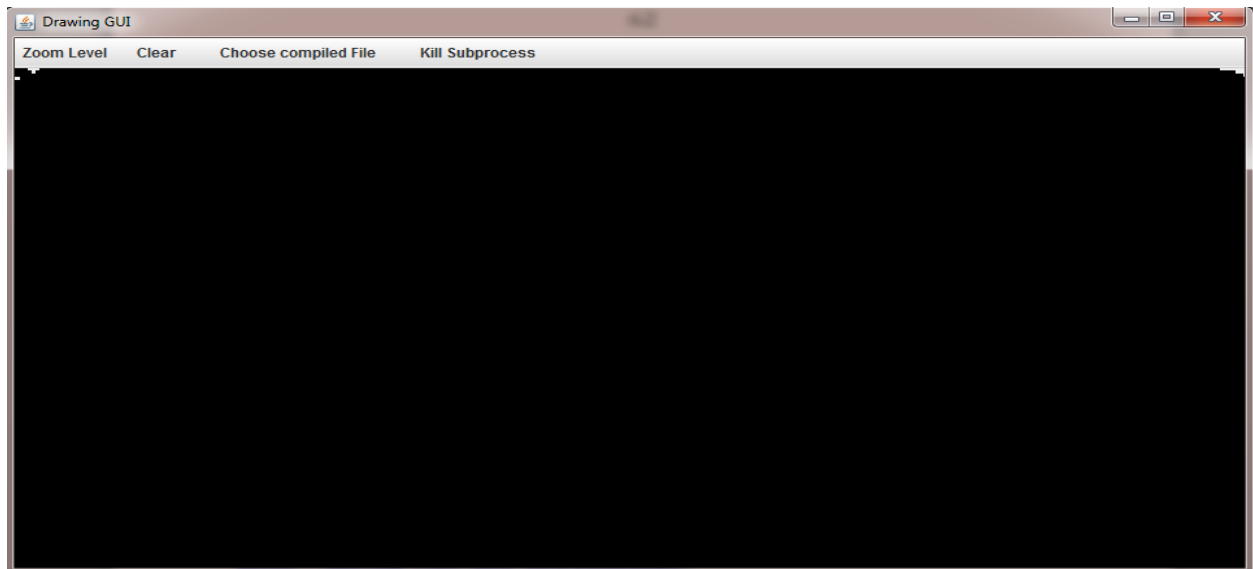


Bild 70. Jpanel mit einem Zooming-Faktor 3 (Java-Swing).

Ergebnis:

Das Zentrum-Zooming wurde erfolgreich durchgeführt und kann der Nutzer das Zentrum des Jpanels bis zum Faktor 3 zoomen.

Bemerkungen:

Der Nutzer kann das Jpanel zoomen und zurück auszoomen.

