GUI 5 E

*Team Members:*

*David Milum, Matt Gold, Michael Burlison, Michael Hardeman, Robert Jenkins*

*CSC4350/6350 – Software Engineering*

*Semester: Spring 2012*

*Project: Pixel editor*

*April 19, 2012*

This page intentionally left blank

## Table of Contents

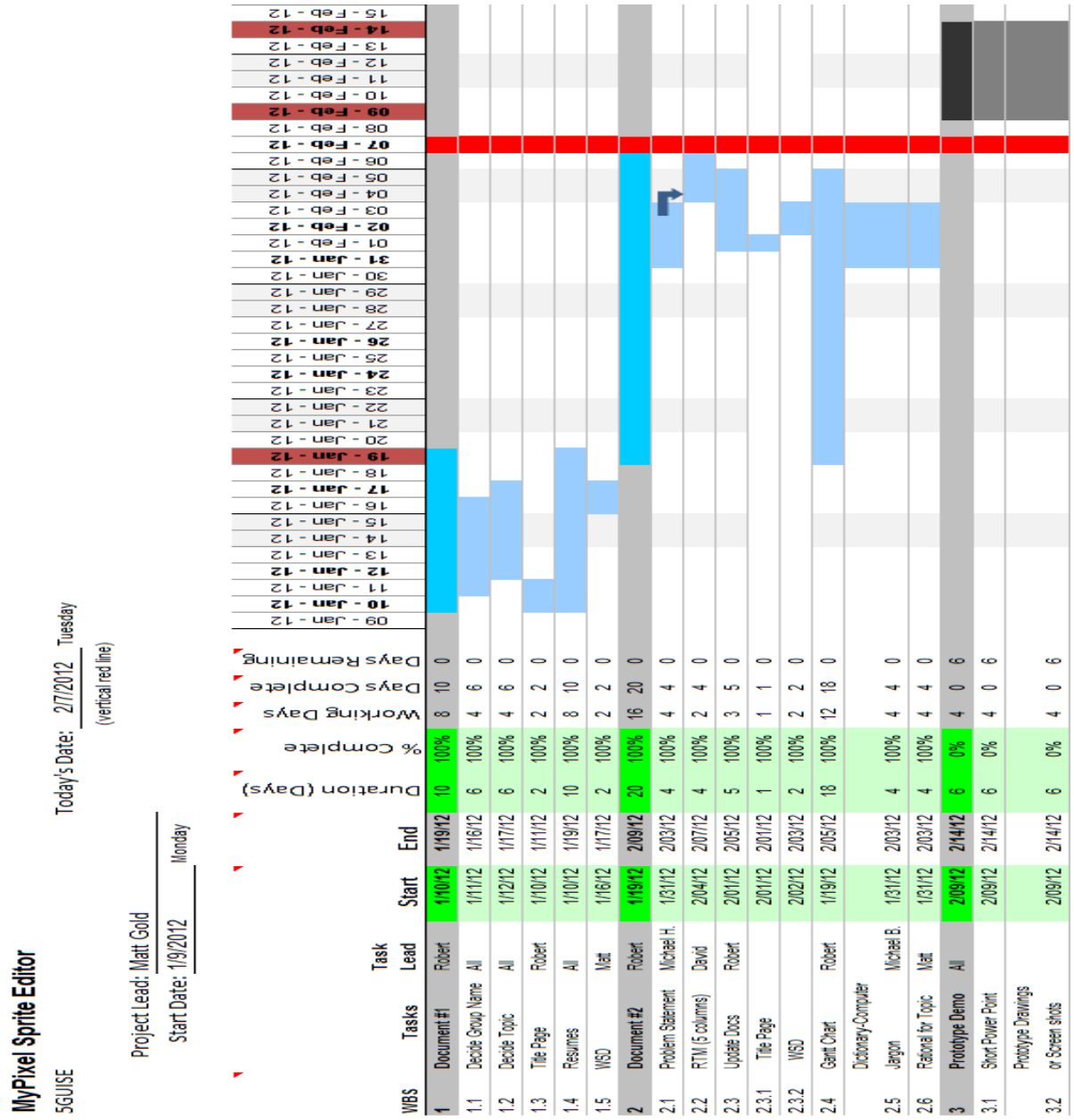
## 1.0 Document Change History

| Date | Author(s) | Description of Change |
|---|---|---|
| 01.15.2012 | All | Original Issue, Report 1 |
| 02.05.2012 | Matt G. | Added Topic Rationale |
| 02.07.2012 | Michael H. | Added Problem Statement |
| 02.07.2012 | David M. | Added Requirements Traceability Matrix |
| 02.08.2012 | Michael B. | Added Glossary |
| 02.08.2012 | Robert J. | Added Gantt Chart |
| 02.08.2012 | Matt G. | Added Requirements Rationale |
| 02.11.2012 | Michael B. | Added Horizontal Prototype |
| 02.13.2012 | Michael B. | Updated Glossary |
| 02.13.2012 | David M. | Updated Requirements Traceability Matrix |
| 02.15.2012 | All | Added Use Cases |
| 02.18.2012 | David M. | Added Function Point Cost Analysis |
| 02.19.2012 | Robert J. | Updated Gantt Chart |
| 02.19.2012 | Michael H. | Updated Work Statement Diagram |
| 02.20.2012 | Matt G. | Updated Requirements Rationale |
| 03.03.2012 | Robert J | Updated Work Statement Diagram |
| 03.06.2012 | All | Added Interaction Diagrams |
| 03.07.2012 | Michael B. | Updated Gantt Chart |
| 03.07.2012 | Matt G. | Added Software Architecture |
| 03.07.2012 | Michael H., Michael B. | Added Class Interface |
| 03.07.2012 | David M. | Updated Requirements Traceability Matrix |
| 03.07.2012 | Michael B. | Added Category Interaction Diagram |
| 03.09.2012 | Robert J. | Combine Rationale Documents into Project Rationale |
| 03.12.2012 | Matt G. | Updated WSD |
| 03.14.2012 | David M. | Updated RTM |
| 03.16.2012 | Michael H. | Updated Gantt Chart |
| 03.18.2012 | Michael B. | Updated Dictionary |
| 03.28.2012 | Michael B. | Added COCOMO |
| 03.29.2012 | Robert J. | Added Test Cases & Rationale |
| 04.01.2012 | Robert J. | Updated WSD |
| 04.01.2012 | Michael B. | Updated RTM |
| 04.09.2012 | Matt G. | Updated Gantt Chart |
| 04.11.2012 | David M. | Updated WSD |
| 04.12.2012 | David M. | Updated Dictionary |
| 04.13.2012 | Michael H. | Added Project Legacy |
| 04.16.2012 | Robert J. | Added Source Code |
| 04.17.2012 | Robert J. | Added Functional Point Cost Analysis vs. COCOMO |
| 04.18.2012 | Michael B. | Added User Manual |

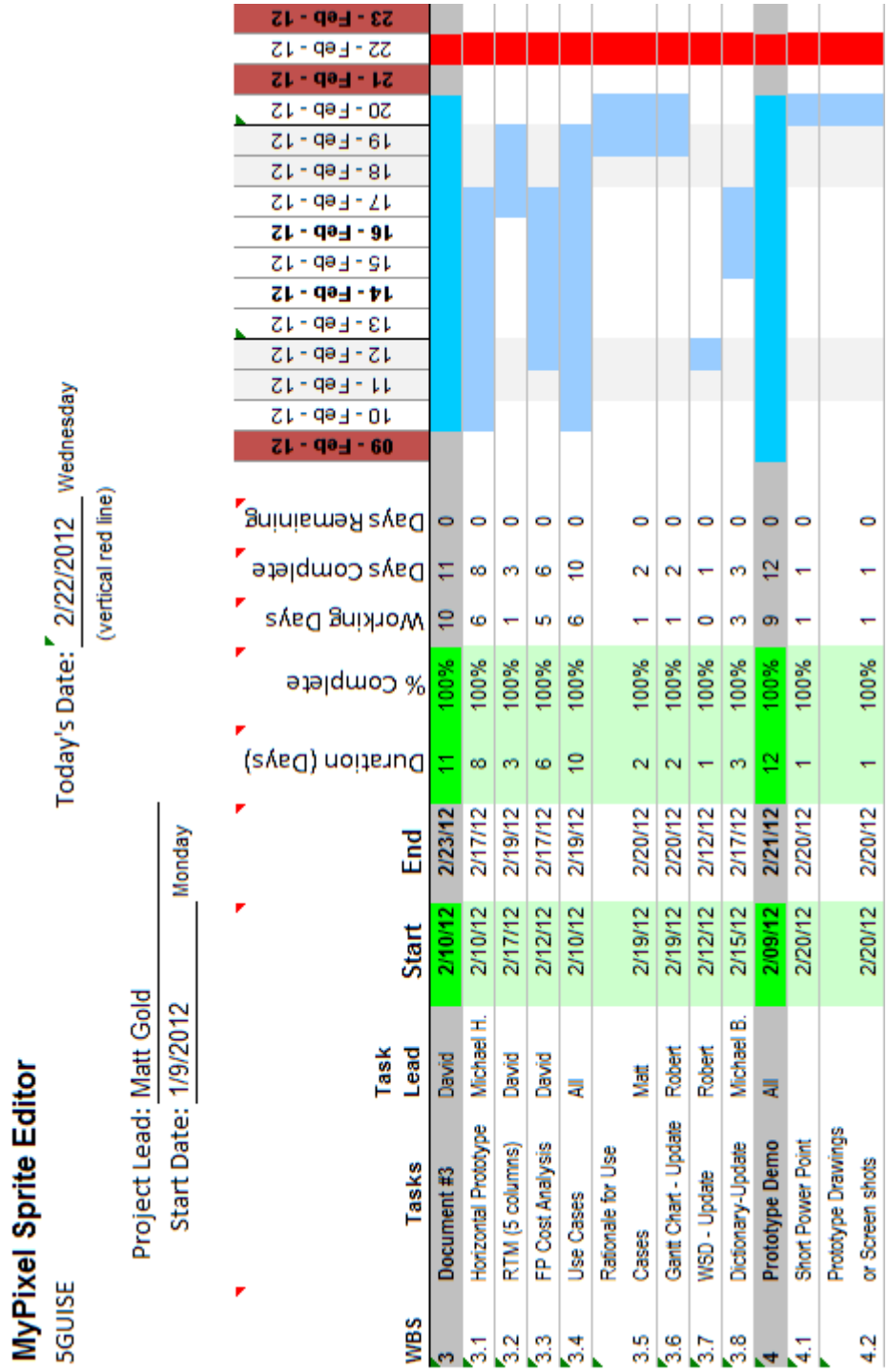
This page intentionally left blank

## 2.0 Gantt Chart

### 2.1 Report 1 &2 Deliverables



## 2.2 Report 3 & Prototype Deliverables



**MyPixel Sprite Editor**

5GUISE

Today's Date: 2/22/2012 Wednesday

(vertical red line)

Project Lead: Matt Gold

Start Date: 1/9/2012 Monday

| WBS | Tasks | Task Lead | Start | End | Duration (Days) | % Complete | Working Days | Days Complete | Days Remaining |
|---|---|---|---|---|---|---|---|---|---|
| 3 | Document #3 | David | 2/10/12 | 2/23/12 | 11 | 100% | 10 | 11 | 0 |
| 3.1 | Horizontal Prototype | Michael H. | 2/10/12 | 2/17/12 | 8 | 100% | 6 | 8 | 0 |
| 3.2 | RTM (5 columns) | David | 2/17/12 | 2/19/12 | 3 | 100% | 1 | 3 | 0 |
| 3.3 | FP Cost Analysis | David | 2/12/12 | 2/17/12 | 6 | 100% | 5 | 6 | 0 |
| 3.4 | Use Cases | All | 2/10/12 | 2/19/12 | 10 | 100% | 6 | 10 | 0 |
| 3.5 | Rationale for Use Cases | Matt | 2/19/12 | 2/20/12 | 2 | 100% | 1 | 2 | 0 |
| 3.6 | Gantt Chart - Update | Robert | 2/19/12 | 2/20/12 | 2 | 100% | 1 | 2 | 0 |
| 3.7 | WSD - Update | Robert | 2/12/12 | 2/12/12 | 1 | 100% | 0 | 1 | 0 |
| 3.8 | Dictionary-Update | Michael B. | 2/15/12 | 2/17/12 | 3 | 100% | 3 | 3 | 0 |
| 4 | Prototype Demo | All | 2/09/12 | 2/21/12 | 12 | 100% | 9 | 12 | 0 |
| 4.1 | Short Power Point | | 2/20/12 | 2/20/12 | 1 | 100% | 1 | 1 | 0 |
| 4.2 | Prototype Drawings or Screen shots | | 2/20/12 | 2/20/12 | 1 | 100% | 1 | 1 | 0 |

09 - Feb - 12 | 10 - Feb - 12 | 11 - Feb - 12 | 12 - Feb - 12 | 13 - Feb - 12 | 14 - Feb - 12 | 15 - Feb - 12 | 16 - Feb - 12 | 17 - Feb - 12 | 18 - Feb - 12 | 19 - Feb - 12 | 20 - Feb - 12 | 21 - Feb - 12 | 22 - Feb - 12 | 23 - Feb - 12

## MyPixel Sprite Editor

5GUISE

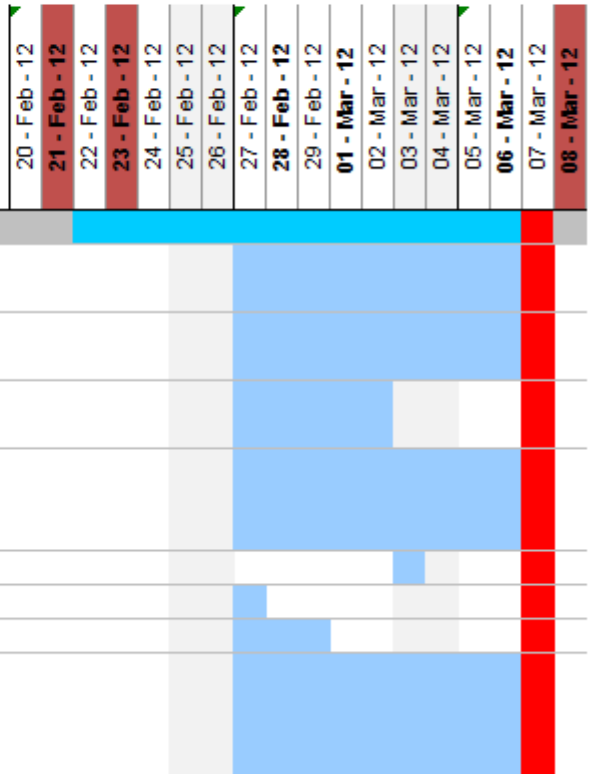
Today's Date: 3/7/2012 Wednesday

(vertical red line)

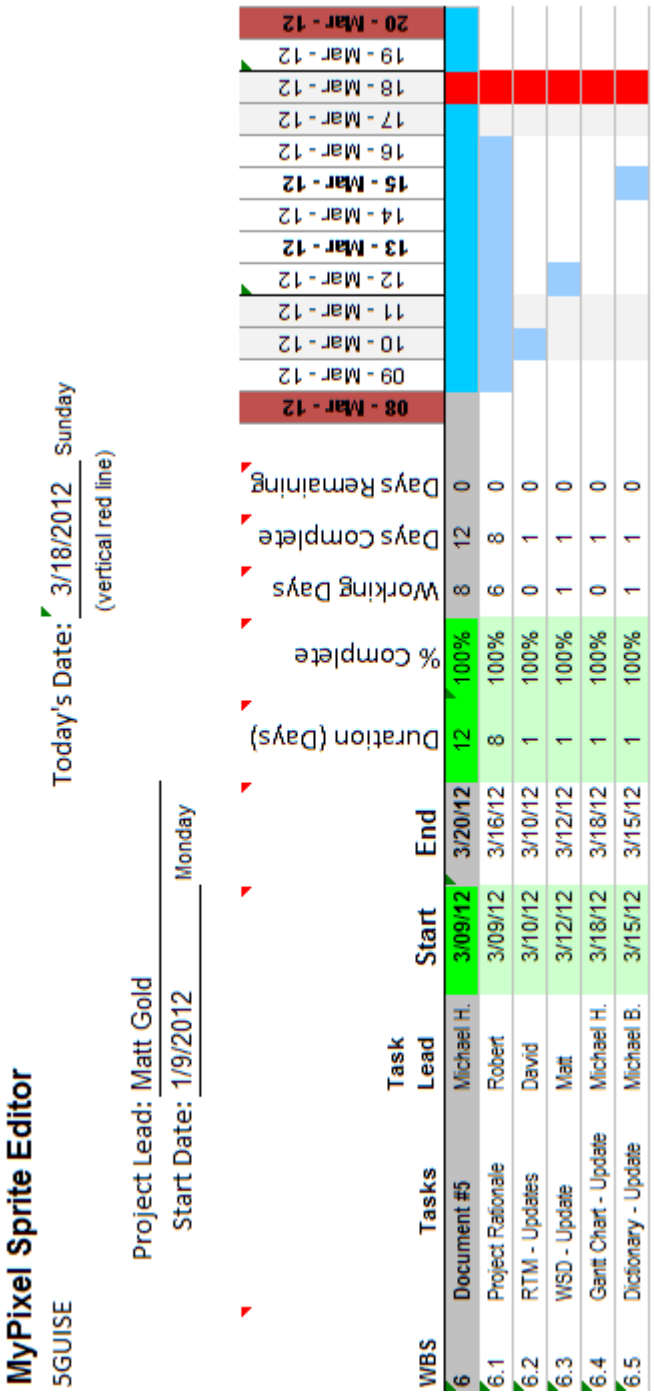
Project Lead: Matt Gold

Start Date: 1/9/2012 Monday

| WBS | Tasks | Task Lead | Start | End | Duration (Days) | % Complete | Working Days | Days Complete | Days Remaining |
|---|---|---|---|---|---|---|---|---|---|
| 5 | **Document #4** | Michael B. | 2/22/12 | 3/08/12 | 14 | 100% | 12 | 14 | 0 |
| 5.1 | Software Architecture Used | Matt | 2/27/12 | 3/06/12 | 9 | 100% | 7 | 9 | 0 |
| 5.2 | RTM (5 columns + Updates) | David | 2/27/12 | 3/06/12 | 9 | 100% | 7 | 9 | 0 |
| 5.3 | Category Interaction Diagram | Michael B. | 2/27/12 | 3/02/12 | 5 | 100% | 5 | 5 | 0 |
| 5.4 | Design of Each Application Object (ie. class interface) | David and Michael H. | 2/27/12 | 3/06/12 | 9 | 100% | 7 | 9 | 0 |
| 5.5 | WSD - Update | Robert | 3/03/12 | 3/03/12 | 1 | 100% | 0 | 1 | 0 |
| 5.6 | Gantt Chart - Update | Michael B. | 2/27/12 | 2/27/12 | 1 | 100% | 1 | 1 | 0 |
| 5.7 | Dictionary-Update | Michael B. | 2/27/12 | 2/29/12 | 3 | 100% | 3 | 3 | 0 |
| 5.8 | Rational for Application Objects, Software Architecture, etc. | Matt | 2/27/12 | 3/06/12 | 9 | 100% | 7 | 9 | 0 |



## 2.4 Report 5 Deliverable



**MyPixel Sprite Editor**

5GUISE

Today's Date: 3/18/2012 Sunday

(vertical red line)

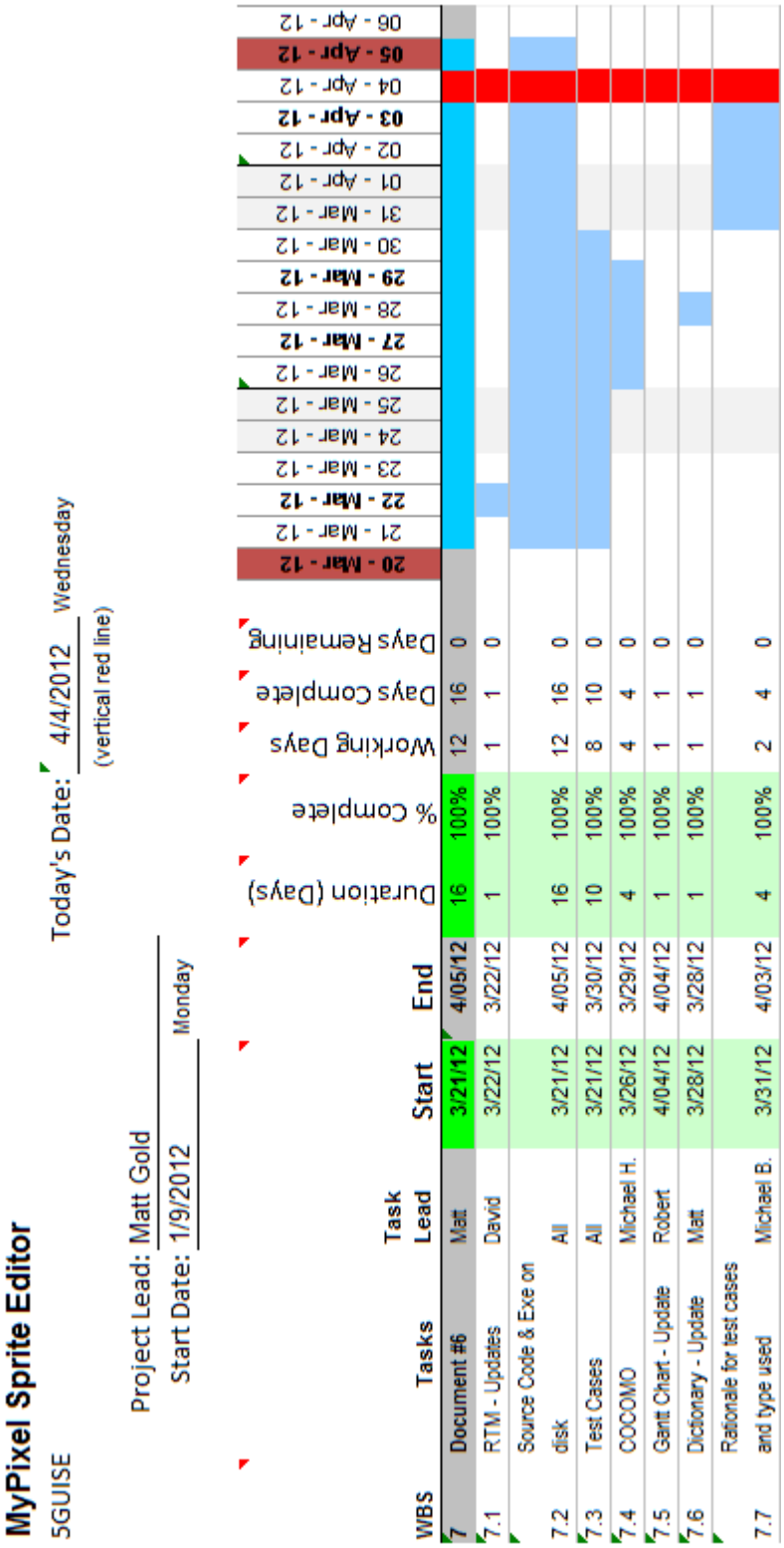
Project Lead: Matt Gold

Start Date: 1/9/2012 Monday

| WBS | Tasks | Task Lead | Start | End | Duration (Days) | % Complete | Working Days | Days Complete | Days Remaining |
|---|---|---|---|---|---|---|---|---|---|
| 6 | Document #5 | Michael H. | 3/09/12 | 3/20/12 | 12 | 100% | 8 | 12 | 0 |
| 6.1 | Project Rationale | Robert | 3/09/12 | 3/16/12 | 8 | 100% | 6 | 8 | 0 |
| 6.2 | RTM - Updates | David | 3/10/12 | 3/10/12 | 1 | 100% | 0 | 1 | 0 |
| 6.3 | WSD - Update | Matt | 3/12/12 | 3/12/12 | 1 | 100% | 1 | 1 | 0 |
| 6.4 | Gantt Chart - Update | Michael H. | 3/18/12 | 3/18/12 | 1 | 100% | 0 | 1 | 0 |
| 6.5 | Dictionary - Update | Michael B. | 3/15/12 | 3/15/12 | 1 | 100% | 1 | 1 | 0 |