3.5 Testing das Zooming gemäß der Mouse-Position

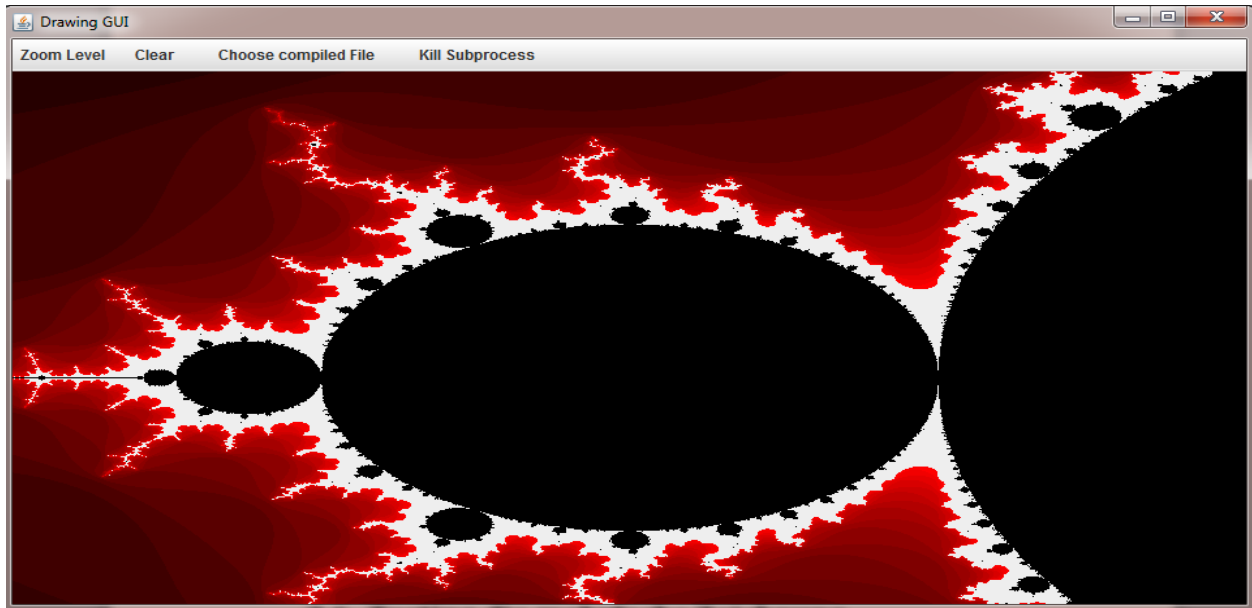


Bild 71. Jpanel ohne Zooming (Java-Swing).

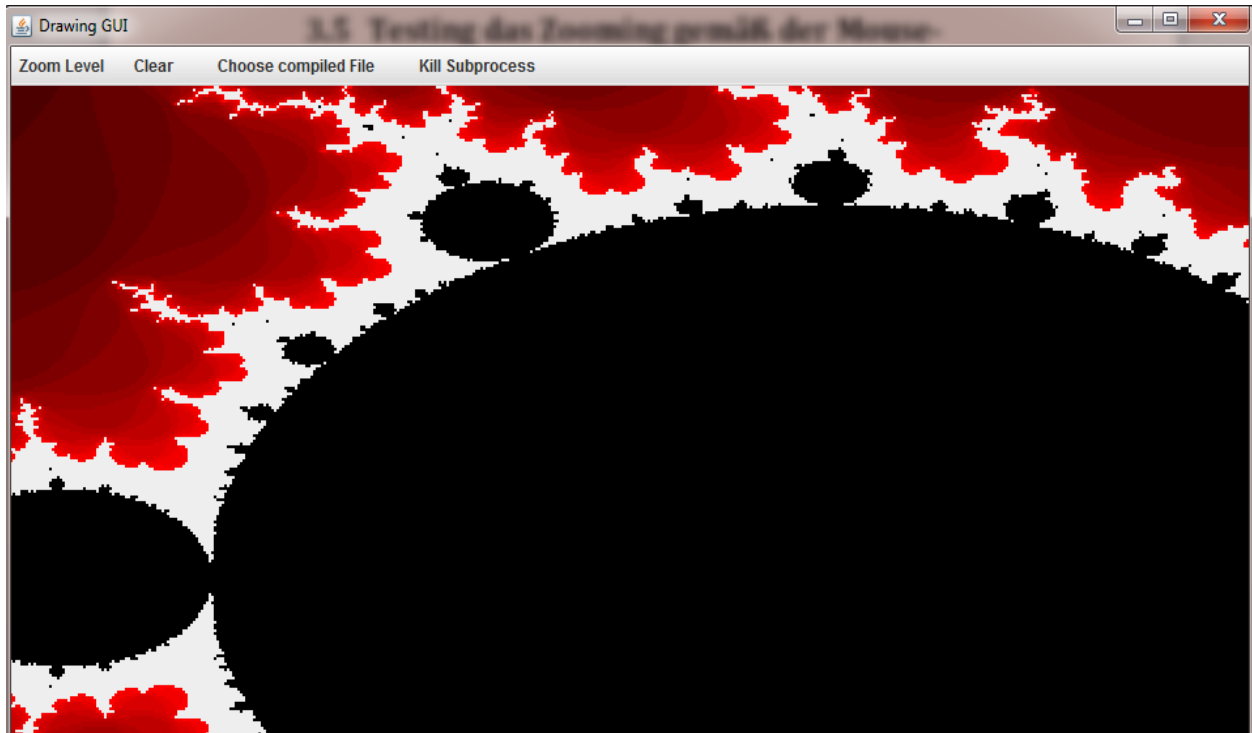


Bild 72. Jpanel mit einem Zooming-Faktor 2 (Java-Swing).