08 - Mar - 12 | 09 - Mar - 12 | 10 - Mar - 12 | 11 - Mar - 12 | 12 - Mar - 12 | 13 - Mar - 12 | 14 - Mar - 12 | 15 - Mar - 12 | 16 - Mar - 12 | 17 - Mar - 12 | 18 - Mar - 12 | 19 - Mar - 12 | 20 - Mar - 12

## 2.5 Report 6 Deliverable



**MyPixel Sprite Editor**

5GUISE

Today's Date: 4/4/2012 Wednesday

(vertical red line)

Project Lead: Matt Gold

Start Date: 1/9/2012 Monday

| WBS | Tasks | Task Lead | Start | End | Duration (Days) | % Complete | Working Days | Days Complete | Days Remaining |
|---|---|---|---|---|---|---|---|---|---|
| 7 | Document #6 | Matt | 3/21/12 | 4/05/12 | 16 | 100% | 12 | 16 | 0 |
| 7.1 | RTM - Updates | David | 3/22/12 | 3/22/12 | 1 | 100% | 1 | 1 | 0 |
| 7.2 | Source Code & Exe on disk | All | 3/21/12 | 4/05/12 | 16 | 100% | 12 | 16 | 0 |
| 7.3 | Test Cases | All | 3/21/12 | 3/30/12 | 10 | 100% | 8 | 10 | 0 |
| 7.4 | COCOMO | Michael H. | 3/26/12 | 3/29/12 | 4 | 100% | 4 | 4 | 0 |
| 7.5 | Gantt Chart - Update | Robert | 4/04/12 | 4/04/12 | 1 | 100% | 1 | 1 | 0 |
| 7.6 | Dictionary - Update | Matt | 3/28/12 | 3/28/12 | 1 | 100% | 1 | 1 | 0 |
| 7.7 | Rationale for test cases and type used | Michael B. | 3/31/12 | 4/03/12 | 4 | 100% | 2 | 4 | 0 |

**MyPixel Sprite Editor**

5GUISE

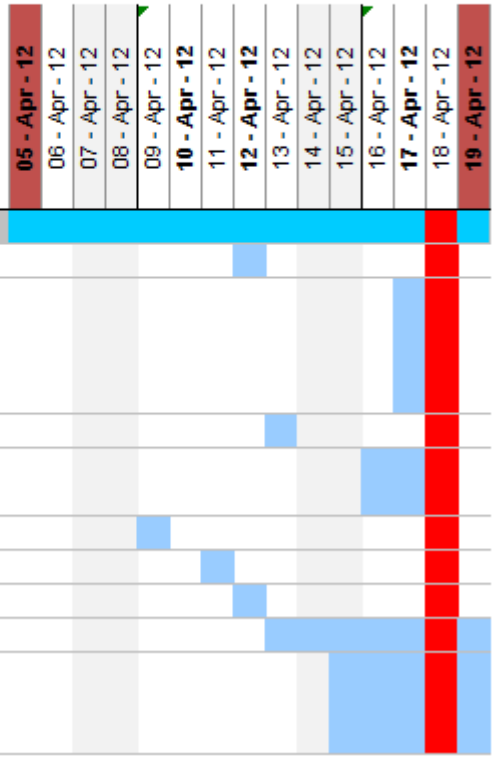
Today's Date: 4/18/2012 Wednesday

(vertical red line)

Project Lead: Matt Gold

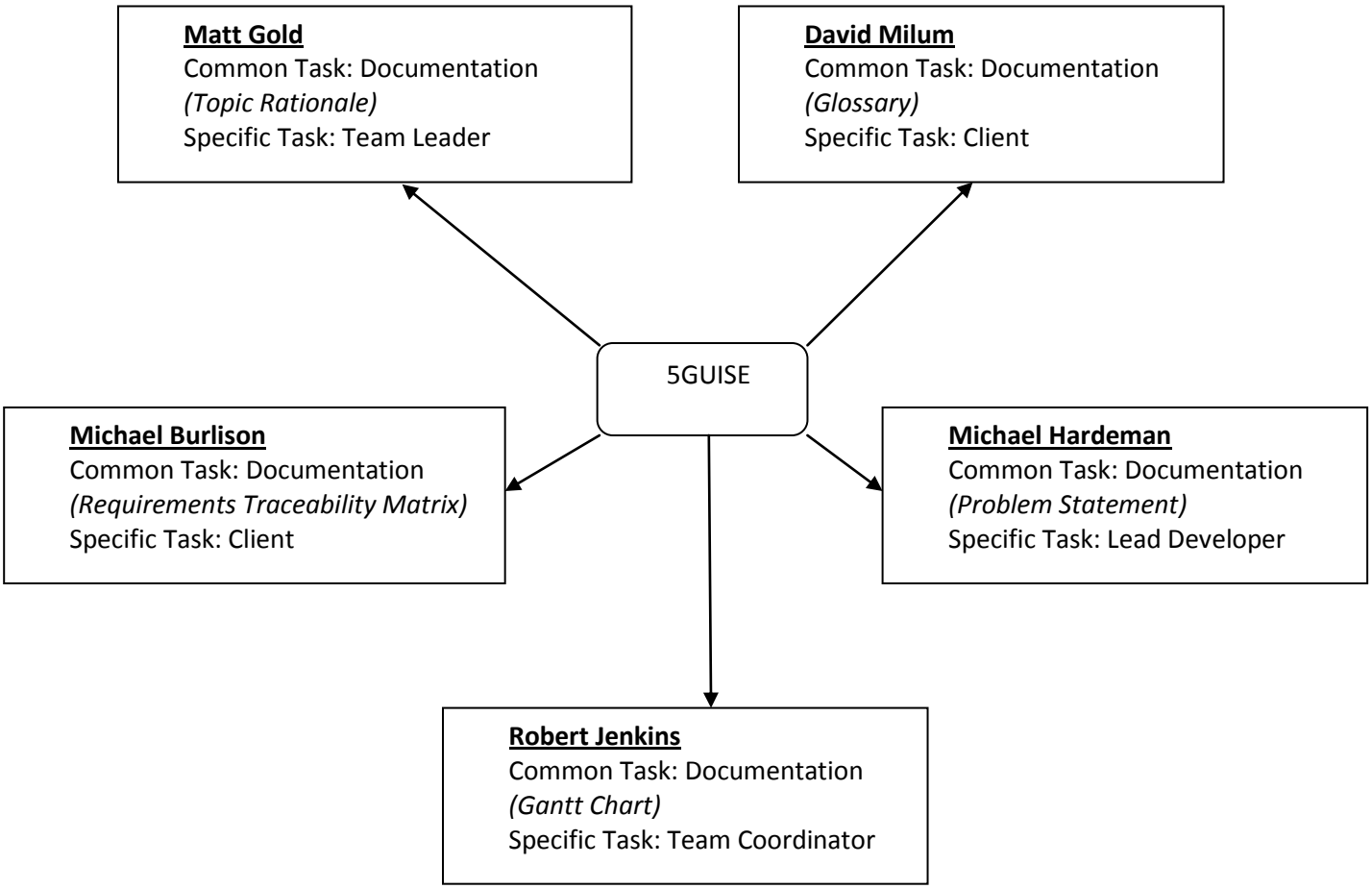
Start Date: 1/9/2012 Monday

| WBS | Tasks | Task Lead | Start | End | Duration (Days) | % Complete | Working Days | Days Complete | Days Remaining |
|---|---|---|---|---|---|---|---|---|---|
| 8 | **Final Document** | Robert | 4/05/12 | 4/19/12 | 15 | 100% | 11 | 15 | 0 |
| 8.1 | Revise COCOMO | Robert | 4/12/12 | 4/12/12 | 1 | 100% | 1 | 1 | 0 |
| 8.2 | COCOMO vs. Functional Point Cost Analysis Comparison and Conclusions | Robert | 4/17/12 | 4/17/12 | 1 | 100% | 1 | 1 | 0 |
| 8.3 | Project Legacy | Michael H. | 4/13/12 | 4/13/12 | 1 | 100% | 1 | 1 | 0 |
| 8.4 | Add Source code to Report | Robert | 4/16/12 | 4/17/12 | 2 | 100% | 2 | 2 | 0 |
| 8.5 | Gantt Chart - Update | Matt | 4/09/12 | 4/09/12 | 1 | 100% | 1 | 1 | 0 |
| 8.6 | WSD - Update | David | 4/11/12 | 4/11/12 | 1 | 100% | 1 | 1 | 0 |
| 8.7 | Dictionary - Update | David | 4/12/12 | 4/12/12 | 1 | 100% | 1 | 1 | 0 |
| 8.8 | User Manual | Michael B. | 4/13/12 | 4/19/12 | 7 | 100% | 5 | 7 | 0 |
| 8.9 | Documentation and Source Code on USB Drive | Robert | 4/15/12 | 4/19/12 | 5 | 100% | 4 | 5 | 0 |

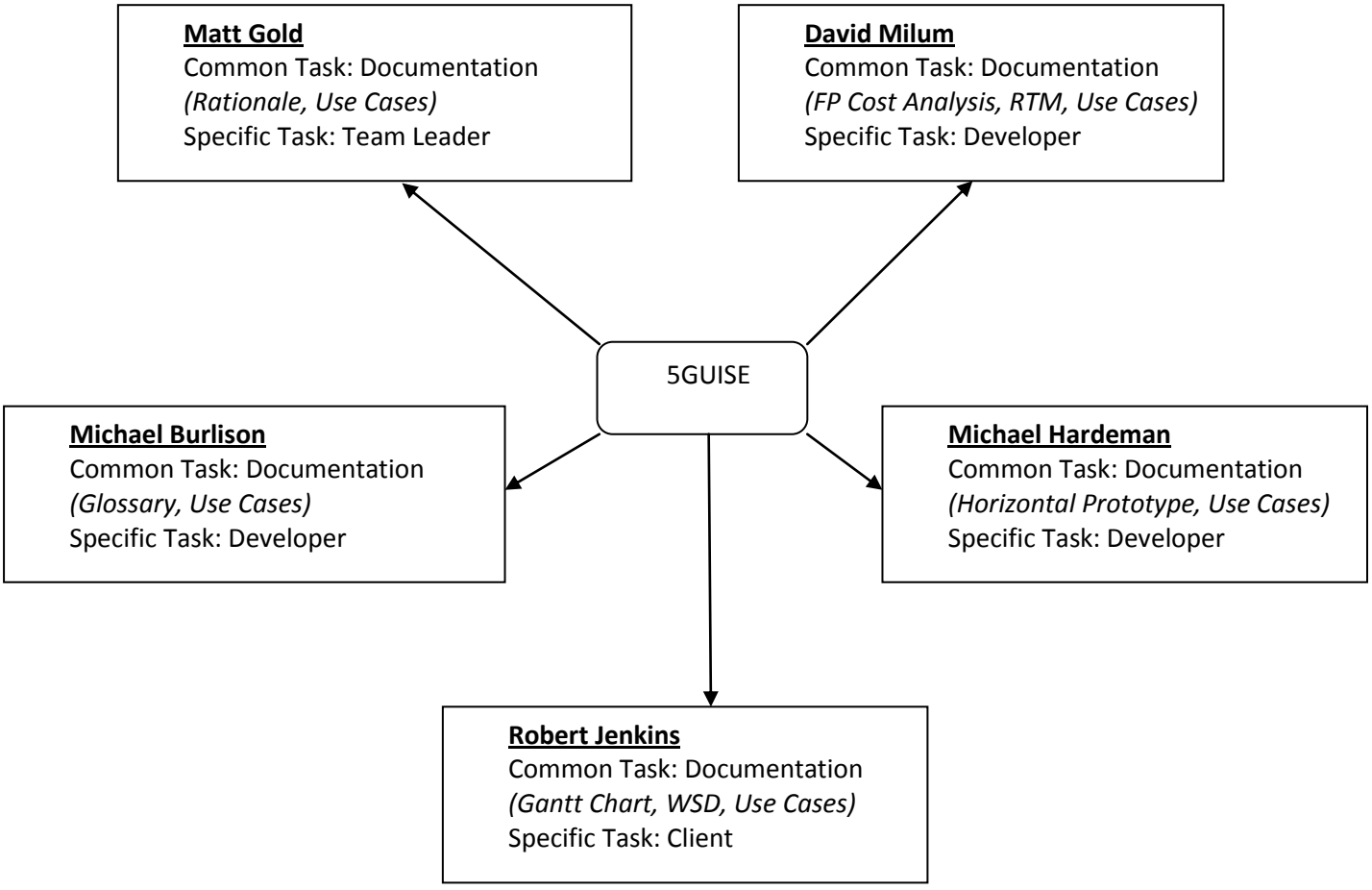


## 3.0 Work Statement Diagram

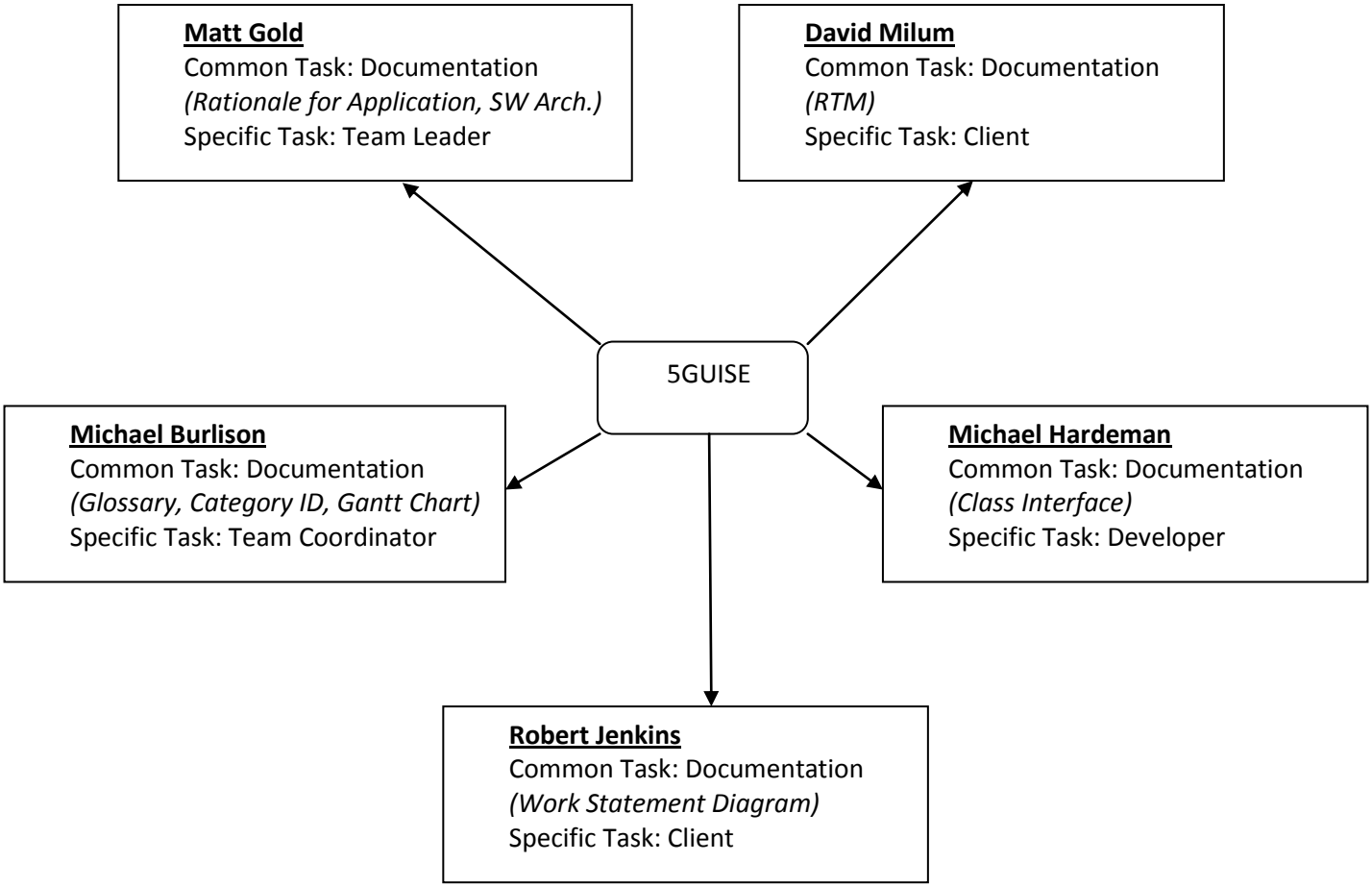
### 3.1 Work Statement Diagram (WSD)



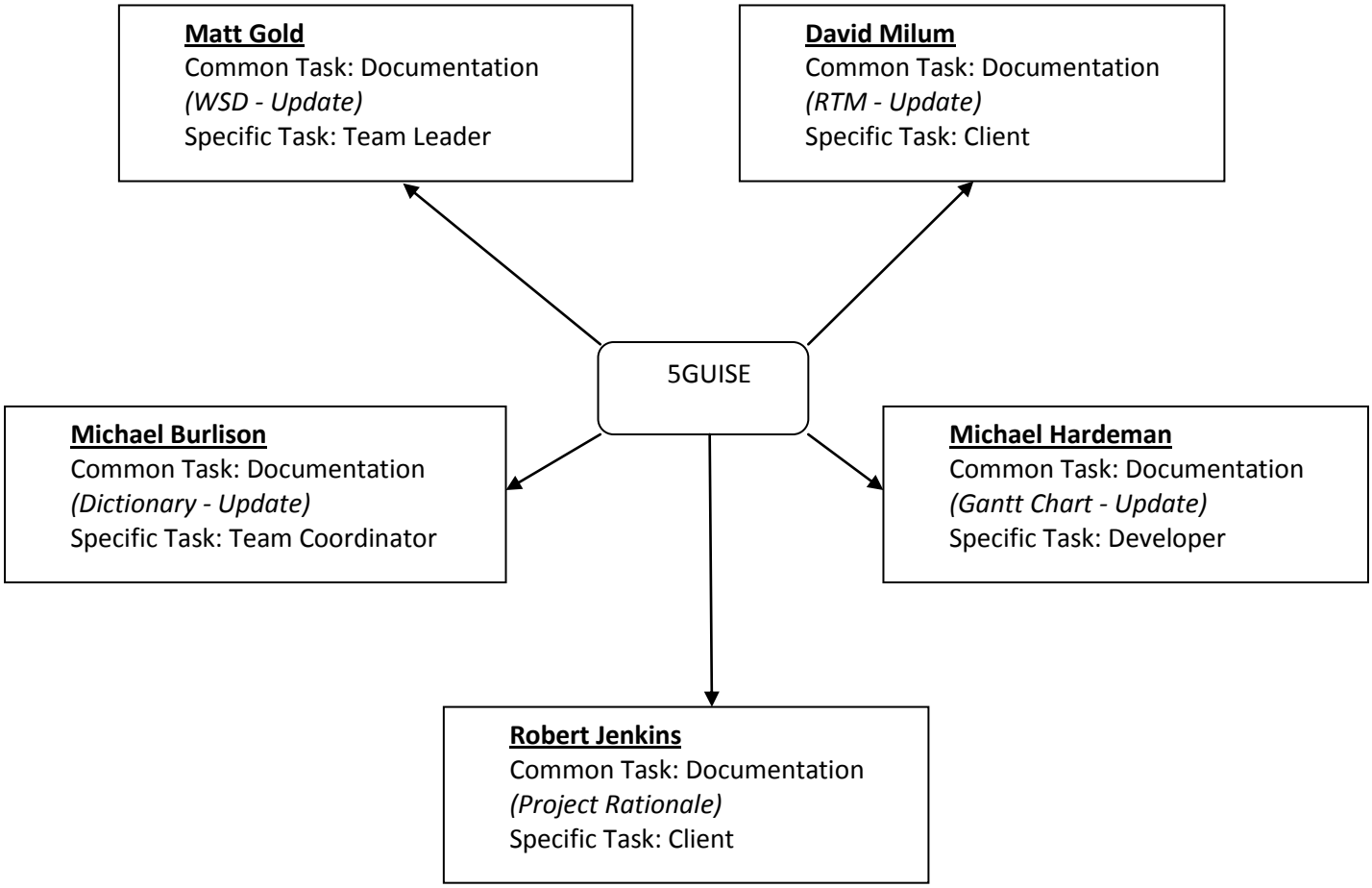
## 3.2 Report 3 Work Statement Diagram (WSD)



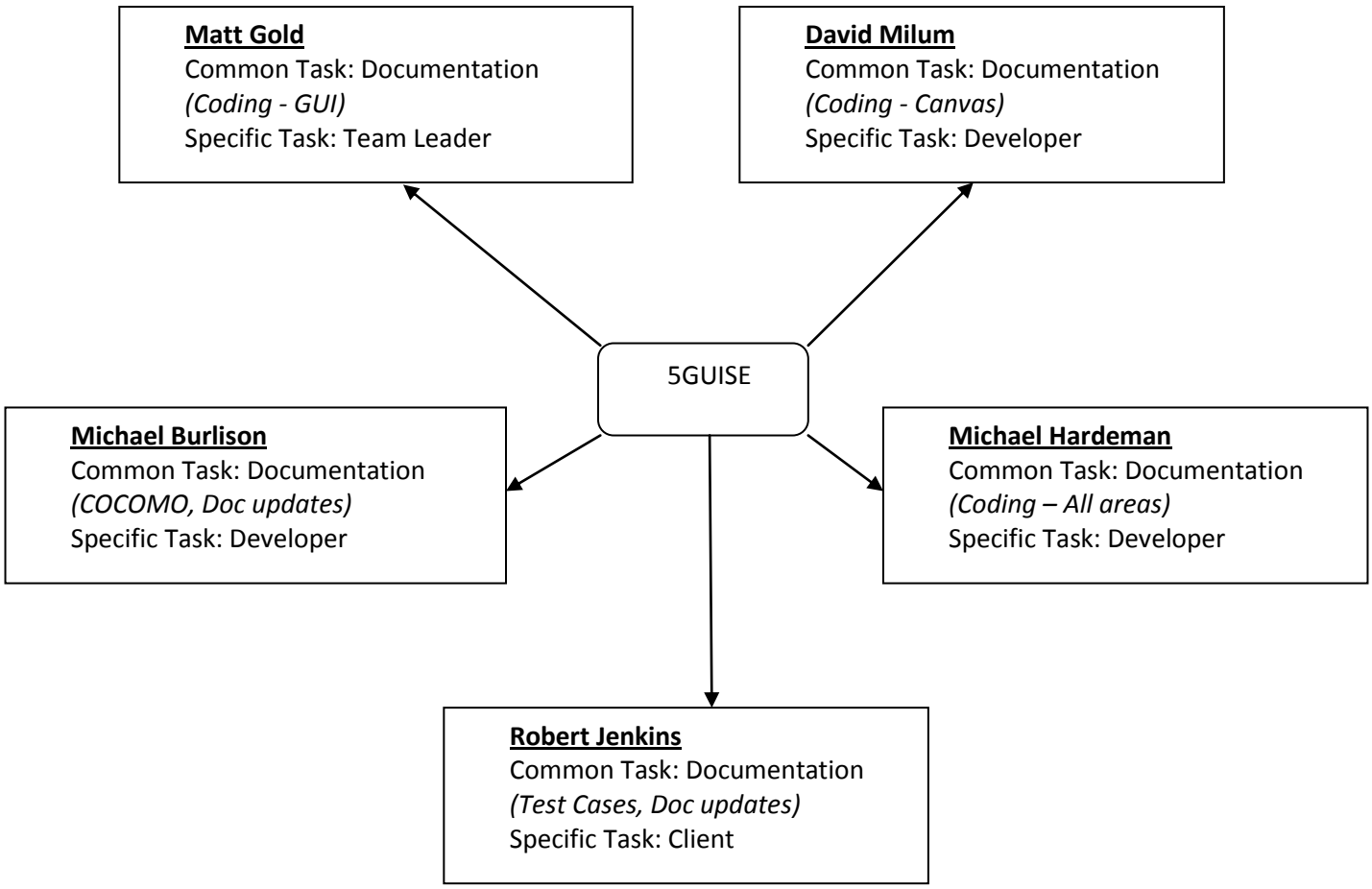
## 3.3 Report 4 Work Statement Diagram (WSD)



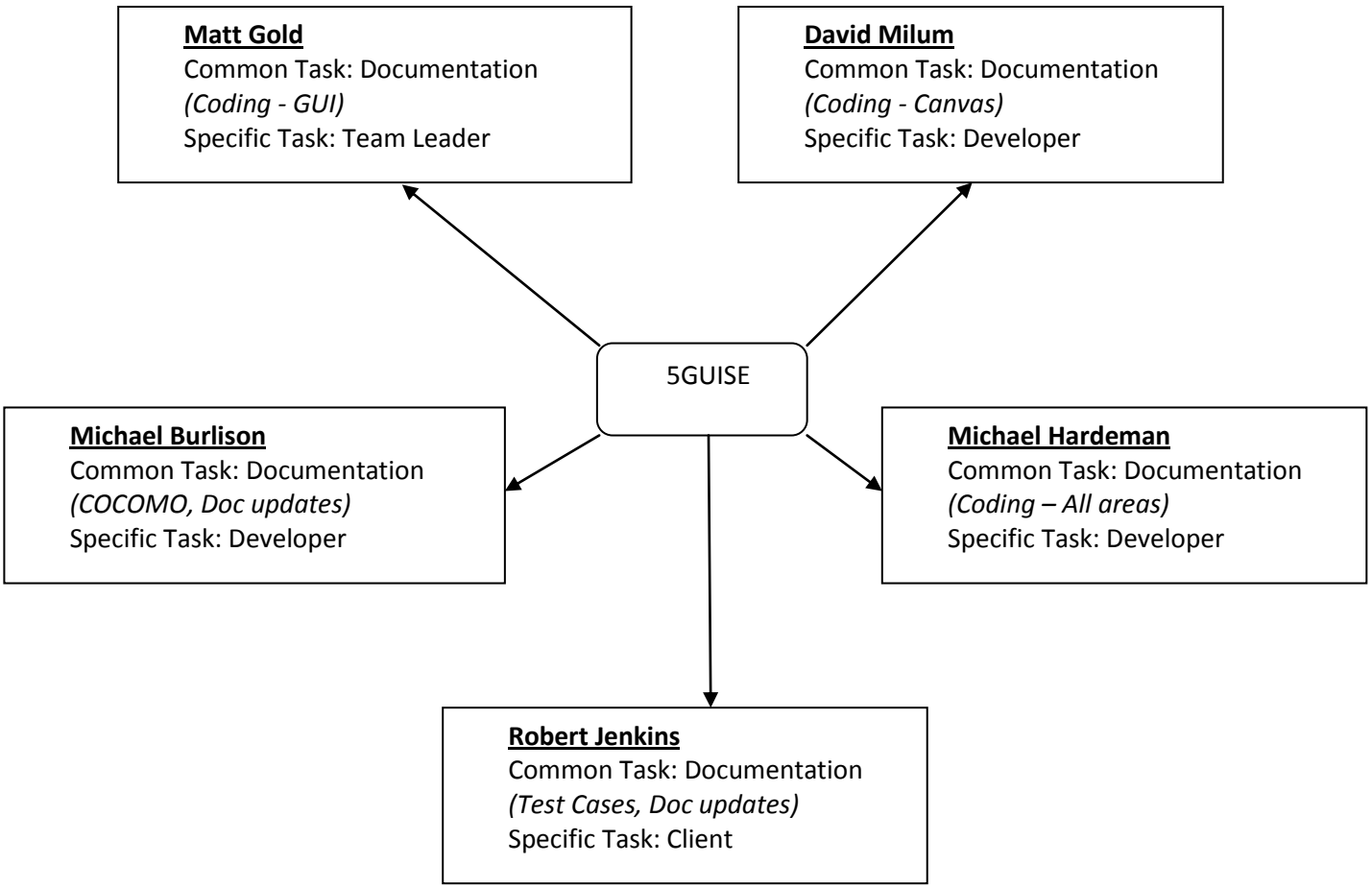
## 3.4 Report 5 Work Statement Diagram (WSD)



## 3.5 Report 6 Work Statement Diagram (WSD)



## 3.6 Final Work Statement Diagram (WSD)



## 4.0 Resumes

This page intentionally left blank

This page intentionally left blank

## 4.1 David Milum

David Milum
1034 Colony Creek Dr.
Lawrenceville, Georgia, 30043
Home: (678)-442-1135
Cell: (404)-838-8519
Email: d.milum@yahoo.com

**Objective**

- Build an understanding of software design and database design principles.
- Get involved in new projects which tackle challenging problems that haven't bee solved.

**Languages Known**

- Java
- C
- SPARC Architecture Assembly
- Python
- UNIX shell scripting, awk, Perl
- MySQL, Oracle SQL, XQuery
- UML

**Experience**

- Java
  - Primary language.
  - Developed software designed to work with databases and networks (socket programming).
  - Able to learn new API's very quickly and easily.
- C
  - Familiar with gcc and gdb.
  - Familiar with system calls and designing C programs for UNIX based filesystems.
- Linux
  - Almost exclusively worked with Ubuntu for a year straight.
  - Installed several distros on various machines, including pendrives.
  - Recently began using Arch Linux.
  - Familiar with most configuration files and many utilities.
  - Written upwards of 50 significant bash scripts.
- Databases
  - Highly proficient with query languages. I'm confident I can solve any meaningful query.
  - DBMS's I've worked with:
    - MySQL, Oracle, db4o (using Java), XML (using XQuery).
  - I've performed complex queries in all of these languages, and designed databases for all of them using ER diagrams.
- General Programming
  - Familiar with a wide variety of algorithms and data structure.
  - Experienced with top-down and object oriented design.
  - Highly motivated problem solver.
  - Readable code.

**Major Programs**

- Wrote a program designed to solve the graph coloring or vertex coloring problem. It successfully colored the 50 contiguous states of the USA using only 4 colors (minimum possible) in a fraction of a second.
- Wrote an SMTP (Simple Mail Transfer Protocol) client in java (gui interface was prebuilt).
- Wrote a frontend for a bookstore database in Java. The program was responsible for interfacing between a customer and a bookstore database on a remote oracle server. Allowed the user to search by various categories, place items, checkout, and save customer information.
- Wrote an oversized bash script (1300+ lines) to manage workout sessions over CLI. User could select the order and pace of the workout, save records,

select weights for different arms, and much more.

**Academic Background**

- Senior at Georgia State University completing a B.S. in Computer Science and a B.A. In Philosophy.

**References**

- Available upon request.

**Hours**

- After Summer: Full time.
- Over the Summer: Part time.
- During the Semester (until April 23rd): Not Available.

## 4.2 Matthew Gold

Matthew B. Gold
2951 Satellite Blvd, Apt 1135
Duluth, GA 30096
Cell: (770) 633-1066
mgold3@student.gsu.edu

### OBJECTIVE

Seeking full-time position in multimedia software programming and development.

### EDUCATION

**B.S. Computer Science,** May 2012
Georgia State University, Atlanta, GA
GPA: 3.74

### WORK EXPERIENCE

**InnovatumInc** - Sugar Hill, GA
February 2011 – Present
Software Development and Documentation Intern

- Personally directed and developed two web applications in ASP.NET, C#, and Javascript adhering to customer requirements.
- Performed database administration and development in MSSQL and Oracle on a daily basis.
- Composed software design and validation documents, executed official software validation.
- Worked alongside a team of developers utilizing source control and unit test technologies efficiently to maintain a working code base.

**Publix Supermarkets -**Atlanta/Peachtree City, GA
August 2007 – February 2011Stock Clerk

- Provided exemplary customer service in the store and frequently over the phone
- Utilized teamwork with associates to finish tasks quickly and efficiently

### SKILLS

Experience with:

- Web development in XHTML/CSS, Javascript /JQuery, PHP, XML
- ASP.NET C#, Web Forms and MVC in Visual Studio IDE.
- Mac OSX, UNIX shell scripting
- Database management in SQL Server Management Studio and SQL Developer
- Java and C in an academic environment
- Windows XP, Vista, 7. Microsoft Office Suite
- Programming games in YoyoGamesGameMaker engine – 4 years
- Programming games and handling 3D assets in Unity 3D
- 3D modeling and keyframe animation in Blender.
- Tortoise SVN ,nUnit
- Data visualization with Google Maps API, Google Charts API.
- Various multimedia applications such as Adobe Photoshop, Tableau, Final Cut, Logic

### HONORS

- 1st place at Georgia Fall Game Jam 2011 - Tekno Tank.
- 2nd place at SIEGE 2011 Student Showcase Competition (Games) - Zero Vector.
- Front-Page feature at yoyogames.com (official GameMaker site) – Air Hockey Unlimited

This page intentionally left blank

## 4.3 Michael Hardeman

Michael Hardeman,
2548, Woodwardia Road,
NE Atlanta, Georgia, 30345,
Home: (404)-636-9742.
Cell: (404)-775-2025.
Email: Mhardeman25@gmail.com

**Objective**

- Gain experience working on large software projects with experienced programmers.
- Better my understanding of Software Design and Production.

**Languages Known**

- Java
- C/C++
- Ada
- Spark Architecture Assembly
- Python
- Bash/C Shell/Borne Shell/Batch
- PHP with MySQL/CSS/HTML

**Experience**

- Data Structures/Software Engineering Course.
  - Advanced Java
  - UML Software Design
  - Abstract Data Types (Stacks/Queues/Arrays/Linked Lists/Trees)
- Spark Assembly/C course.
  - Worked in the Unix environment through SSH tunnel
  - Compiled with GCC.
  - Debug with GDB
- Unix Shell/C course.
  - Worked in the Unix environment through SSH tunnel.
  - C compiled with GCC.
  - Learned make files.
  - Shell Scripting/Unix Shell Commands.
- Network Admin Internship with Emory Healthcare
  - Worked in a full production enviroment.
  - Maintained workstations.
  - Honed troubleshooting skills.
- High level math and linear algebra courses taken.
- Created and Managed a Website on Apache Local Server written in HTML / CSS / PHP. Server used a MySQL database with login system and file uploading / downloading.
- Knowledge of advanced programming techniques such as Abstraction, Recursion, and OO design.
- Partial subsystem review of Doom3 source code.

**Software Written that I'm proud of**

- Wrote a C webserver/client program. The server took commands in the form of *get <filename>*and returned the file. Both interacted through TCP ports in the Unix Environment.
- Wrote an international airline simulator in java with gui visualizer.
- Wrote a Batch script that checked config files and verified installation over workstations attached to a domain.
- Wrote a simple 2.5d (3d graphics confined to 2d space) arcade shooter in the Unity engine.

**Academic Background**

- Junior, Georgia State University for B.S. in Computer Science.

**References**

- Available upon request.

**Hours**

- Over the Summer: Full time.
- During the Semester (until April 23rd): Part time.
  - Tuesday/Thursday: 12PM – As Needed.
  - Weekends: As Needed.

# Michael Burlison

6137 Lenox Park Cir NE Atlanta, GA 30319
T: (615) 545 7433 E: mburlison@gmail.com

## Objective

To inform classmates in my project group of my skills and experience that may help in our goal of successfully designing and developing a new software program.

## Experience

**United States Air Force**
***Systems Administrator*** **July, 2003 - May, 2007**

Installed and managed info systems for an Air Force base of more than 5,000 people. These systems included:

- Theatre Battle Management System using Oracle SQL
- Microsoft Exchange Server
- Tape Backup System (Unix Environment)
- Blackberry Enterprise Server

**Air National Guard**
***Cyber Systems Operations*** **May, 2007 – Present**

I setup, maintain, and repair computers and user domain accounts using Microsoft SMS, Active Directory, and a combination of remote control software and batch scripting.

**Tennessee Association of Realtors®**
***IT Manager*** **June, 2007 – Present**

- I manage a website (PHP & MySQL) for approximately 24,000 Realtors®.
- I install and maintain 20+ computers in an all Apple environment where command-line management is Unix based.
- I installed and continue to manage an enterprise level, wideband wireless network to cover a three story, ~20,000 square foot building. This wireless network was tied into a gigabit Ethernet network that I also maintain.
- I provide phone support for the Realtors® using the website and remote control support for the Association employees based in Nashville, TN.

## Education & Skills

**B.S. Computer Science** **August, 2007 - Present**

Languages and technologies I am familiar with:

- Java
- Unix Shell
- Web languages (HTML, CSS, PHP, Javascript)
- Query languages for Oracle and MySQL
- OS X and OS X Server 10.4 – 10.7
- Windows 2000, XP, Vista, 7 & Windows Server
- Multimedia applications including Photoshop and Vegas

This page intentionally left blank

# Robert S. Jenkins

45 James Court
Hoschton, GA 30548
H: (706)654-9046
C: (770)317-3924
rs.jenkins@me.com

## OBJECTIVE

Pursuing a career with an organization that will leverage my strong troubleshooting background and provide opportunity for advancement upon completing my degree in Computer Science.

## SUMMARY

Engineering minded professional and supervisor with seventeen years of combined experience in the electronic and fiber optic industries. Thorough knowledge of fiber optic and electronic assembly, including troubleshooting to the component level. Highly proficient in writing business spreadsheet applications, databases and test station automation applications. Fourth-year student at Georgia State University with a cumulative 3.81 GPA.

## EMPLOYMENT

2000 – Present | Micron Optics, Inc. | Atlanta, GA

**Manager – New Product Introduction**

- Currently serve as an integral part of the engineering design team, assembling and testing fiber optic mechanical sensing instrumentation.
- Assemble and document all new prototypes to ease the transition from concept to manufacture ready; such as, providing optical, electrical and mechanical schematics and hands on training to manufacturing personnel.
- Provides on the spot manufacturing support, including test station automation by writing LabVIEW software applications and assisting in troubleshooting defective hardware.

1999 – 2000 | Sigma Diagnostics | St. Louis, MO

**Field Engineer**

- Southeast US service representative responsible for routine maintenance and repair of hospital blood analyzers.
- Provided over-the-phone and on-site repair of critical equipment to reduce overall downtime.
- Responsible for training and demonstration of equipment at trade shows.

1993 – 1999 | United States Navy | Norfolk, VA

**Electronic / Radar System Technician**

- Performed repairs on analog, digital, mechanical and electrical components of a technologically advanced radar system.
- As the senior micro-miniature technician, was accountable for troubleshooting to the component level, soldering new components onto circuit cards and testing repairs for quality assurance.
- Calibration technician: calibrated gauges, meters and switches to manufacturer's specifications.

## EDUCATION

Associate of Science, Computer Science | Georgia Perimeter College, 2008

## REFERENCES

Available upon request.

## 5.0 Project Rationale

### 5.1 Topic Rationale

With the game industry overtaking the movie industry back in 2007, video games are bigger than ever. However, there is one division of the game industry that has been growing faster than any other. Independent games, or 'indie games' are on the rise. Crucial game creation tools that were once only available to developers funded by big-name publishers are now very affordable. Because of this shift, many developers have started to create games on their own or with a small team, free of the creative shackles that go hand-in-hand with corporate game development.

Independent developers often create two-dimensional games because they simply lack the manpower to create games in full 3D. This is not a bad thing. In fact, gamers have embraced the new wave of 'retro' games that have recently arisen from the indie scene. Although they appear to adhere to the ways of the past, they exhibit modern design and often are among the most innovative of games released each year.

Pixel art is a longstanding trademark of two-dimensional video games, and thus an essential component in many modern indie games. The creation of this art is a unique process that demands specific features that are not at the forefront of heavyweight image editing software such as Adobe Photoshop, and are not present in simpler image editors such as Microsoft Paint. For all of the reasons listed above, we at 5GUISE believe the market for a lightweight image editor tailored for the specific needs of pixel art and sprite animation is growing along with that of independent games. We aim to meet the demand.

## 5.2 Requirements Rationale

| TM Entry | Rationale | Use Case Name |
|---|---|---|
| 1 | Sprite art and animation is the primary function of the software. | |
| 2 | Users are expected to create new assets with the software. | UC2 SW Create_New_Sprite |
| 3 | Users are expected to edit existing or created assets with the software. | UC3 SW Modify_Existing_Sprite |
| 4 | A virtual canvas is used to conform to artists' conditioned expectations and promote usability. | UC4 SW Edit_Sprite |
| 5 | Built-in tools allow users to create assets out-of-the-box. | UC5 SW Use_Built-in_Tools |
| 6 | Support for user-created tools promotes flexibility and custom solutions. | UC6 SW Create_New_Tools |
| 7 | Pixel art is typically displayed on a much larger scale than the native screen resolution. Zooming is a necessary component of the software. | UC7 SW Canvas_Window_Zoom |
| 8 | Layers provide support for a modular workflow and are an essential tool for efficient sprite animation. This feature is a hallmark of professional image editors and will help the software stand apart from competitors. | |
| 9 | Images should be presented to the user in a natural form. | |
| 10 | The user should be able to see the results of their actions on the canvas. The canvas must display the graphical representation of the current image data matrix. | |
| 11 | The user should be able to switch to different images at will without worrying about data loss. The entire animation will be treated as one cohesive item as opposed to each image being portrayed to the user as a separate file. This reinforces the user's perception of the software as a sprite editor rather than a typical image editor. | UC11 SW User_Swtiches_Between_Images |
| 12 | The software will implement a D.O.D. approach by separating the image editing screen from the animation composition screen. The user can switch between the two at will. This separation promotes usability and prevents overcrowding of the interface. | UC12 SW User_Swtiches_Between_Animations |
| 13 | Pixels should have a position, color, and transparency. By definition, sprites are intended to be displayed on top of other graphics. For this reason, transparency is an essential feature of the software. | |
| 14 | The mouse, as an extension of the hand and fingers, is the primary method for the user to affect the canvas state. | UC14 SW Mouse_Based_Tool_Interaction |
| 15 | In the same manner that a traditional artist would use different artistic tools to compose a drawing, the user of the software should be able to change the function of the mouse | UC15 SW Image_Interaction_Via_Tools |

| | | |
|---|---|---|
| | by selecting various mouse-based tools. | |
| 16 | The transparency background image lets the user know that an area is indeed empty instead of painted a solid color. Sometimes a certain color pallet requires different transparent image or color. | UC16 SW Adjust_Layer_Properties |
| 17 | An onscreen pixel grid outline is a necessary feature of a sprite editor. Working with small scale images often requires precise alignment of pixels. The pixel grid color and resolution should be customizable to work with any color pallet. | UC17 SW Adjust_Image_Properties |
| 18 | The user should be able to perceive the images as a cohesive sprite animation on the animation screen. | UC18 SW Animation_Playback |
| 19 | A sprite animation's playback rate can vary depending on how it is implemented in the game engine. Allowing the user to change the playback rate of the animation preview will make it easier for users to create assets adhering to the game requirements. | UC19 SW Animation_Playback_Frame_Rate_Adjustment |
| 20 | A separate set of tools for the animation screen allows the user to perform complex actions on all images at once. | |
| 21 | The canvas state will be affected by a combination of the selected tool and the user's mouse input. | |
| 22 | Pixel selection is a necessary functionality of any image editor. This allows users to perform work on a focused section of the image. | UC22 Change_Selected_Pixels |
| 23 | Modifying color and opacity of pixels is the central action the user will perform to create assets. | UC23 SW Modify_Individual_Pixel_Color_And_Opacity |
| 24 | Groups of pixels defined by pixel selection will be treated as a separate working layer until deselected. This allows users to perform complex actions on focused sections of the image. | UC24 SW Modify_Groups_Pixels_Color_And_Opacity |
| 25 | Users need access to functions that can affect the entire image selection. Mouse based tools alone are not sufficient for an efficient pixel art workflow. Such modifiers include HSV alteration, transformations, and anti-aliasing. | UC25 SW Transform_Sprite |
| 26 | Support for user-created tools promotes flexibility and custom solutions. | |
| 27 | All customizable tools will be mouse-based and positioned in a central location on the GUI to enforce unambiguous structure. | |
| 28 | Color picker interface increases usability. | |
| 29 | Layers should be visible at all times because of their importance in the workflow. Stacking them vertically to indicate their depth can be done in the right toolbar. | |
| 30 | The power of layering functionality comes from the ability to deactivate them, manipulate their depth, change opacity, and merge them together | UC30 SW Modify_Layers |
| 31 | The user should be able to see the working sprite in its native resolution regardless of the canvas zoom level. An image preview will reside in the right toolbar at all times. | |

## 5.3 Software Rationale

PixelEditor is constructed as a closed, three-their architecture made up of Tools, Actions and Sprite. The user interfaces with the application via Tools (interface tier). Tools perform Actions that are sent to the Action Manager as an xml feed. Actions are made of commands, parameters, a previousState, and currentState. The Action Manager will processActions() (application logic tier). The result of the processed actions is then stored in Sprite (storage tier), which is made up of multiple Pixel and Layer instances.