Bild 73. Jpanel mit einem Zooming-Faktor 3 (Java-Swing).

Ergebnis:

Das Zooming gemäß der Mouse-Position wurde erfolgreich durchgeführt und kann der Nutzer beim Drücken auf der Taste “+” das Jpanel durch die Mouse-Position bis zum Faktor 3 zoomen und beim Drücken auf der Taste “-” das Jpanel durch die Mouse-Position auszoomen.

3.6 Testing die Fähigkeit der mehrfachen Zeichnung

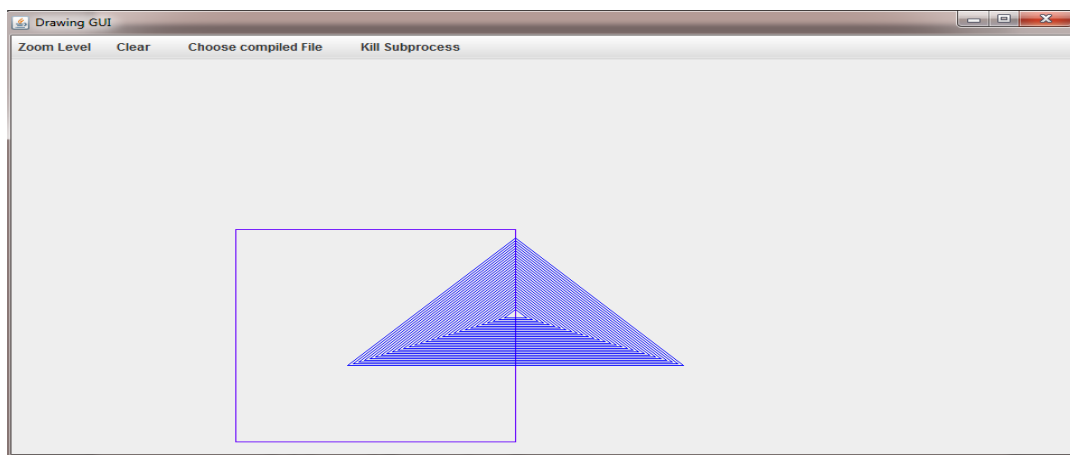


Bild 74. Die Mehrfache Zeichnung (Java-Swing).

Ergebnis:

Der Nutzer kann mehrere kompilierte Dateien nacheinander im Java-Swing zeichnen ohne die alte Zeichnung zu verlieren.

3.7 Testing die Löschung der Pixel im Jpanel

Ergebnis:

Clear-Taste hat bei mir vernünftig funktioniert und hat die notwendige Aufgabe davon geschafft. Beim Drücken die Taste Clear im JFrame werden alle die Pixel im Jpanel gelöscht.

3.8 Testing die Eingabe von falschen X,Y-Koordinaten im Jpanel

```
int main() {  
  
    Rectangle* r = new Rectangle(1000,1000,250,250,100,0,255);  
  
    r->draw();  
}
```

Bild 75. Die X,Y-Koordinaten überschreiten die Breite und die Höhe des Jpanels (Shapes_Main.cpp).

Schritt 1: Ich habe die Datei wrongxy.exe erfolgreich kompiliert.

Schritt 2: Ich habe dann die kompilierte Datei wrongxy.exe über File-Chooser ausgewählt.

Ergebnis: Die Ausführung des Subprozesses wurde gestoppt und der Subprozess wurde getötet, wenn die falschen X,Y-Koordinaten gegeben wurden.

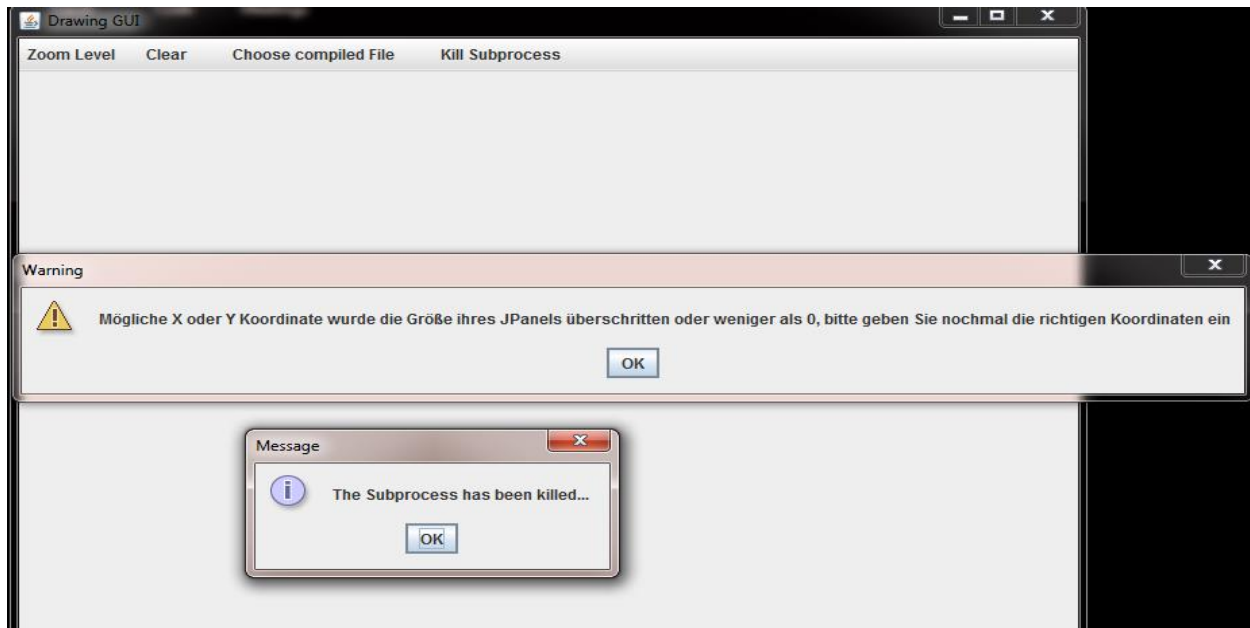


Bild 76. Die X,Y-Koordinaten sind falsch gegeben (Java-Swing).

3.9 Testing das Killing des Subprozesses

```
int main() {
    while(true){
        getPixel(0,0);
    }
}
```

Bild 77. Der Subprozess geht in einer Endlosschleife (Shapes_Main.cpp).

Schritt 1: Ich habe ein Subprozess in einer Endlosschleife erstellt.

Schritt 2: Dann habe ich die Datei block.exe erfolgreich kompiliert.

Schritt 3: Danach habe ich die kompilierte Datei block.exe über File-Chooser ausgewählt.

Schritt 4: Die laufenden Prozessen zeigen lassen.