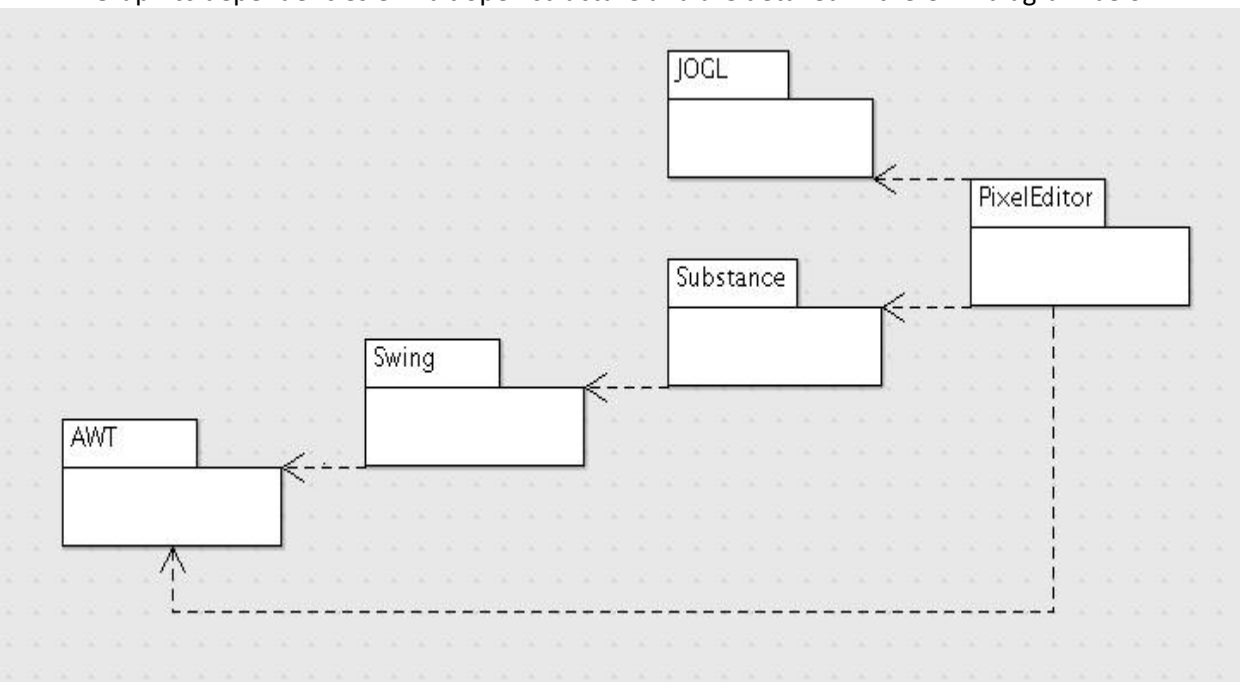
Tools make up the interface tier of PixelEditor. Tools contain command queues which are processed by the ActionManager when notified by the user's mouse input. ToolManager keeps track of the available tools in a toolsHashMap and acts as the bridge to the rest of the system by providing appropriate Getters. Some Tools are directly represented in the UI under the toolbar menu as clickable icons. Modifiers such as image transformations are located in the menu strip, but like the mouse tools, affect the Canvas properties through command queues which are processed by ActionManager.

The ActionManager class handles the application logic of PixelEditor. Incoming actions are added to the actionsToProcess queue, which are pushed onto the actionsDone stack upon completion. When the user wishes to 'undo' an action, the system can move backwards by retrieving the previousState property from the actionsDone stack elements. Actions popped from the actionsDone stack are pushed onto the actionsUndone stack, which will in turn be popped when the user wished to 'redo' actions that were undone. A single Action contains a command string, associated parameters, and the currentState and previousState properties. A command string is an XML representation of Pixels on the canvas whose properties are to be changed. The currentState and previousState properties are both instances of State, which contains a linkedList of changed Pixels.

The LayeredImage class contains an ArrayList of Layers. Each Layer object contains a two-dimensional array of Pixel objects. Each Pixel object has color and opacity properties. The superimposed pixel arrays of the layer objects are a direct representation of the image file that can be exported by the user. When the Action Manager processes actions, it is the properties of the Pixel objects in the currently active layer that are

affected. Thus, the LayeredImage acts as intermediary Storage for the PixelEditor. The Sprite class is used to store multiple instances of the LayeredImage class, and it represents the working sprite animation. When the user switches images, the current LayeredImage properties are transferred to a linked list node in a Sprite object, and the canvas is then emptied. When the user returns to an image to edit, the canvas properties are populated from the appropriate Sprite node. On the animation screen, the user has the ability to delete, duplicate, and reorder the images in the animation, which will manipulate the Sprite nodes accordingly.

Graphics dependencies exhibit open structure and are detailed in the UML diagram below.



## Sprite Art and Animation (SAA)

### ◆ SAA-1.0 Introduction

Sprites are small, square images commonly found in 2D games. The term Sprite also can refer to several images that, when played in sequence, constitute a simple animation. Sprite Art and Animation (SAA) will be produced in our software MyPixel Sprite Editor (MPSE).

### ◆ SAA-2.0 MyPixel Sprite Editor (MPSE)

MPSE shall allow users to create new, and modify existing sprites. Sprite editing shall occur on a canvas that represents the grid of pixels involved, and shall be done using built-in tools, and will let the user define tools (layering of built-in tools). The canvas will have multiple zoom options, and will have several image layers.

### ◆ SAA-2.1 Canvas

MPSE's canvas shall contain, and display data relating to each individual sprite image, and the full sprite animation depending on the currently active screen. It shall allow the user to switch back and forth between viewing/editing each image, or viewing/editing the entire animation.

### ◆ SAA-2.2 Image

The image shall consist of a square matrix of pixels. Each pixel shall contain the following data:

- Current canvas position (x,y coordinates)
- Current color (3D vector 1-255 [rgb value])
- Current transparency (number 1-255)

The user shall be able to interact with and modify the animation on the animation screen using the provided UI elements. The animation shall consist of all images in the sprite and are intended to be played back in an animation preview.

### ◆ SAA-2.3 Animation

The animation shall consist of the multiple images played in sequence at a specified rate. The user shall be allowed to change the rate at which the images are played back. The user shall also have several tools with which to change the animation.

### ◆ SAA-2.4 Tools

The user shall have some basic mouse-based tools allowing him to change the canvas state. Examples include:

- Changing pixel color
- Changing pixel opacity (transparency)
- Selecting groups of pixels
- Modifying the color and opacity of the selected group(s).
- Filling selected pixels with color.

### ◆ SAA-2.5 Modifiers

The user shall have access to modifiers whichcan change the properties of an entire image, selection, or animation. Examples include:

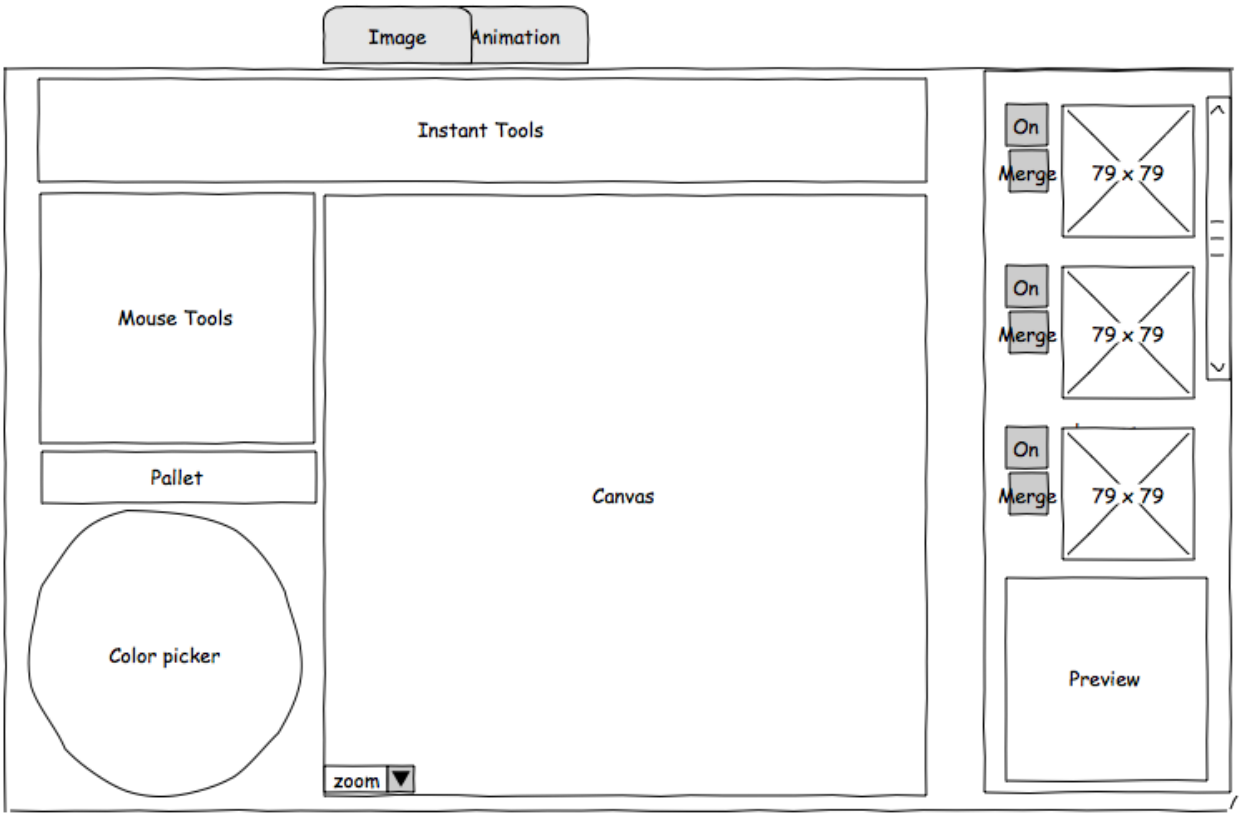
1. Transformations
2. Cropping and resizing
3. Editing color and opacity of groups of pixels

### ◆ SAA-3.0 Left Toolbar

The left toolbar shall contain a panel that houses mouse-based tools. The user will be able to customize the tools. The left toolbar shall also contain a color picker that will allow the user to choose their color pallet.

### ◆ SAA-3.1 Right Toolbar

The right toolbar shall contain a panel that houses the different layers in the image. The user shall be able to add, remove, deactivate, and merge layers from this panel. The right toolbar




shall also contain an image preview of the currently active image in its native resolution.

## 7.0 Requirements Traceability Matrix (RTM)

| Entry # | Para # | System Specification text | Type | Use Case Name |
|---|---|---|---|---|
| 1 | 1.0 | Sprite Art and Animation(SAA) shall be produced in our software MyPixel Sprite Editor (MPSE) | SW | n/a |
| 2 | 2.0 | MPSE shall allow users to create new sprites. | SW | UC2 SW Create_New_Sprite |
| 3 | 2.0 | MPSE shall allow users to modify existing sprites. | SW | UC3 SW Modify_Existing_Sprite |
| 4 | 2.0 | editing shall occur on a canvas that represents the grid of pixels involved | SW | UC4 SW Edit_Sprite |
| 5 | 2.0 | editing shall be done using built-in tools | SW | UC5 SW Use_Built-in_Tools |
| 6 | 2.0 | will let the user define new tools | SW | UC6 SW Create_New_Tools |
| 7 | 2.0 | canvas will have multiple zoom options. | SW | UC7 SW Canvas_Window_Zoom |
| 8 | 2.0 | canvas will have several image layers. | SW | n/a |
| 9 | 2.1 | image shall consist of a square matrix of pixels. | SW | n/a |
| 10 | 2.1 | canvas shall contain, and display data relating to full sprite animation based on the active screen. | SW | n/a |
| 11 | 2.1 | canvas shall allow the user to switch back and forth between viewing/editing each image | SW | UC11 SW User_Swtiches_Between_Images |
| 12 | 2.1 | canvas shall allow the user to switch back and forth between viewing/editing the entire animation. | SW | UC12 SW User_Swtiches_Between_Animations |
| 13 | 2.2 | pixel data shall include canvas position, color, and transparency | SW | n/a |
| 14 | 2.2 | image shall be interacted with via mouse | SW | UC14 SW Mouse_Based_Tool_Interaction |
| 15 | 2.2 | Image shall be interacted with via "tools" | SW | UC15 SW Image_Interaction_Via_Tools |
| 16 | 2.2 | transparency and backrgound image shall be customizable by the user. | SW | UC16 SW Adjust_Layer_Properties |
| 17 | 2.2 | Pixel grid display properties shall be customizable by the user. | SW | UC17 SW Adjust_Image_Properties |
| 18 | 2.3 | Animation shall consist of all images in the sprite and are played in sequence at a specified rate | SW | UC18 SW Animation_Playback |
| 19 | 2.3 | The user shall be able to change the animation rate with an animation preview | SW | UC19 SW Animation_Playback_Frame_Rate_Adjustment |
| 20 | 2.3 | Several built-in tools shall be available for changing the animation | SW | n/a |

| | | | | |
|---|---|---|---|---|
| 21 | 2.4 | Mouse-based tools shall have control over the canvas state | SW | n/a |
| 22 | 2.4 | changes to the canvas state shall include pixel selection | SW | UC22 Change_Selected_Pixels |
| 23 | 2.4 | changes shall include: modifying color and opacity of individual pixels. | SW | UC23 SW Modify_Individual_Pixel_Color_And_Opacity |
| 24 | 2.4 | changes shall include: modifying color and opacity of groups of pixels. | SW | UC24 SW Modify_Groups_Pixels_Color_And_Opacity |
| 25 | 2.5 | users shall have access to modifiers which can change the properties of an entire image, selection, or animation | SW | UC25 SW Transform_Sprite |
| 26 | 3.0 | tools will be able to be customized by the user | SW | n/a |
| 27 | 3.0 | left toolbar shall contain a panel that houses instant tools which are customizable | SW | n/a |
| 28 | 3.0 | left toolbar shall contain a color picker | SW | n/a |
| 29 | 3.1 | right toolbar shall contain a panel that houses the different layers in the image | SW | n/a |
| 30 | 3.1 | right toolbar shall allow the user add, remove, deactivate, and merge layers | SW | UC30 SW Modify_Layers |
| 31 | 3.1 | right toolbar shall contain an image preview of the currently active image in its native resolution | SW | n/a |

## 8.0 Use Cases

### 8.1 Use Case 02: Create_New_Sprite

**Overview:**
This Use Case enables the user to create a new working sprite with a specified file name. The new working sprite consists of one image initially, with a specific width and height in pixels.

**Preconditions:**
1. There is no unsaved worked in the actions stack.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the 'File' button in the menu strip. | 1. Dropdown appears. File option 'New' is visible. |
| 2. User clicks on 'New' button within the File menu dropdown. | 2. 'New Sprite' dialog box appears. |
| 3. Enter values for image width, image height, and file name inputs. | 3. Input fields are updated. |
| 4. User clicks the 'OK' button. | 4. Dialog box disappears. A blank editable canvas with the selected size is visible in the canvas window. New sprite has been created. |
| 5. User clicks the 'Cancel' button. | 5. Dialog box disappears. The new sprite was not created. |

**Scenario Notes:**
Items 4 and 5 are mutually exclusive. Default values will be provided in the image width, image height, and name fields, so item 3 is optional.

**Post Conditions:**
1. A new working sprite was created (if OK button was selected).
2. A blank sprite canvas is visible - The image screen is currently active.

**Exceptions:**
1. Filename already exists.
2. Image size outside bounds.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. 'New Sprite' dialog box pop-up.

**Timing Constraints:**
None.

## 8.2 Use Case 03: Modify_Existing_Sprite

**Overview:**
This describes a case in which the user wants to import, and edit and existing sprite sheet.

**Preconditions:**
1. There is a valid sprite sheet.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the 'File' button in the menu strip. | 1. Dropdown appears. File option 'Import' is visible. |
| 2. User clicks on 'Import' button within the File menu dropdown. | 2. JFileChooser dialogue appears. User browses to the file they want to import. After the file is chosen, the 'Import Sprite Sheet' dialogue appears. Options 'Height', 'Width', 'Coordinates', 'Next/Previous Frame' are visible. An Image preview panel is also visible. |
| 3. User sets the properties of the Sprite within the 'Import Sprite Sheet' dialgue. User may preview the results of their options in the preview panel. User selects ok. | 3. MPSE creates new sprite with the images extracted from the sprite sheet. The first image extracted is set as the selected image. |
| 4. Users may now edit sprite as directed in **Use Case 4.** | 4. **Use Case 4** |

**Scenario Notes:**
Items must be completed as directed.

**Post Conditions:**

1. The sprite has been imported.
2. 'Canvas' state has been set.
3. 'Images' have been populated in the 'Image Selector' on the right.

**Exceptions:**
1. Sprite Sheet is invalid

**Use Cases Utilized:**
4

**Required GUI:**
1. Menu strip
2. Import Sprite Sheet Dialogue
3. Main GUI

**Timing Constraints:**
None.

## 8.3 Use Case 04: Edit_Sprite

**Overview:**
This describes a case in which the user wants to edit a sprite in the 'Canvas'

**Preconditions:**
1. There is a Sprite loaded.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1.Userinteractswithimageviamousetoolfoundinthelefttoolbar. | 1. **Use Case 5** |
| 2.Usermodifiesaselectioninsomeway | 2. **Use Case 22** |
| 3. User adds/deletes/modifies layers in image found in the right toolbar. | 3. **Use Case 30** |
| 4. User selects different image to edit found in the right toolbar. | 4. **Use Case 29** |
| 5. User changes tabs between Image and Animation | 5. **Use Case 11** |
| 6. User modifies Animation FPS | 6. **Use Case 18** |
| 7. User modifies Layer Properties | 7. **Use Case 16** |
| 8. User modifies Image Properties | 8. **Use Case 17** |

**Scenario Notes:**
Item 2 must be completed after Item 1 as Item 2 requires a selection as directed by a mouse tool. Items 1, 3, 4, 5, 6, 7, 8 may occur in any order, any number of times. Item 1, 2, 3, 4, 7, 8, require the Image tab to be selected. Item 6 requires the Animation tab to be selected.

**Post Conditions:**

1. Sprite that has been loaded has been edited.

**Exceptions:**
1. No Sprite has been selected.

**Use Cases Utilized:**
5, 11, 16, 17, 18, 22, 29, 30

**Required GUI:**
1. Main GUI.

**Timing Constraints:**
None.

## 8.4 Use Case 05: Use_Built-in_Tools

**Overview:**
The user will have access to a number of various built-in tools to facilitate mouse-based editing of the image.

**Preconditions:**
1. A canvas must be active for editing

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left clicks the “Tools” button in the menu. | 1. A dropdown menu revealing different available tools appears. Some might be greyed out. |
| 2. User left clicks on an option from the dropdown that isn't greyed out. | 2.The specified tool is engaged and will be used when the mouse interacts with the canvas. |

**Scenario Notes:**
Tools specify different ways of interacting with the canvas. Whenever a tool is selected, a new editing or selection mode is engaged and it will affect the way the user's mouse clicks interact with the canvas.

**Post Conditions:**
1. The editing state will be changed according to the tool selected.

**Exceptions:**
None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. ‘Tools' button

**Timing Constraints:**
None.

## 8.5 Use Case 06: Create_New_Tools

**Overview:**
This describes a case in which the user wants to create new tools.

**Preconditions:**
1. The user wants a new tool to modify sprite data.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1.User createsanew'tool.xml'filein'/MySprite/assets/tools' | 1. On load, Software will read the xml file and populate it's properties based on metadata contained within the xml file. The tool will be added to the Menu Strip > 'Tools'. |
| 2.User uses tool with mouse based interaction. | 2. **Use Case 5** |

**Scenario Notes:**
None

**Post Conditions:**

1. A new tool is created and added to the interface upon restart of the program.

**Exceptions:**
1. 'tool.xml' has invalid metadata

**Use Cases Utilized:**
None.

**Required GUI:**
1. Main GUI
2. Menu Strip

**Timing Constraints:**
None.

## 8.6 Use Case 07:Canvas_Window_Zoom

**Overview:**
This Use Case enables the user to alter the zoom level of the canvas.

**Preconditions:**
1. The image screen is currently active.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the 'View' button in the menu strip. | 1. Dropdown appears. View options 'Zoom In', 'Zoom Out', and 'No Zoom' are visible. |

| | |
|---|---|
| 2. User clicks the 'Zoom In' button. | 2. The canvas view zooms in at a factor of $2^{x+1}$. |
| 3. User clicks the 'Zoom Out' button. | 3. The canvas view zooms out at a factor of $2^{x-1}$. |
| 4. User clicks the 'No Zoom' button. | 4. The canvas view returns to normal (x=1). |

**Scenario Notes:**
Items 2, 3, and 4 must be executed directly after item 1.

**Post Conditions:**
1. The canvas zoom level has been altered.

**Exceptions:**
1. Zoom limit exceeded.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip

**Timing Constraints:**
None.

## 8.7 Use Case 11: User_Swtiches_Between_Images

**Overview:**
The software will support tabbed images, enabling the user to switch between editing them with quickly and with ease.

**Preconditions:**
1. At least one image must be tabbed

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left clicks on the image tab. | 1. Canvas for the currently active image appears, and is available for editing |
| 2. User left clicks a different image from the image selector. | 2.The Canvas now shows the image corresponding to the one clicked. The previous image is now available in the image selector. |

**Scenario Notes:**
If the image tab is already up, clicking the image tab will do nothing.

**Post Conditions:**
1. Canvas displays the image corresponding to the one in clicked in the image selector.
2. The previous image is available for selection in the image selector.

**Exceptions:**
None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Image tab.
2. Image selector.

**Timing Constraints:**
<1/10 sec

## 8.8 Use Case 12: User_Swtiches_Between_Animations

**Overview:**
The software will support tabbed animations, enabling the user to switch between editing them with quickly and with ease.

**Preconditions:**
1. At least one animation must be tabbed

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left clicks on the animation tab. | 1. Under the animation tab, the current animation is played in a loop |
| 2. User left clicks a different animation from the animation selector. | 2. Under the animation tab, a different animation corresponding to the one clicked is played in a loop. The previous animation is now available in the animation selector. |

**Scenario Notes:**
If the animation tab is already up, clicking it will do nothing.

**Post Conditions:**
1. The area under the animation tab will play a loop of the newly selected animation.
2. The previous animation is available for selection in the animation selector.

**Exceptions:**
None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Animation tab.
2. Animation selector.

**Timing Constraints:**
<1/10 sec

## 8.9 Use Case 14: Mouse_Based_Tool_Interaction

**Overview:**
This Use Case enables the user to use a mouse to interact with the working image. The user can use the mouse pointer to apply a selected tool to the working image by clicking and dragging on the area of the image where they desire the tools effects to appear.

**Preconditions:**
1. There is a working image, saved or unsaved, loaded in the canvas.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left-clicks on a mouse-based tool button in the toolbar, or selects a tool from the 'Tools" dropdown menu. | 1. The selected tool's icon in the toolbar is highlighted, and the tool's name in the 'Tools' dropdown menu displays a checkmark beside itself. |
| 2. User left-clicks once on a working image pixel. | 2. The selected tool's effect is applied to the clicked pixel. |
| 3. User left-clicks and drags over multiple working image pixels. | 3. The selected tool's effect is applied to the clicked pixel and every pixel the mouse pointer hovers over before the user releases the left mouse button. |

**Scenario Notes:**
Each mouse-based tool may have additional preconditions and/or exceptions.

**Post Conditions:**
1. The working image displays the effects of the mouse-based tool applied.

**Exceptions:**
1. Filename already exists.
2. Image size outside bounds.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. Toolbar
3. Image Canvas

**Timing Constraints:**
None.

## 8.10 Use Case 15: Image_Interaction_Via_Tools

**Overview:**
An image on the canvas will be interacted with using tools. Individual tools are activated from the menu.

**Preconditions:**
1. Exactly one tool is activated from the menu.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User interacts with the image on the canvas via mouse. | 1. The currently activated tool handles changes made to the image. Depending on the tool activated, and the location clicked, the resulting image will be different. |

**Scenario Notes:**
The currently active tool handles interactions between the user and the image. Any time the user clicks a button on the mouse over the image, the currently active tool is responsible for making the appropriate changes.