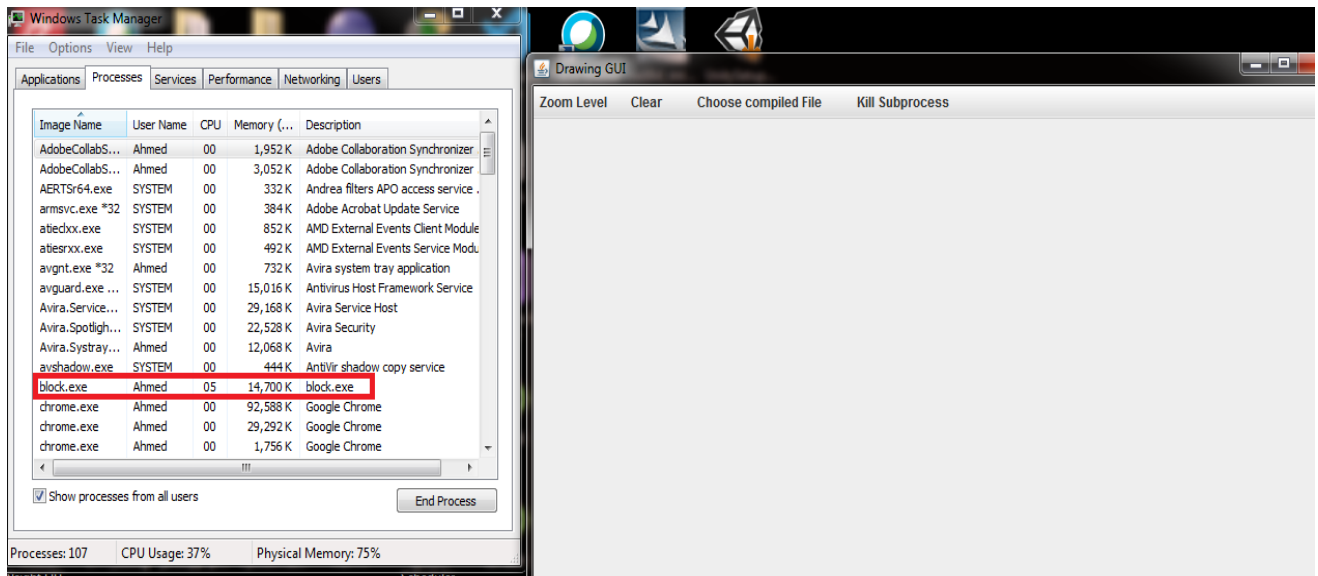


Bild 78. Zeigen der laufenden Prozessen (Windows Task Manager).

Schritt 5: Killing des Subprozesses beim Drücken auf die Taste “Kill Subprocess”.

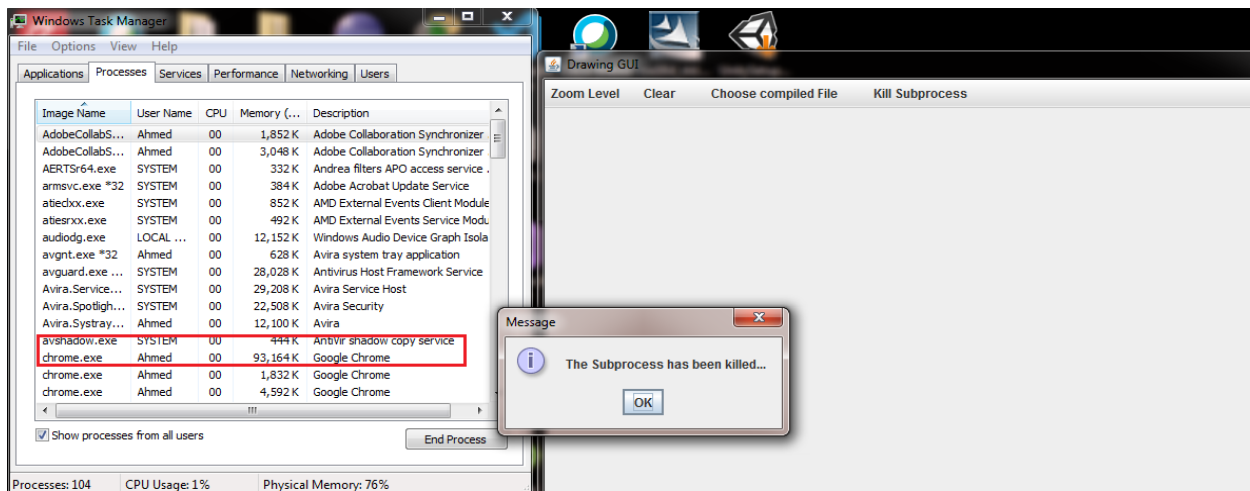


Bild 79. Der Prozess block.exe wurde erfolgreich getötet und von den laufenden Prozessen gelöscht (Windows Task Manager).

Ergebnis: Der Nutzer ist fähig beim Drücken auf die Taste “Kill Subprocess” einen Subprozess zu töten, wenn es blockiert sein könnte.