**Post Conditions:**
1. The image will change, depending on the location clicked and the tool which is active.

**Exceptions:**
1. No tool currently selected.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Canvas

**Timing Constraints:**
None.

## 8.11 Use Case 16: Adjust_Layer_Properties

**Overview:**
This describes a case in which the user wants to alter the properties of the currently selected layer.

**Preconditions:**
1. The 'Image Tab' is currently active.
2. A 'Layer' is currently active.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the ‘Layer’ button in the menu strip. | 1. Dropdown appears. Layer options 'Properties’ is visible. |
| 2. User clicks the ‘Properties’ button. | 2. 'Layer Properties' dialogue appears. Option |

| | |
|---|---|
| | 'Opacity', is visible |
| 3. User slides 'Opacity' slider to desired measurement. | 3. Opacity property on the current layer is set. Member pixel alpha value are recalculated. |

**Scenario Notes:**
Items must be executed in the order described.
Currently Opacity is the only property. More will be available later in the implementation.

**Post Conditions:**
1. The 'Layer' now has desired Opacity.
2. The 'Pixel' now has appropriately scaled alpha value.

**Exceptions:**
1. None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip.
2. Layer Properties Dialogue.

**Timing Constraints:**
None.

## 8.12 Use Case 17: Adjust_Image_Properties

**Overview:**
This describes a case in which the user wants to alter the properties of the currently selected Image.

**Preconditions:**
1. The 'Image Tab' is currently active.
2. An 'Image' is currently active.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the ‘Image’ button in the menu strip. | 1. Dropdown appears. Image options 'Properties’ is visible. |
| 2. User clicks the ‘Properties’ button. | 2. 'Image Properties' dialogue appears. Option 'Opacity' is visible |
| 3. User slides 'Opacity' slider to desired measurement. | 3. Opacity property on the current Image is set. Member layer alpha values are recalculated. |

**Scenario Notes:**
Items must be executed in the order described.
Currently Opacity is the only property. More will be available later in the implementation.

**Post Conditions:**
1. The 'Image' now has desired Opacity.
2. Each 'Layer' now has appropriately scaled alpha value.

**Exceptions:**
1. None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip.
2. Image Properties Dialogue.

**Timing Constraints:**
None.

## 8.13 Use Case 18: Animation_Playback

**Overview:**
This Use Case enables the user playback all the images in the working sprite in sequence and at a specific rate.

**Preconditions:**
1. There is a working image, saved or unsaved, loaded in the canvas.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left-clicks on the animation tab. | 1. The software changes the display to show the animation window with the images in the current working sprite being played back in sequence at the default or user specified frame rate. |
| 2. User can adjust the frame rate using the frame rate slider to the right of the Animation window. | 2. **USE CASE 10** |

**Scenario Notes:**
If the working sprite does not yet contain any images, the animation will appear blank. If the working sprite contains only one image, or identical images, the animation will appear as a single unmoving image.

**Post Conditions:**
1. The animation window is displayed with the images in the working sprite being played back in sequence at the default or user specified frame rate.

**Exceptions:**
1. Working sprite contains one or less images.

**Use Cases Utilized:**
Use Case 10.

**Required GUI:**
1. Animation Tab
2. Animation Preview Window

**Timing Constraints:**
None.

## 8.14 Use Case 19: Animation_Playback_Frame_Rate_Adjustment

**Overview:**
This Use Case enables the user to adjust the frame rate of the animation. The user-specified frame rate will remain in effect until the application is closed or a different working sprite is loaded.

**Preconditions:**
1. There is a working image, saved or unsaved, loaded in the canvas.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User views the animation window. | 1. **USE CASE 09** |
| 2. User left-clicks and drags the animation frame rate slider to the left to reduce the frame rate and to the right to increase the frame rate. | 2. The software stores and adjusts the frames per second of the working sprite images. The software reflects the frame rate adjustment in the animation preview window. |

**Scenario Notes:**
If the working sprite does not yet contain any images, the animation will appear blank. If the working sprite contains only one image, or identical images, the animation will appear as a single, unmoving image.

**Post Conditions:**
1. The animation window is displayed with the images in the working sprite being played back in sequence at the user specified frame rate.

**Exceptions:**
1. Working sprite contains one or less images.

**Use Cases Utilized:**
09

**Required GUI:**
1. Animation Tab
2. Animation Preview Window
3. Frame Rate Adjustment Slider

**Timing Constraints:**
None.

## 8.15 Use Case 22: Change_Selected_Pixels

**Overview:**

This Use Case enables the user to select a specific group of pixels in the working image and change their state on the canvas using mouse-based or menu-based tools.

**Preconditions:**

1. There is a working image, saved or unsaved, loaded in the canvas.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left-clicks on the 'Pixel Selector' tool in the toolbar or from the 'Tools' dropdown menu. | 1. Mouse pointer becomes a set of crosshairs to facilitate pixel selection. |
| 2. User left-clicks the top left most pixel of the desired group of pixels and drags down and to the right. | 2. A transparent box with a dashed outline appears in which the length of the vertical and horizontal sides are determined by how far down and to the right respectively the user drags the pointer after the initial click. |
| 3. User drags the pointer so that only the desired pixels are contained within the dashed box that appears, then releases the left mouse button. | 3. Upon release of the left mouse button, the software will leave the dashed box surrounding the group of pixels until another selection is made or the pixels are deselected. |
| 4. User clicks on a tool or transformation in the toolbar or the 'Tools' dropdown menu. | 4. The pointer will change from the set of crosshairs to the selected tool's appropriate pointer. |
| 5. User applies transformation or tool action to the working image. | 5. The effects of the transformation or tool action is applied only to the selected pixels of the working image. The animation preview is updated to reflect the changes. |

**Scenario Notes:**

Some transformations or tools require that non-empty pixels be selected.

**Post Conditions:**

1. The selected transformation or tool has been applied only to the selected pixels.

**Exceptions:**

1. Selected pixels are empty & transformation or tool requires non-empty pixels.

**Use Cases Utilized:**

**Required GUI:**

1. Image Tab
2. Mouse Tools
3. Top Menu

**Timing Constraints:**

None.

## 8.16 Use Case 23: Modify_Individual_Pixel_Color_And_Opacity

**Overview:**
Changes to the image shall include: modifying the color and opacity of individual pixels.

**Preconditions:**
1. A single pixel is selected on the image.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on desired pixel to edit color. | 1. Pixel is outlined indicating it's ready to be edited. |
| 2. User clicks on a color from the color pallet. | 2. The outlined pixel changes color to the one chosen by the user. |
| 3. User clicks on a desired pixel to edit opacity. | 3. Pixel is outlined indicating it's ready to be edited. |
| 4. User adjusts the transparency slider. | 4. The outlined pixel changes level of opacity. |

**Scenario Notes:**
Items 1 and 3 are mutually exclusive. The pixel's current values prior to editing are highlighted on the color pallet and transparency slider.

**Post Conditions:**
1. The desired pixel's color and opacity has been changed.

**Exceptions:**
None

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. Image editor canvas

**Timing Constraints:**
None.

## 8.17 Use Case 24: Modify_Groups_Pixels_Color_And_Opacity

**Overview:**
Changes to the image shall include: modifying the color and opacity of groups of pixels.

**Preconditions:**
1. A group of pixels is selected on the image.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on desired pixels to edit | 1. Pixels are outlined indicating they are ready to be |

| | |
|---|---|
| color. | edited. |
| 2. User clicks on a color from the color pallet. | 2. The outlined pixels change color to the one chosen by the user. |
| 3. User clicks on desired pixels to edit opacity. | 3. Pixels are outlined indicating they are ready to be edited. |
| 4. User adjusts the transparency slider. | 4. The outlined pixels opacity level changes. |

**Scenario Notes:**
Items 1 and 3 are mutually exclusive. The pixels' current values prior to editing are highlighted on the color pallet and transparency slider.

**Post Conditions:**
1. The desired pixels' color and opacity has been changed.

**Exceptions:**
None

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. Image editor canvas

**Timing Constraints:**
None.

## 8.18 Use Case 25: Transform_Sprite

**Overview:**
This Use Case enables the user to transform the working image or entire sprite animation. The user will be able to choose between the 'Shift', 'Flip', 'Rotate', or 'Scale' transformation options. Each transformation will have its own dialog box to specify the parameters of the transformation.

**Preconditions:**
1. A editable sprite selection exists. Sprite selection consist of current working space, be it an entire animation, a single image, a single layer, or box-selected section of a layer.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User clicks on the 'Transform' button in the menu strip. | 1. Dropdown appears. Transform options 'Shift', 'Flip', 'Rotate', and 'Scale' are visible. |
| 2. User clicks on desired button within the Transform menu dropdown. | 2. 'Transform' dialog box appears for the selection transformation. |
| 3. Enter integer values for horizontal and vertical shift inputs. | 3. Input fields are updated. Image preview reflects pending transformation. |
| 4. Check any combination of the 'Flip Vertical' and 'Flip Horizontal' checkboxes. | 4. Input fields are updated. Image preview reflects pending transformation. |
| 5. Select 90º, 180º, or 270º from the 'Rotate' | 5. Input fields are updated. Image preview |

| | |
|---|---|
| dropdown, or enter a number into the field. | reflects pending transformation. |
| 6. Enter a percentage in the 'Scale Amount' input. | 6. Input fields are updated. Image preview reflects pending transformation. |
| 7. Check the 'Additive' checkbox. | 7. Animation preview is updated reflecting the pending transformations. |
| 8. User clicks the 'OK' button. | 8. Dialog box disappears. The transformation is applied to the sprite selection. |
| 9. User clicks the 'Cancel' button. | 9. Dialog box disappears. The transformation is not applied. |

**Scenario Notes:**
Item 3 only applies if 'Shift' was selected. Item 4 only applies if 'Flip' was selected. Item 5 only applies if 'Rotate' was selected. Item 6 only applies if 'Scale' was selected. Item 7 applies if sprite selection is the entire animation. Items 8 and 9 are mutually exclusive.

**Post Conditions:**
1. The selected transformation has been applied to the sprite selection. (if OK button was selected).

**Exceptions:**
None.

**Use Cases Utilized:**
None.

**Required GUI:**
1. Menu strip
2. 'Transformation' dialog box pop-up.
3. Image Modification Preview

**Timing Constraints:**
None.

## 8.19 Use Case 30: Modify_Layers

**Overview:**
This Use Case enables the user to add, remove, deactivate, and merge layers using the respective buttons in the right toolbar.

**Preconditions:**
1. There is a working image, saved or unsaved, loaded in the canvas.
2. The user is viewing the Image tab.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User left-clicks on a layer in the right toolbar. | 1. The selected layer is highlighted in the right toolbar to indicate that it is the active working layer. |
| 2. User holds Ctrl while left-clicking multiple layers. | 2. The selected layers are highlighted in the right toolbar to indicate that they are the active working layers. |
| 3. User left-clicks on the 'Add Layer' button in the right toolbar. | 3. A new layer appears below the last layer in the layers list and becomes the active layer. |

| | |
|---|---|
| 4. User left-clicks the 'Remove Layer' button in the right toolbar. | 4. The currently selected layer(s) is/are removed from the layers list and is/are discarded. |
| 5. User left-clicks the 'On/Off' button next to the desired layer in the right toolbar. | 5. The respective layer's visibility is toggled. If toggled off, the layer is no longer visible in the image canvas or animation preview, and the On/Off button displays an 'Off' label. If toggled On, the selected layer becomes visible in the Image canvas and the animation preview, and the On/Off button displays an 'On' label. |
| 6. User left-clicks the 'Merge Layers' button in the right toolbar while multiple layers have been selected. | 6. The currently selected layers are merged into one layer, and that layer becomes the active working layer. |

**Scenario Notes:**

The 'Remove Layer' button's actions are ignored if no layers exist. Multiple layer must be selected in order for the user to utilize the 'Merge Layers' button.

**Post Conditions:**

1. Layer(s) added to the working sprite.
2. Layer(s) removed from working sprite.
3. Layer(s) hidden from working sprite.
4. Selected layers are merged into one layer

**Exceptions:**

1. Selected pixels are empty & transformation or tool requires non-empty pixels.

**Use Cases Utilized:**

**Required GUI:**

1. Image Tab
2. Right Toolbar

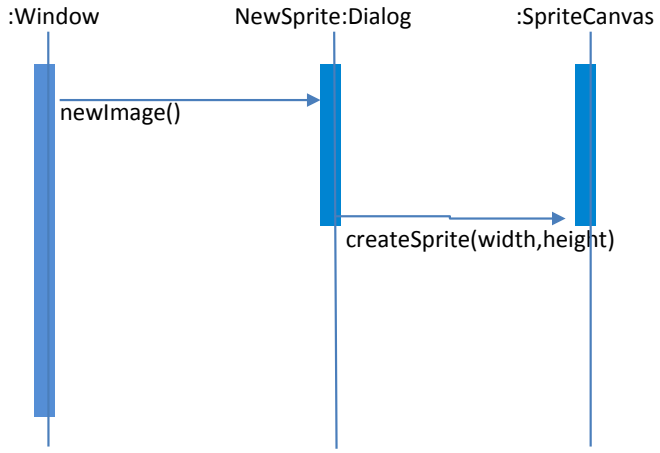
**Timing Constraints:**

None.

## 9.0 Interaction Diagrams

### 9.1 UC02 Create_New_Sprite ID

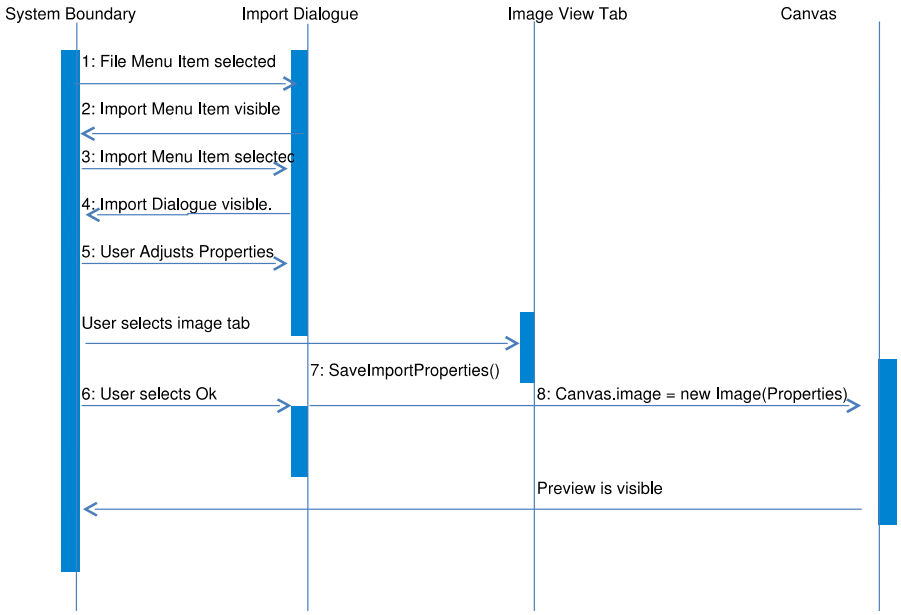
1. User clicks on the File button in the menu strip
2. User clicks on the New button within the file menu dropdown
3. Enter values for image width, image height, and file name.
4. Click 'OK'
5. Click 'Cancel'



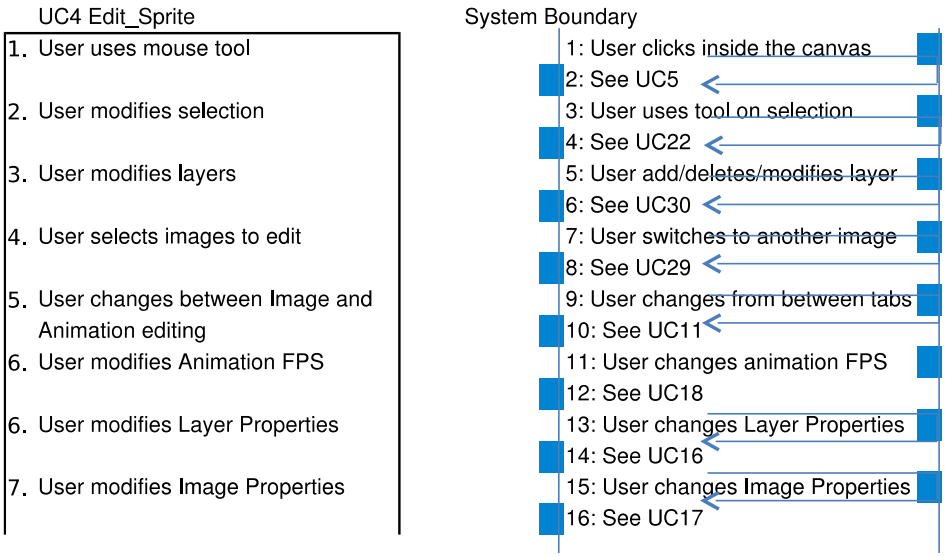

### 9.2 UC03 Modify_Existing_Sprite ID

UC3 Modify_Existing_Sprite

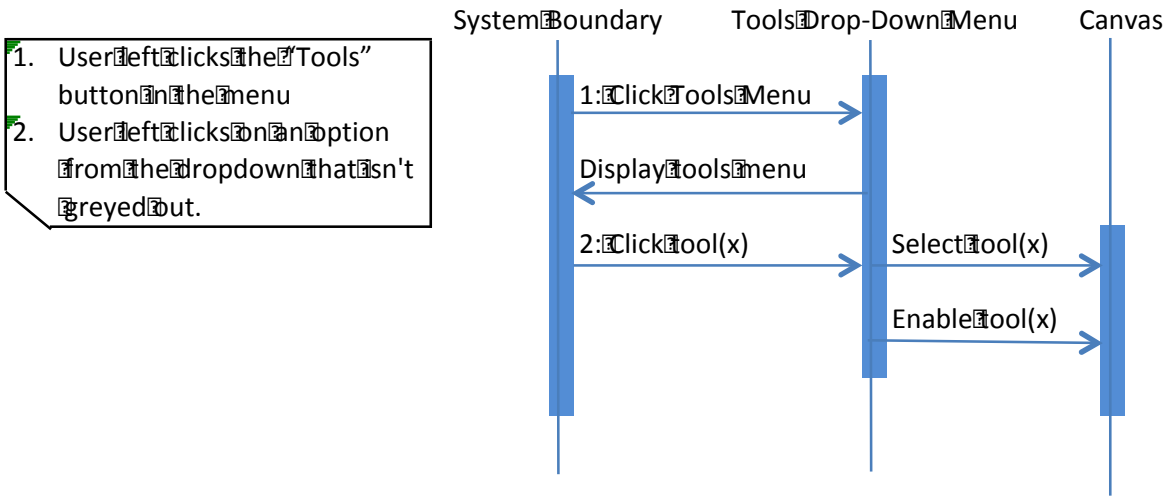
1. User selects the 'File' in the menu strip.
2. Dropdown appears, layer options 'Import' is visible.
3. User selects the 'Import' button.
4. 'Import Sprite Sheet' dialogue appears. Several import options are visible.
5. User adjusts options to the desired levels. A preview is visible showing the final sprite.
6. User selects 'Ok'
7. UC4



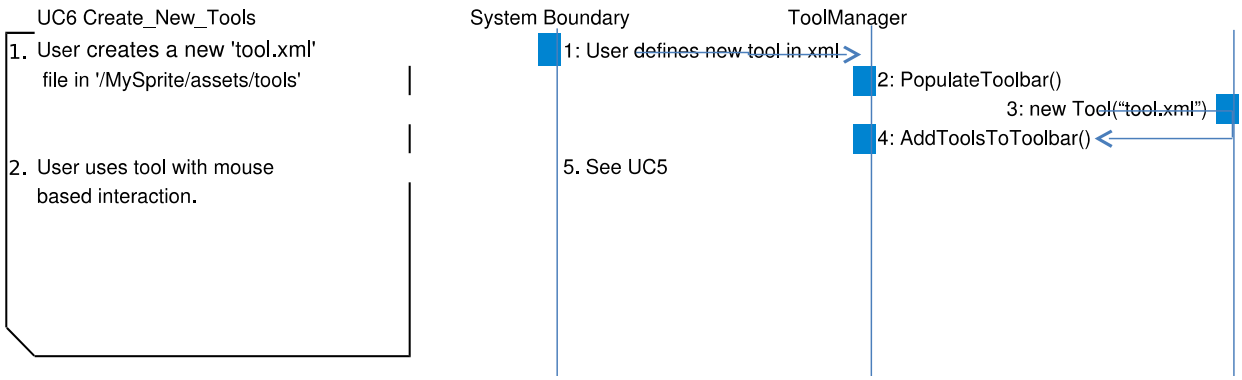
## 9.3 UC04 Edit_Sprite ID



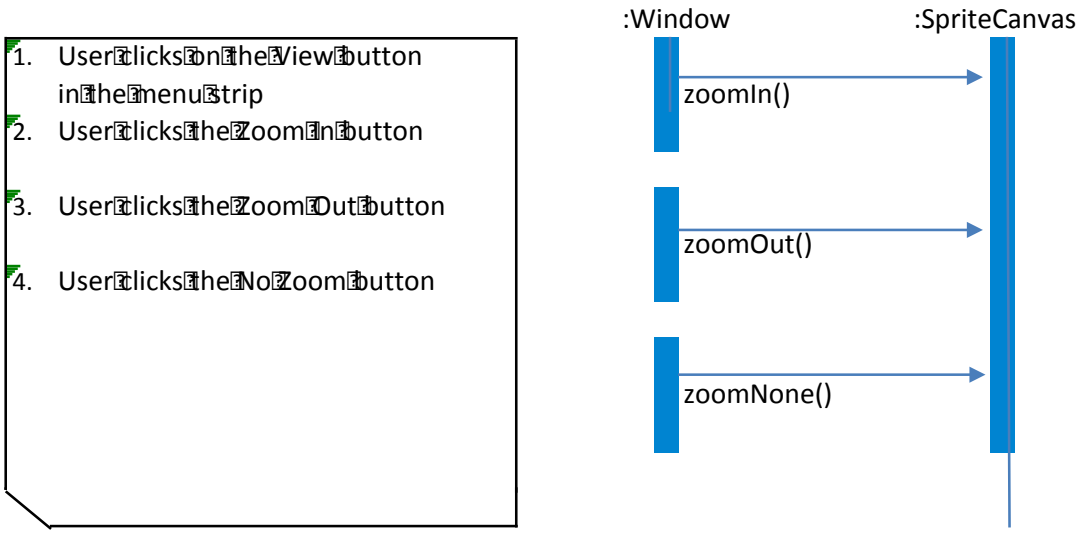

## 9.4UC05 Use_Built-in_Tools ID



## 9.5 UC06 Create_New_Tools

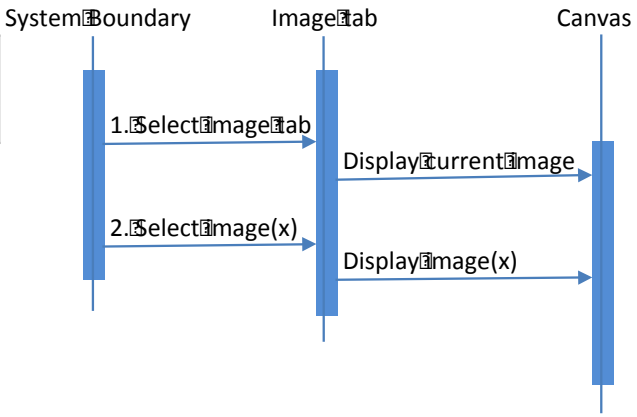



## 9.6 UC07 Canvas_Window_Zoom ID



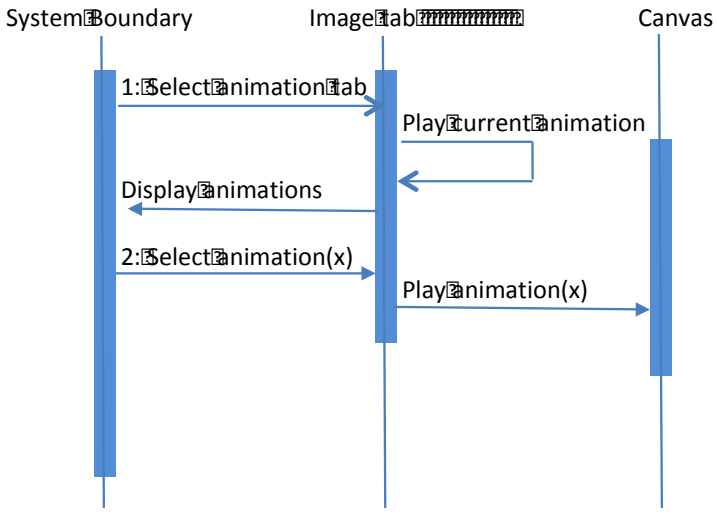
## 9.7 UC11 User_Switches_Between_Images ID

1. User left clicks on the Image tab.
2. User left clicks a different image from the Image selector.



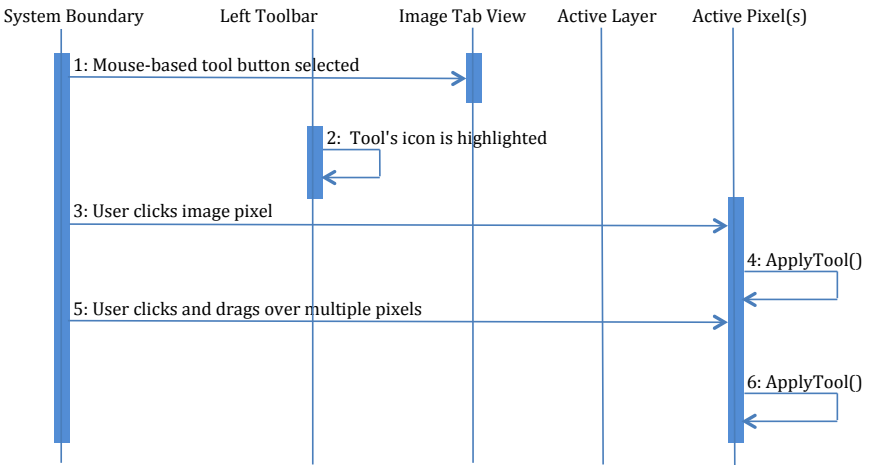
## 9.8 UC12 User_Switches_Between_Animations ID

1. User left clicks on the animation tab.
2. User left clicks a different animation from the animation selector.

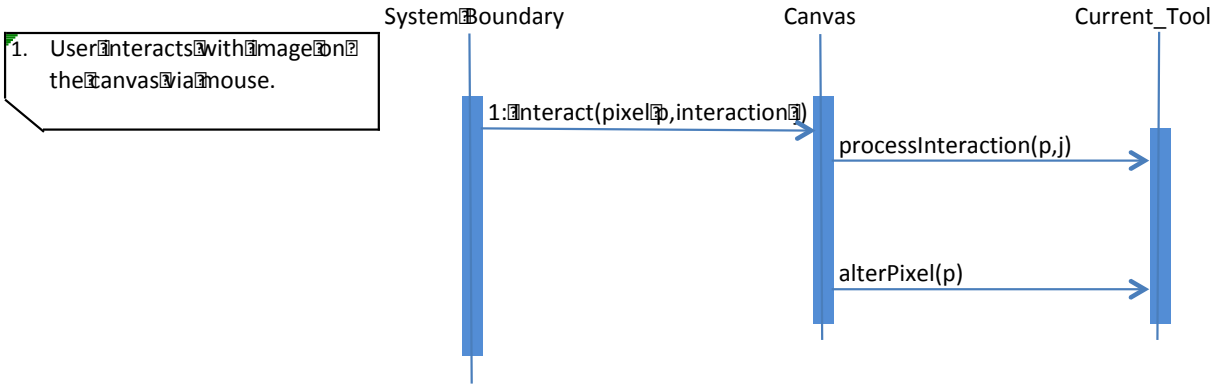



## 9.9 UC14 Mouse_Based_Tool_Interaction ID

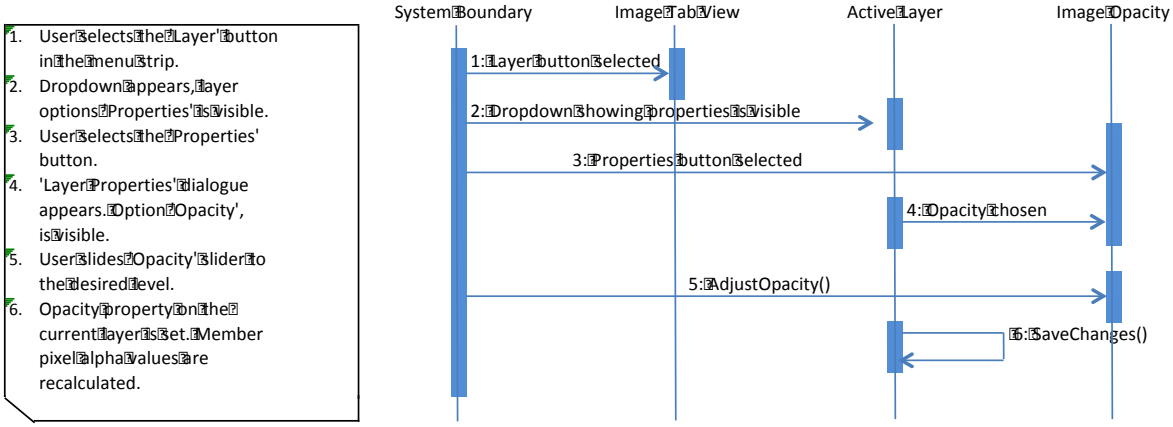
1. User left-clicks on a mouse-based tool button in the toolbar, or selects a tool from the "Tools" dropdown menu.
2. Selected tool's icon in the toolbar is highlighted, and the tool's name in the 'Tools' dropdown menu displays a checkmark beside itself.
3. User left-clicks once on a working image pixel.
4. Selected tool's effect is applied to the clicked pixel
5. User left-clicks and drags over multiple working image pixels.
6. The selected tool's effect is applied to the clicked pixel and every pixel the mouse pointer hovers over before the user releases the left mouse button.



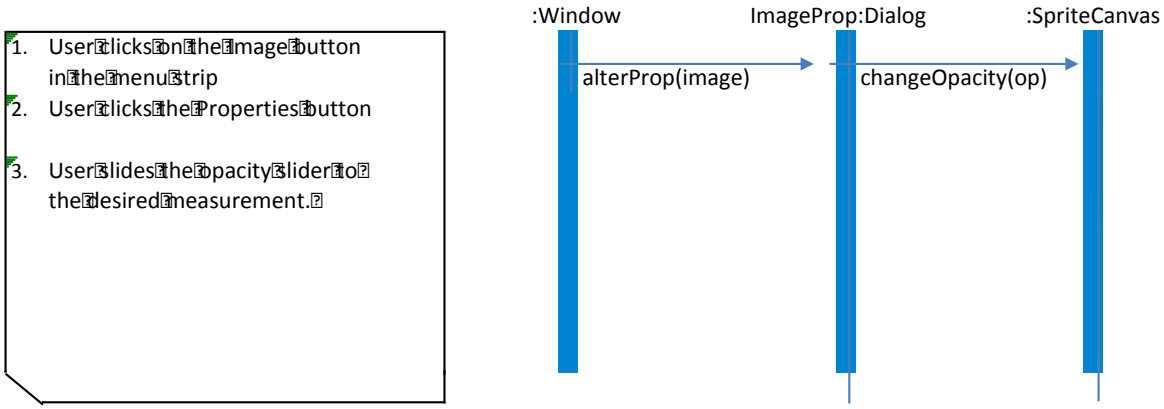
## 9.10 UC15 Image_Image_Interaction_Via_Tools ID



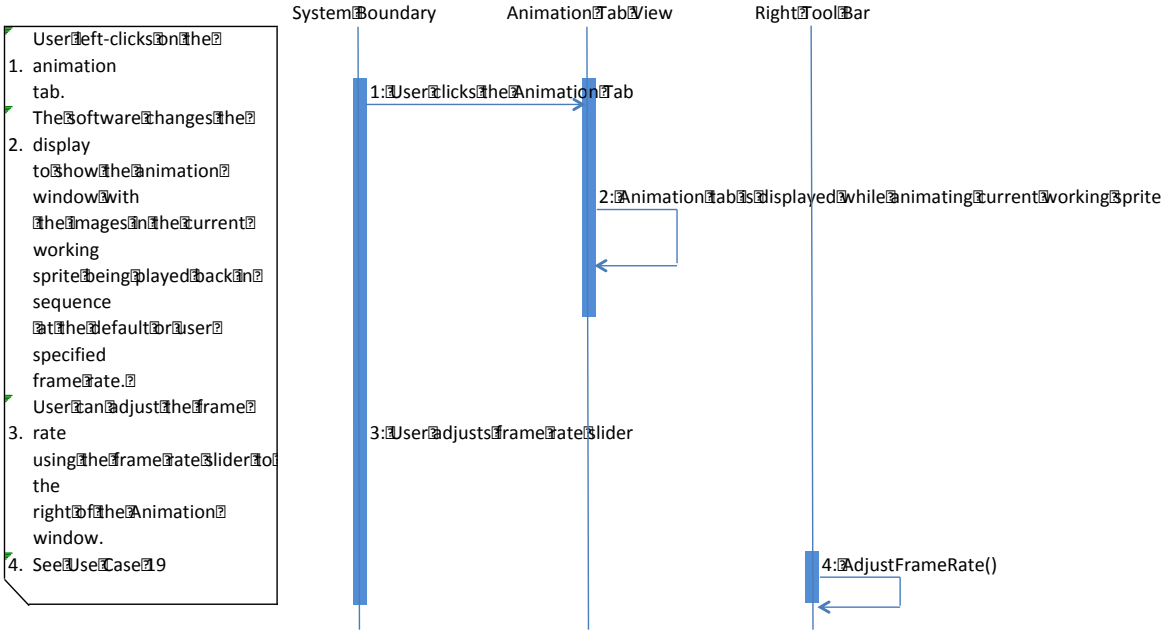

## 9.11 UC16 Adjust_Layer_Properties ID



## 9.12 UC17 Adjust_Image_Properties ID

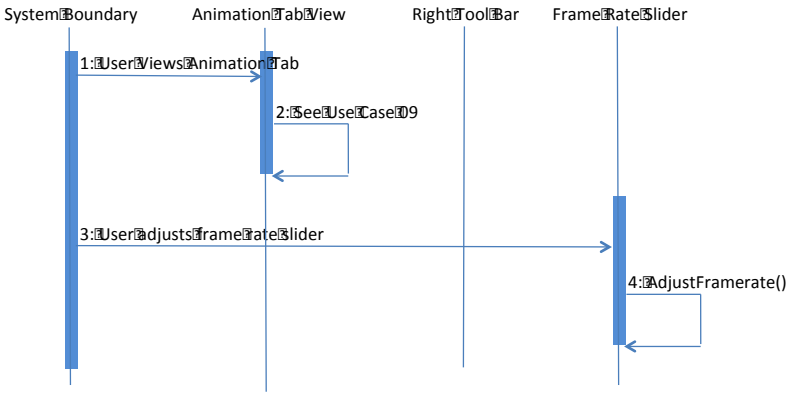



## 9.13 UC18 Adjust_Layer_Properties ID



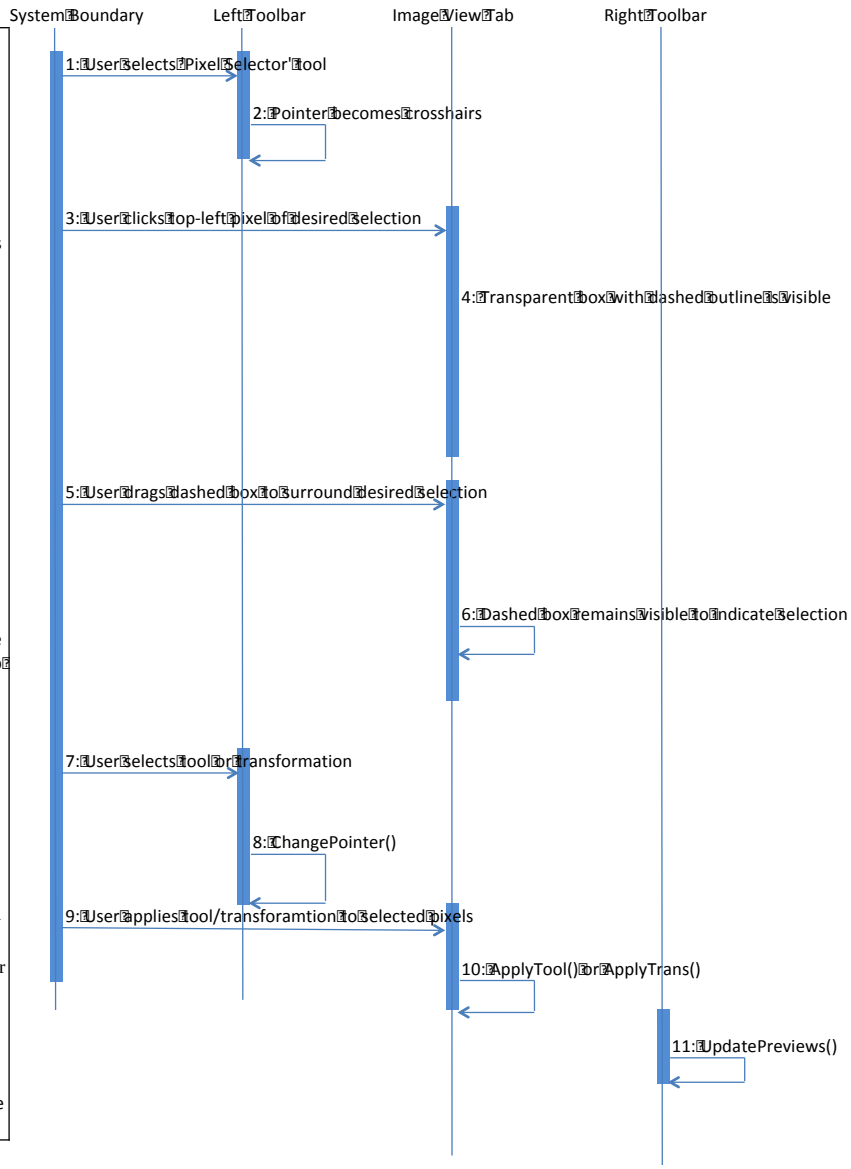
## 9.14 UC19 Animation_Playback_Frame_Rate_Adjustment ID

1. User views the animation window.
2. See Use Case 09
   User left-clicks and drags the animation
3. frame rate
   slider to the left to reduce the frame rate
   and to the
   right to increase the frame rate.
   The software stores and adjusts the frames
4. per
   second of the working sprite images. The
   software
   reflects the frame rate adjustment in the
   animation
   preview window.

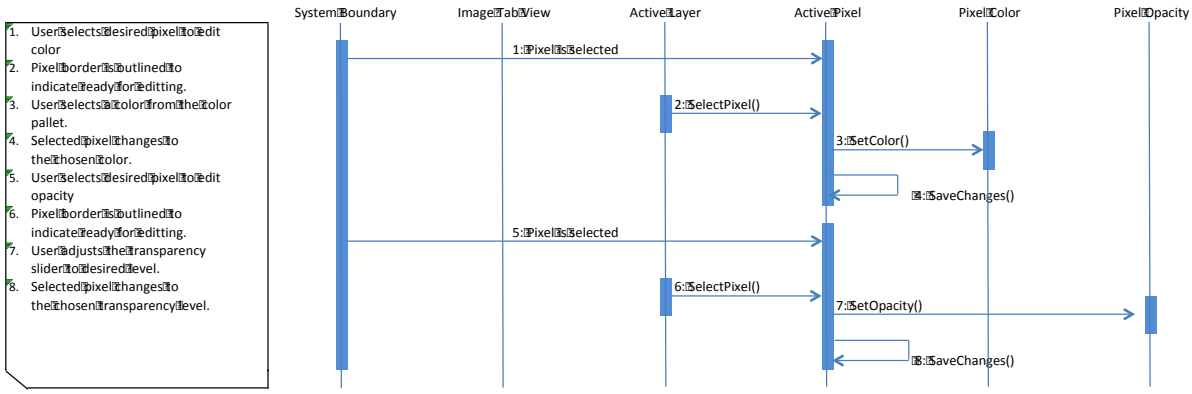


## 9.15 UC22 Change_Selected_Pixels ID

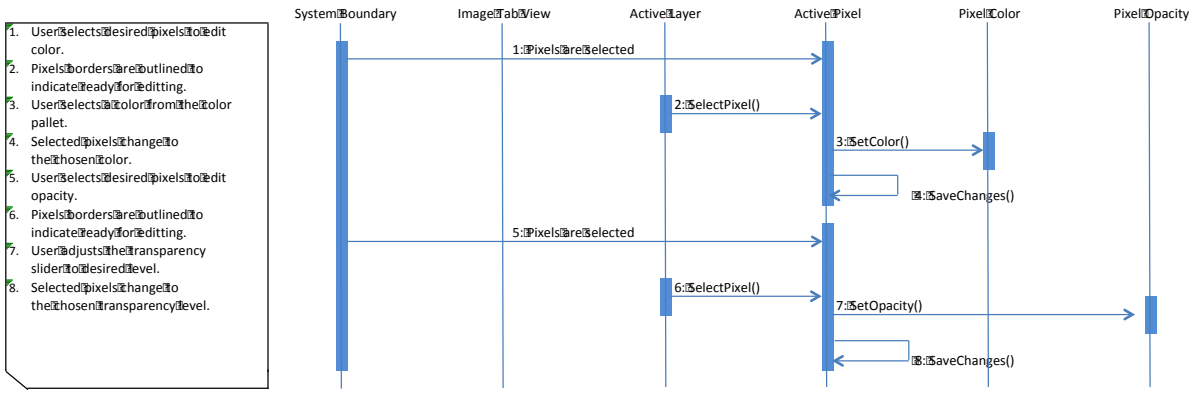
1. User left-clicks on the 'Pixel Selector' tool in the toolbar or from the 'Tools' dropdown menu.
2. Mouse pointer becomes a set of crosshairs to facilitate pixel selection.
3. User left-clicks the top left most pixel of the desired group of pixels and drags down and to the right.
4. A transparent box with a dashed outline appears in which the length of the vertical and horizontal sides are determined by how far down and to the right repectively the user drags. the pointer after the initial click.
5. User drags the pointer so that only the desired pixels are contained within the dashed box that appears, then releases the left mouse button.
6. Upon release of the left mouse button, the software will leave the dashed box surrounding the group of pixels until another selection is made or the pixels are deselected.
7. User clicks on a tool or transformation in the toolbar or the 'Tools' dropdown menu.
8. The pointer will change from the set of crosshairs to the selected tool's appropriate pointer.
9. User applies transformation or tool action to the working image.
10. The effects of the transformation or tool action is applied only to the selected pixels of the working image. The animation preview is updated to reflect the changes.
11. Image and Animation previews are updated.



## 9.16 UC23 Modify_Individual_Pixel_Color_And_Opacity ID

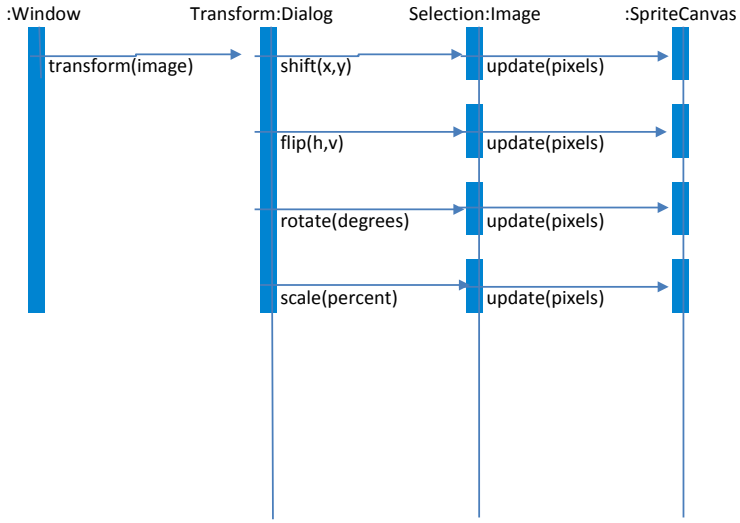



## 9.17 UC24 Modify_Groups_Pixels_Color_And_Opacity ID



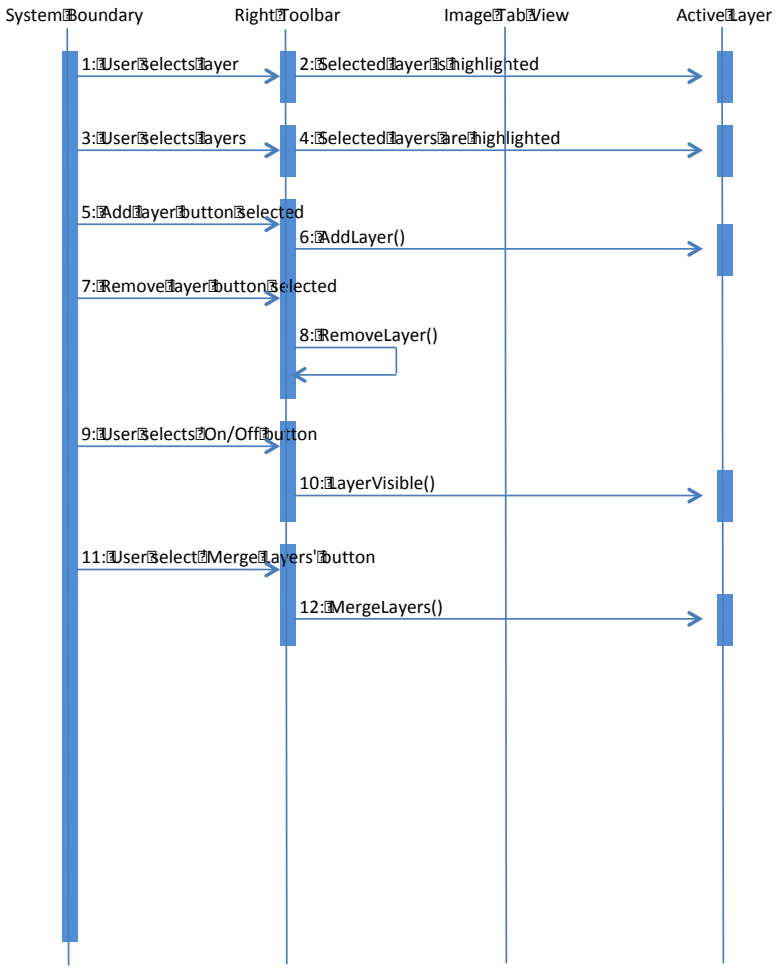
## 9.18 UC25 Transform_Sprite ID

1. User clicks on the Transform button in the menu strip
2. User clicks on the desired button within the transform dropdown
3. Enter integer values for horizontal and vertical shift inputs.
4. Check any combination of the Flip checkboxes.
5. Select 90, 180, or 270 from the Rotate dropdown
6. Enter a percentage in the Scale Amount input.
7. Check the Additive checkbox
8. User clicks the OK button.
9. User clicks the Cancel button.