3.10 Allgemeine Fragen für die Bewertung der Anwendung

Bewertungsniveau	schlecht	Mittelwert	gut
Sehen Sie, dass diese Anwendung hilfreich ist			X
Finden Sie diese Anwendung flexibel zu nutzen			X
Können Sie die kompilierten Dateien durch den File-Chooser auswählen			X
Können Sie dadurch die gewünschten Pixel durch den C++-Programm in Java-Swing zeichnen			X
Können Sie irgend ein Pixel in Java-Swing abholen , lesen und zeigen lassen			X
Können Sie Die Breite und die Höhe des Jpanels im C++-Programm abholen			X
Können Sie die gezeichneten Pixel im Jpanel löschen			X
Finden Sie die Anwendung effizient genug für die mehrfache Zeichnung			X
Finden Sie die Anwendung schnell genug in der Bearbeitung			X
Übliche Problembehandlung			X
Können Sie irgend ein Prozess töten, wenn es blockiert sein könnte			X
Können Sie Das Zentrum des Jpanels Zoomen			X
Ist das Zooming gemäß der Mouse-Position möglich			X
Gibt es keine Veränderung in den Positionen der Pixel nach dem Zooming			X
Ist diese Anwendung erweiterbar		X	

4. Quellen

[1] *July 6, 2020*. Java ProcessBuilder tutorial. Abgerufen von <https://zetcode.com/java/processbuilder/>

[2] GUI-Programmierung mit Swing. Abgerufen von <https://www.java-tutorial.org/swing.html>

[3] Ricky Barnes 11-Oct-2018. Process vs Parent Process vs Child Process. Abgerufen von <https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process>

[4] Tim Aschermann 13.11.2015. Was ist RGB? Einfach und verständlich erklärt. Abgerufen von https://praxistipps.chip.de/was-ist-rgb-einfach-und-verstaendlich-erklaert_44407

[5] <https://www.mksoftware.com/docs/man3/fork.3.asp>

[6] Heidemarie Schuster 17.09.2019. Was sind Pixel?. Abgerufen von <https://www.it-business.de/was-sind-pixel-a-925513/>

[7] I/Ostreams. Abgerufen von <https://docs.oracle.com/javase/tutorial/essential/io/streams.html>

[8] Data Buffer. Abgerufen von https://en.wikipedia.org/wiki/Data_buffer

[9] Java JFrame. Abgerufen von <https://www.javatpoint.com/java-jframe>

[10] Schneider, Fred B. (1997). Bei gleichzeitiger Programmierung . Springer-Verlag New York, Inc. ISBN 978-0-387-94942-0. Synchronisation (Informatik). Abgerufen von [https://de.qaz.wiki/wiki/Synchronization \(computer science\)](https://de.qaz.wiki/wiki/Synchronization_(computer_science))

[11] JPanel. Abgerufen von <https://www.java-tutorial.org/jpanel.html>

[12] How to implement ActionListener in Java. Abgerufen von <https://javapointers.com/java/java-se/actionlistener/>

[13] Class AbstractButton. Abgerufen von <https://docs.oracle.com/javase/8/docs/api/javax/swing/AbstractButton.html>

[14] Class JFrame. Abgerufen von <https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>

[15] Class Component. Abgerufen von <https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html>

[16] Class Button. Abgerufen von <https://docs.oracle.com/javase/7/docs/api/java/awt/Button.html>

[17] Interface MouseMotionListener. Abgerufen von <https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseMotionListener.html>

[18] C++ sprintf(). Abgerufen von <https://www.programiz.com/cpp-programming/library-function/cstdio/sprintf>

[19] Rendern – das große Finale der Bild- und Videobearbeitung, 28.12.20.

Abgerufen von

<https://www.ionos.de/digitalguide/websites/webdesign/rendern/>

[20] Double-Buffering. Abgerufen von

https://en.wikipedia.org/wiki/Multiple_buffering

[21] Class AffineTransform. Abgerufen von

<https://docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html>

[22] Class JFileChooser. Abgerufen von

<https://docs.oracle.com/javase/7/docs/api/javawx/swing/JFileChooser.html>

[23] Class Process. Abgerufen von

<https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html>