## 9.19 UC30 Modify_Layers ID

1. User selects a layer in the right-toolbar.
2. The selected layer is highlighted in the right toolbar to indicate it is the active working layer.
3. User selects multiple layers by holding down CTRL while left-clicking.
4. The selected layers are highlighted in the right toolbar to indicate they are the active working layers.
5. User left-clicks on 'Add Layer' button in the right toolbar.
6. A new layer appears below the last layer in the layers list and becomes the active layer.
7. User left-clicks on 'Remove Layer' button in the right toolbar.
8. The currently selected layer(s) is/are removed from the layers list and is/are discarded.
9. User left-clicks the 'On/Off' button next to the desired layer in the right toolbar.
10. The respective layer's visibility is toggled. If toggled off, the layer is no longer visible in the image canvas or animation preview, and the On/Off button displays an Off' label. The opposite if 'On'.
11. User left-clicks the 'Merge Layers' button in the right toolbar while multiple layers are selected.
12. The currently selected layer(s) are merged into one layer, and that layer becomes the active working layer.



## 10.0 Function Point Cost Analysis

1.Doesthesystemrequirereliablebackupandrecovery?

2

2.Aredatacommunicationsrequired?

0

3.Aretheredistributedprocessingfunctions?

0

4.Isperformancecritical?

4

5.Willthesystemberunninginanexisting,heavilyutilizedenvironment?

0

6.Doesthesystemrequireon-linedataentry?

0

7.Doestheonlinedataentryrequire the inputtransactiontobebuiltovermultiplescreensoroperations?

0

8.Arethemasterfilesupdatedon-line?

1

9.Aretheinputs,outputs,files,orinquiriescomplex?

4

10.Istheinternalprocessingcomplex?

4

11.Isthecodedesignedtobereusable?

4

12.Areconversionandinstallationincludedindesign?

1

13.Isthesystemdesignedformultipleinstallationsindifferentorganizations?

5

14.Istheapplicationdesignedtofacilitatechangeandeaseofusebytheuser?

5

Total=30+1.17=31.17

| | | | | | | |
|---|---|---|---|---|---|---|
| Numberofuserinputs | 30 | x | 3 | = | 90 | |
| Numberofusers | 1 | x | 4 | = | 4 | |
| Numberofuserinquiries | 5 | x | 3 | = | 15 | |
| Numberoffiles | 10 | x | 7 | = | 70 | |
| Numberofexternalinterfaces | 1 | x | 5 | = | 5 | |
| **Total** | | | | = | | **184** |

FP=184x(0.65+0.01x31.17)
=**177**

Assume5GUISEproduces6.5FP'spermonth,andlaboris$8000permonth.

Thus1FPis$1230

*Totalcostofthesoftware:*
1230x177=**$217,710**

## 11.0 Horizontal Prototype

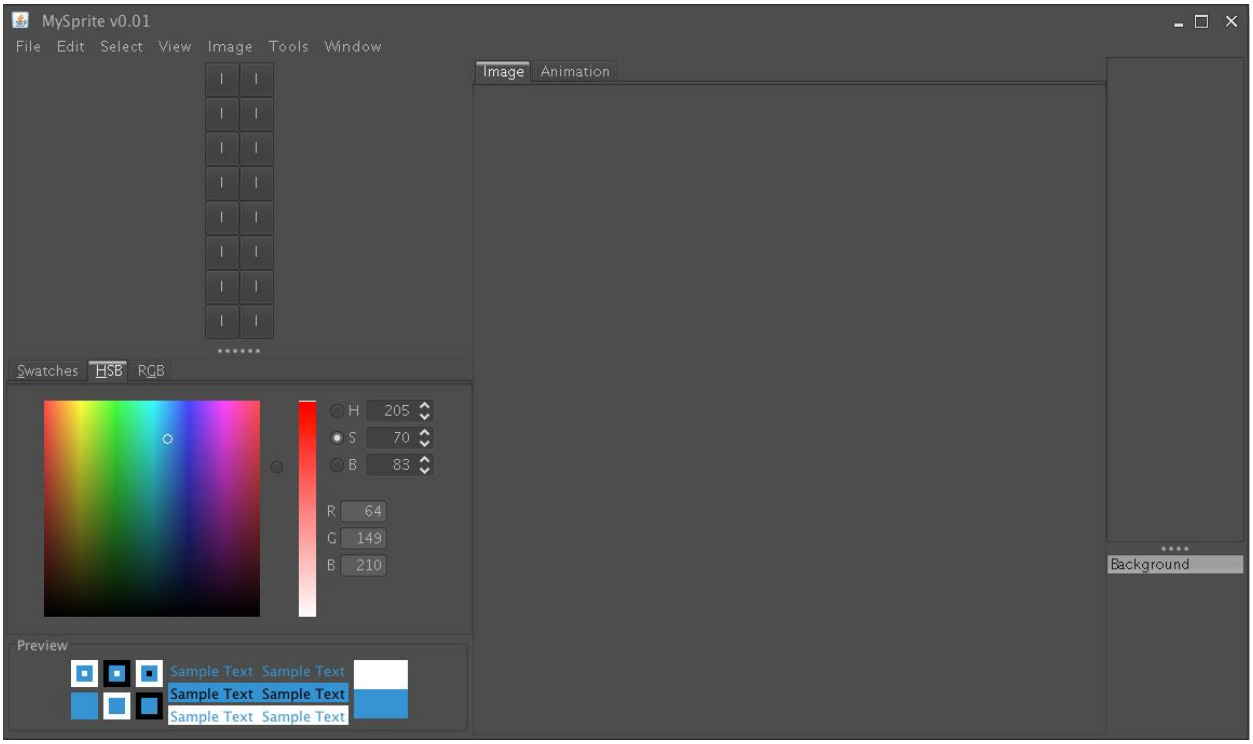



Fig 11.0a – MySprite GUI

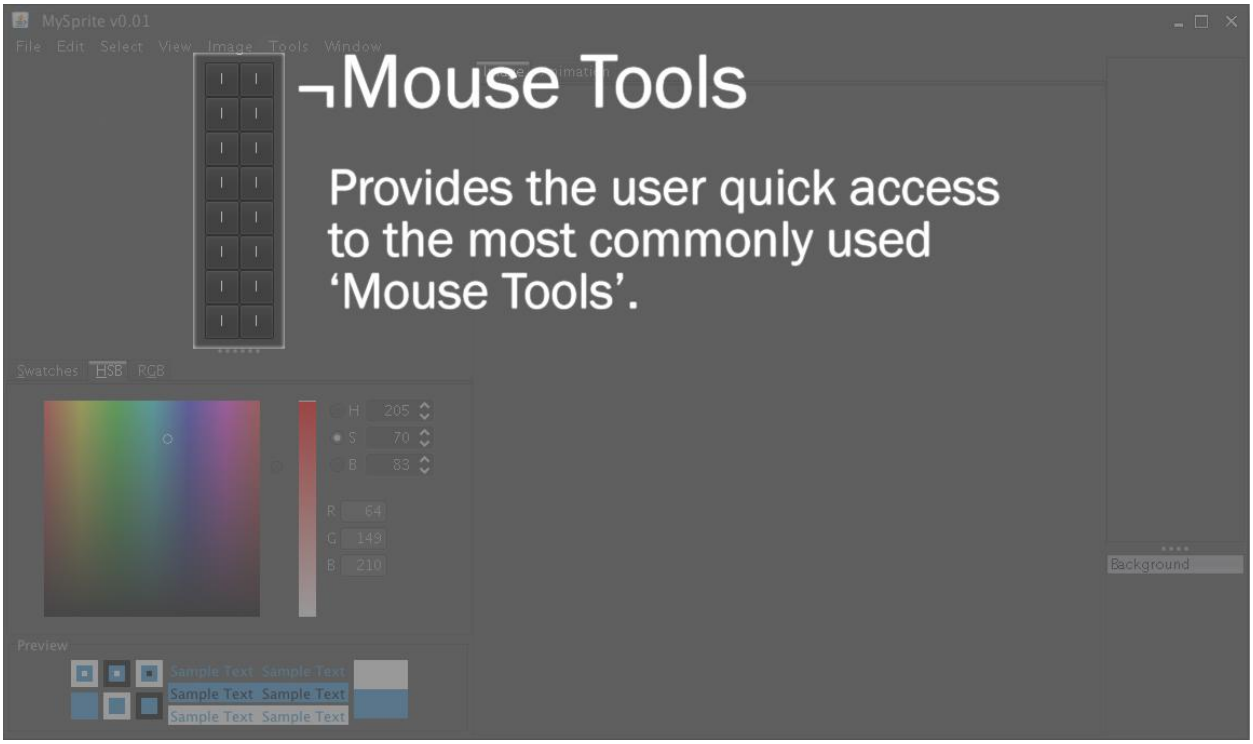



Fig 11.0b – Mouse Tools

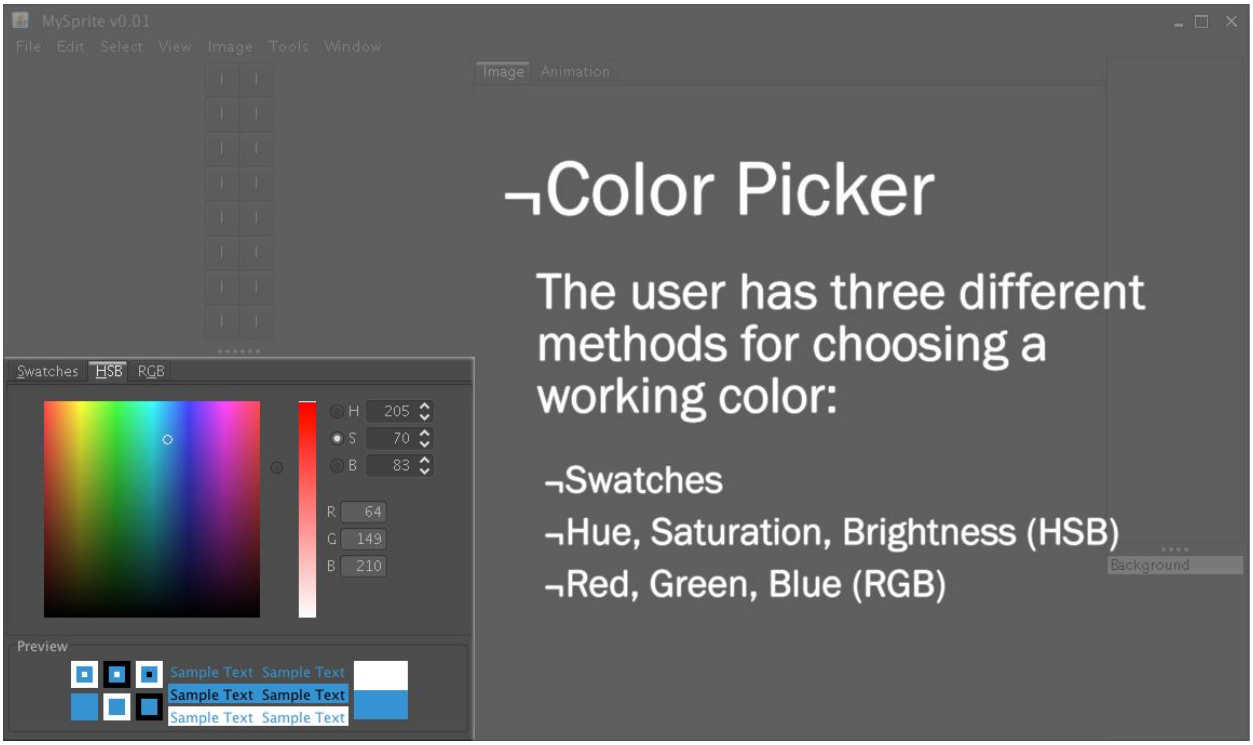



Fig 11.0c – Color Picker

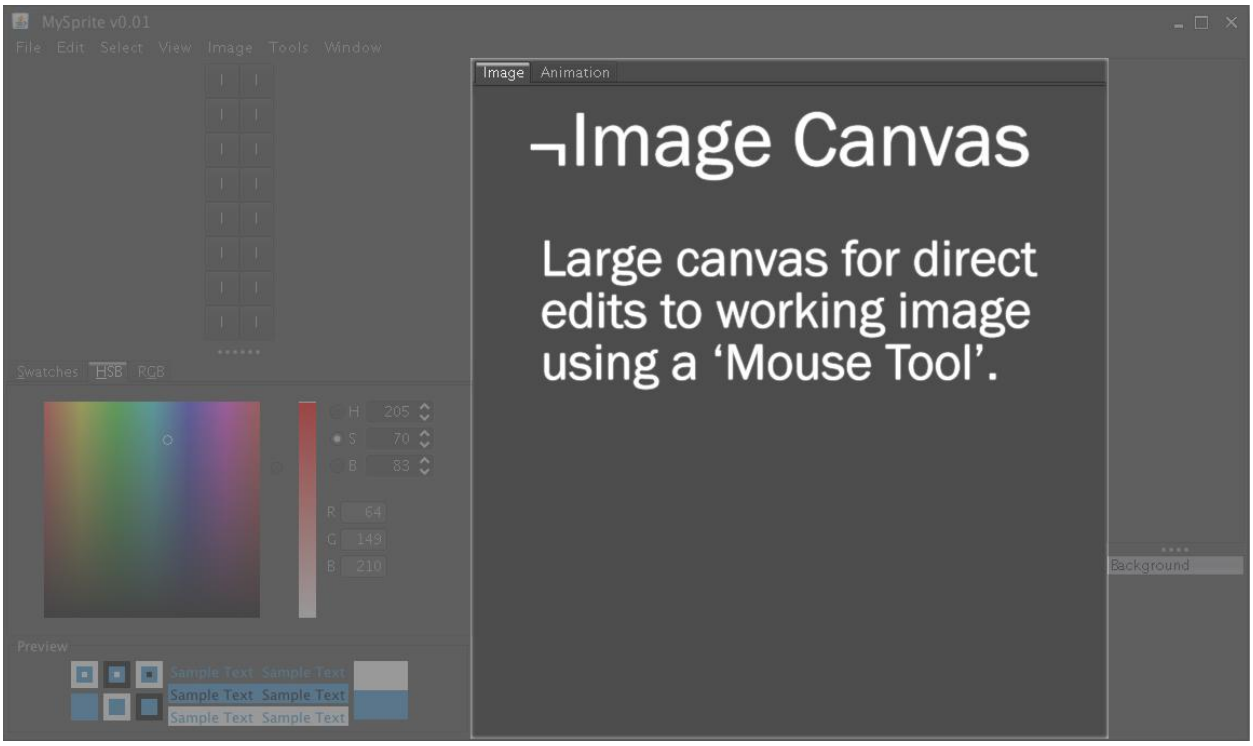



Fig 11.0d – Image Canvas

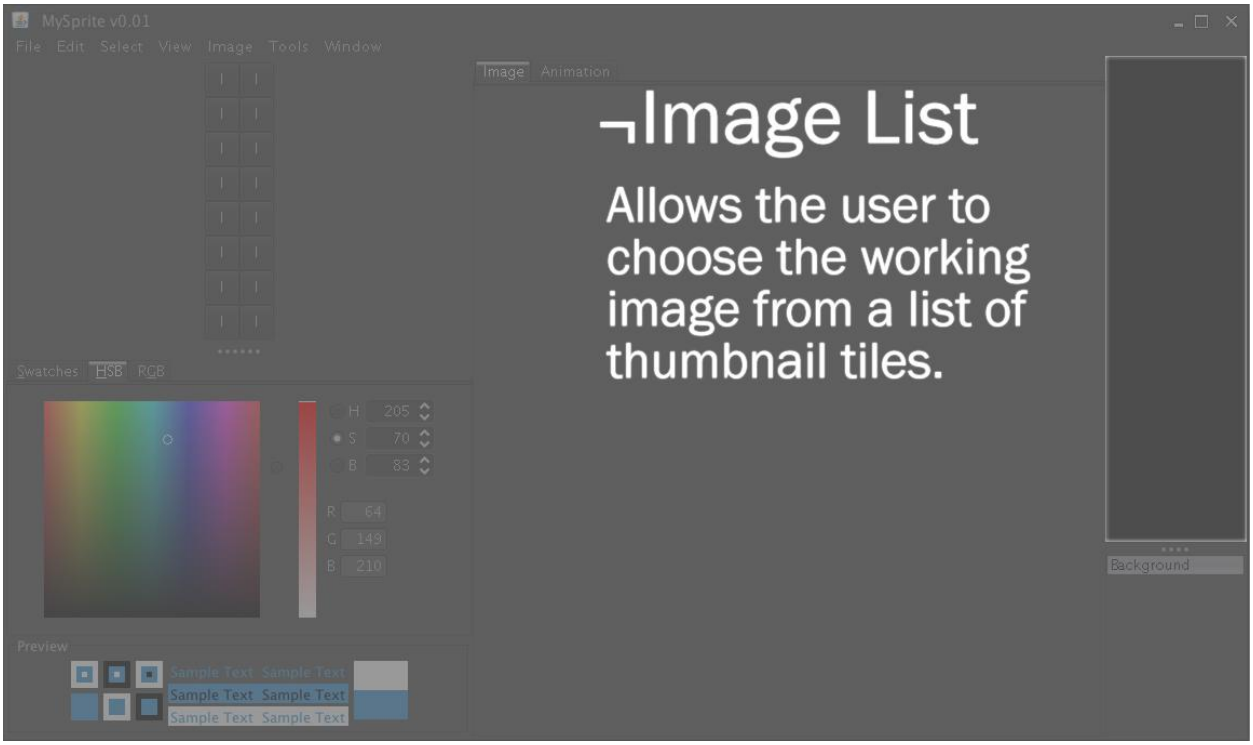



Fig 11.0e – Image List

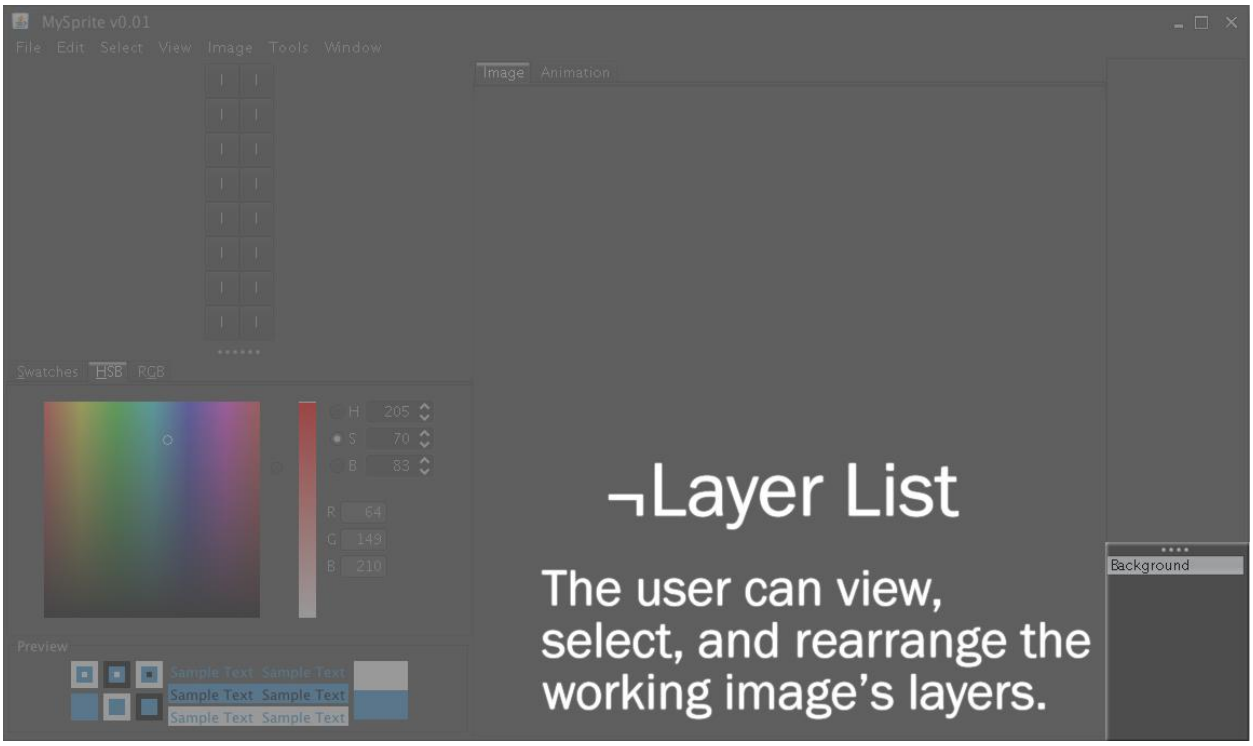



Fig 11.0f – Layer List

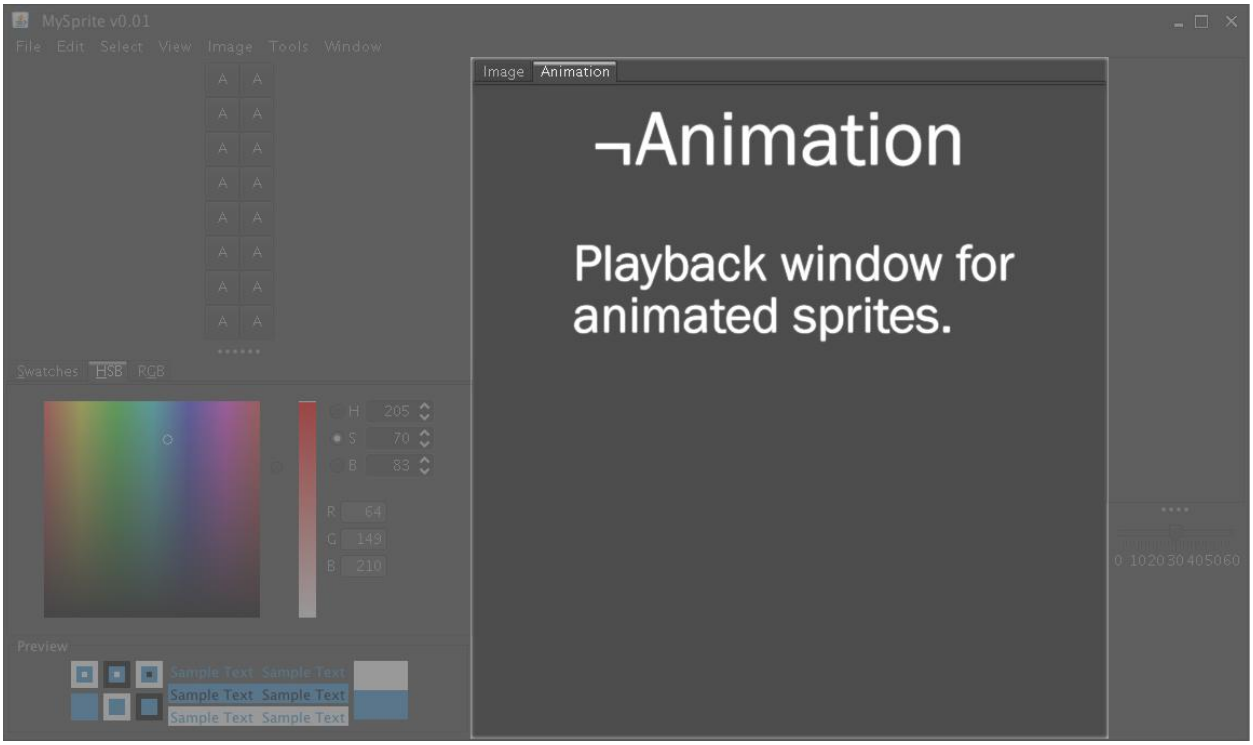



Fig 11.0g – Animation Tab

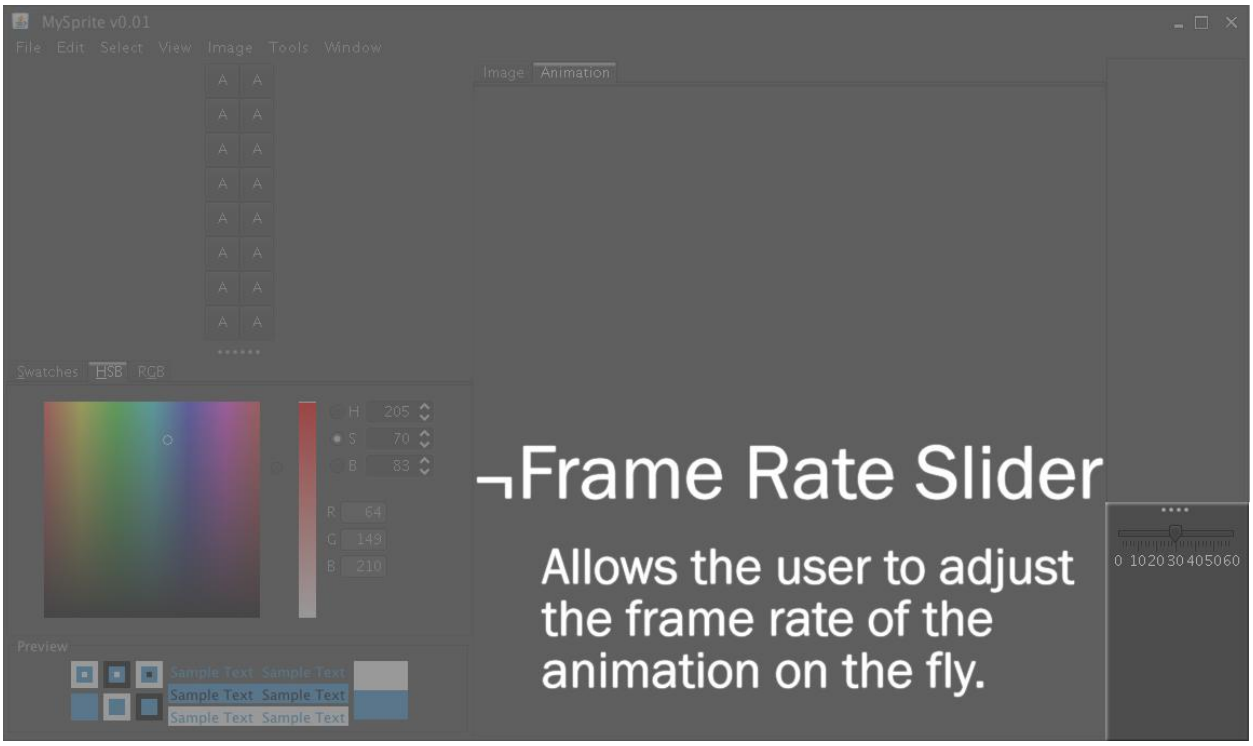



Fig 11.0h – Frame Rate Slider

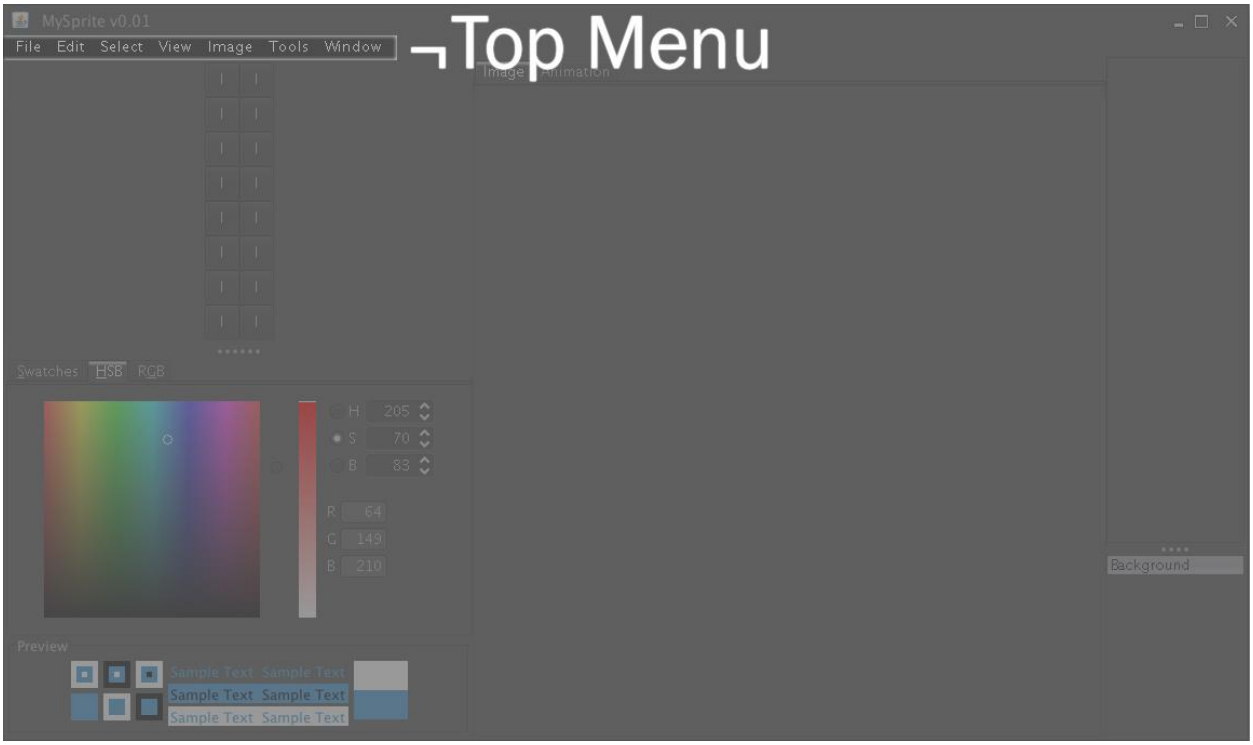



Fig 11.0i – Top Menu

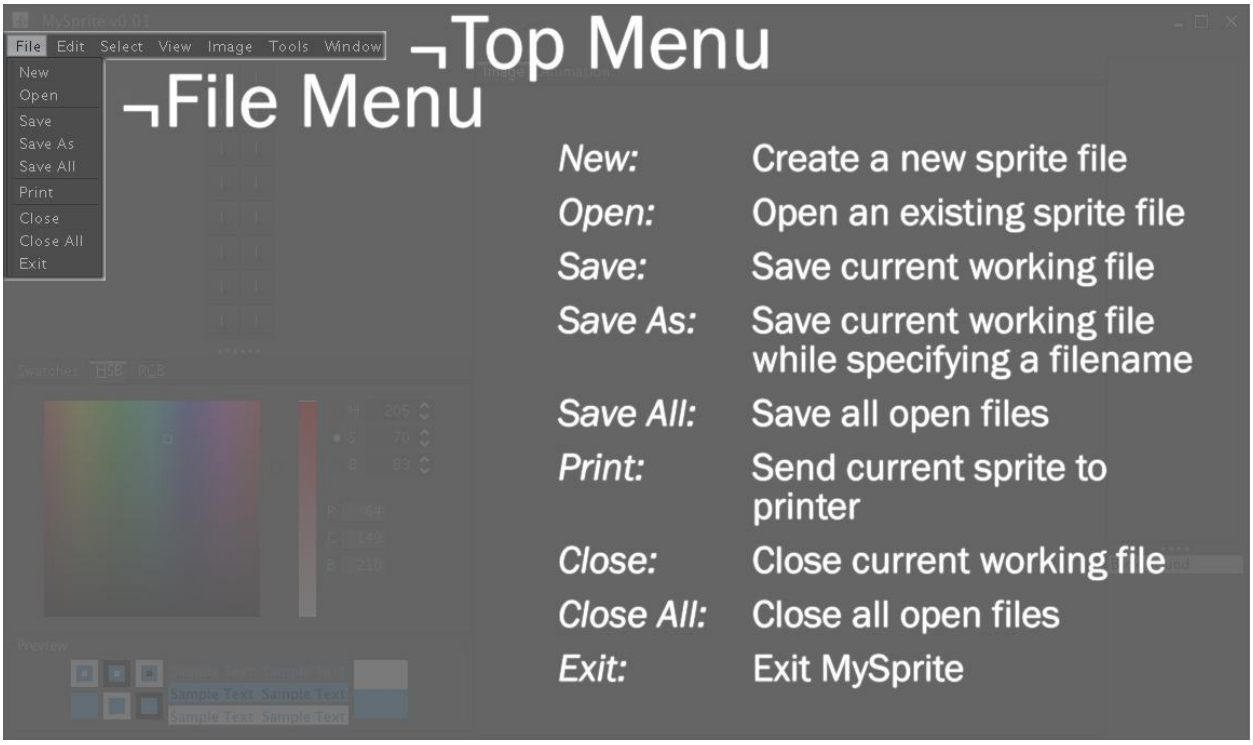



Fig 11.0j – File Menu

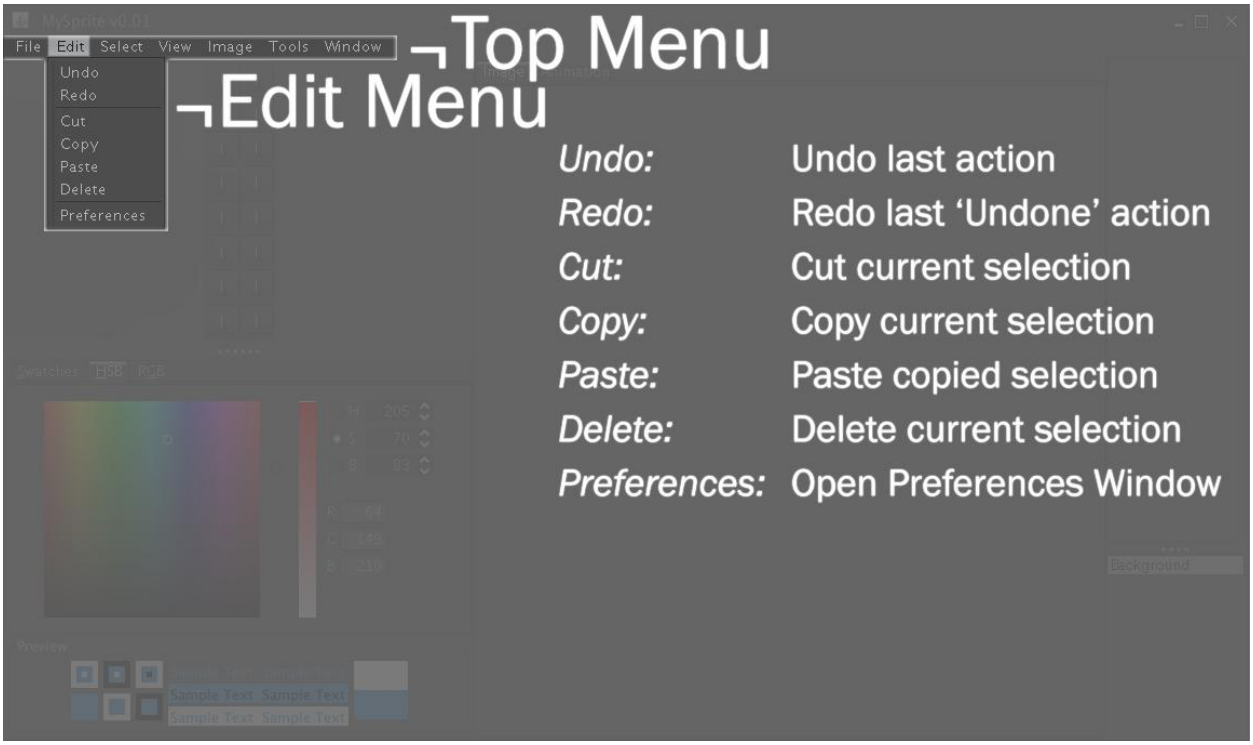



Fig 11.0k – Edit Menu


Fig 11.0l – Select Menu

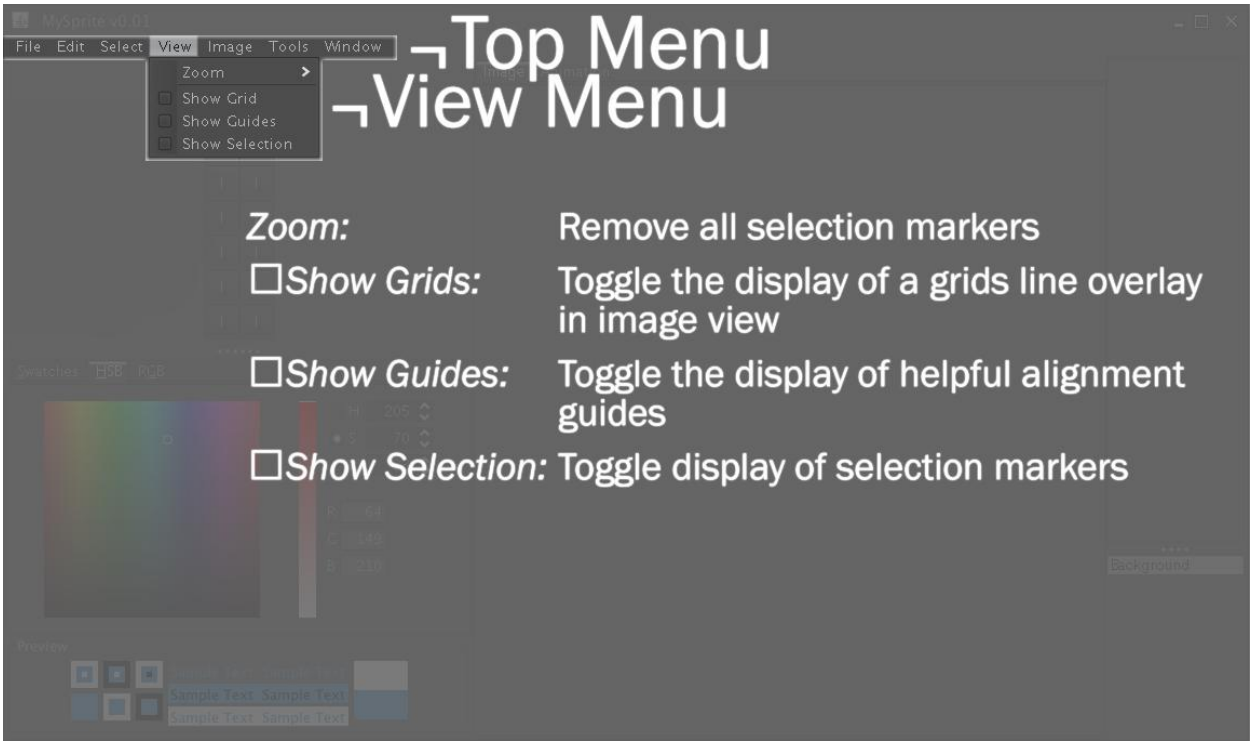



Fig 11.0m – View Menu

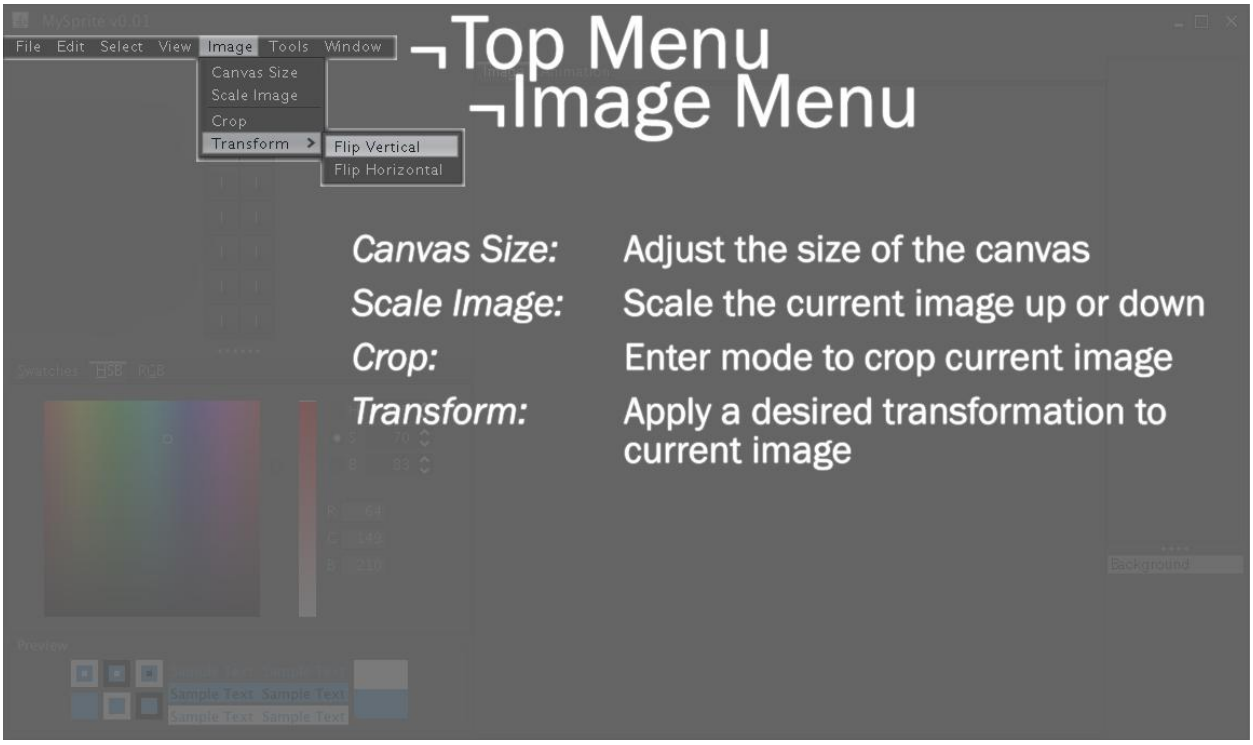
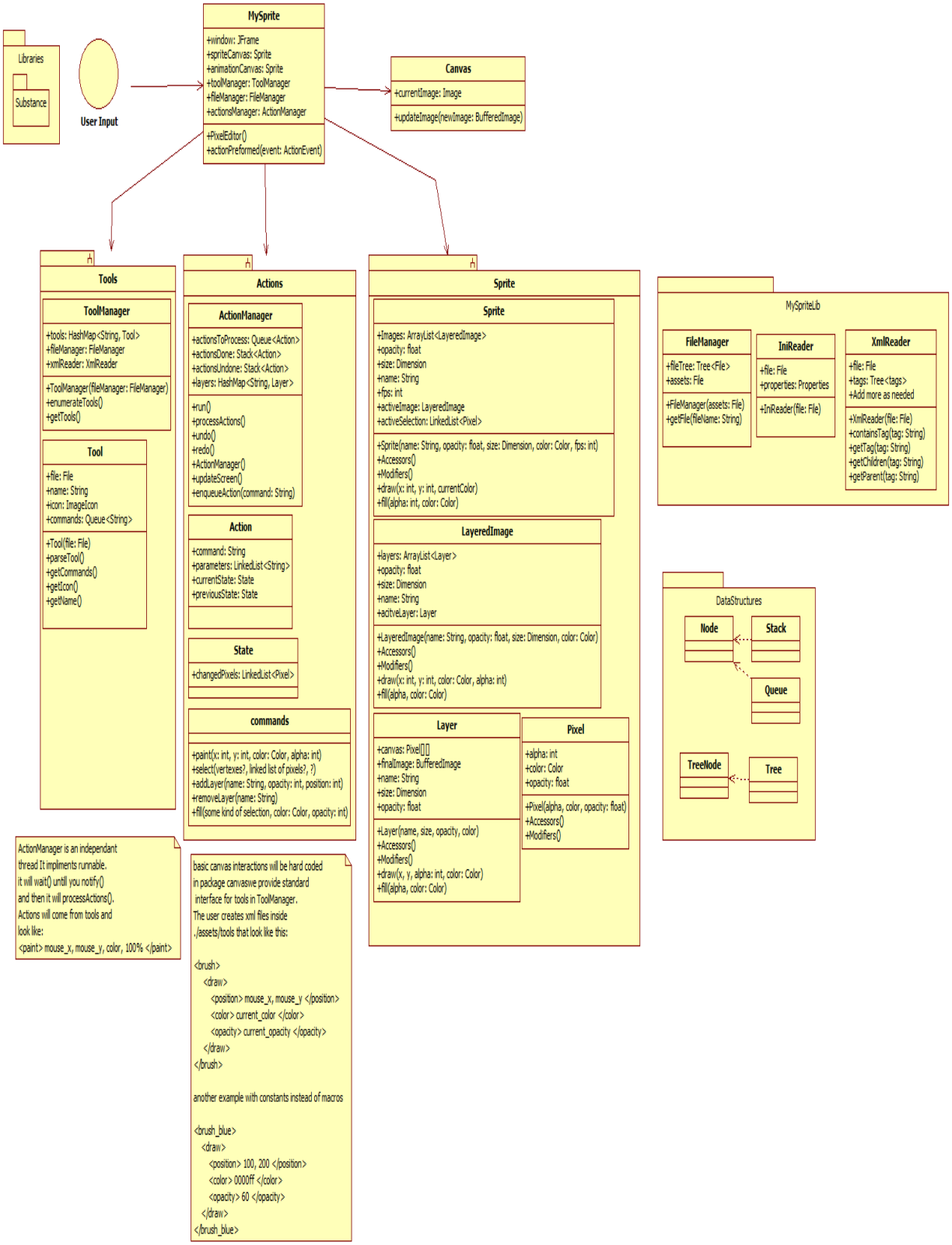



Fig 11.0n – Image Menu

## 12.0 Class Interface Diagram



## 13.0 Class Interface

```
/** A Sprite is a two-dimensional image or animation that is integrated into a larger scene.  A sprite is
* typically used for characters and other moving objects in video games.
*
* A sprite is designed from a blank canvas that varies in size.  The user can select from an array of
* predetermined sizes or make a custom canvas.  The user starts designing the first layer, which can
*then be copied to the next layer for modification or add a blank layer and start from scratch.
*
* Invariants:
* The number of layers is positive at all times.
*         @invariant       getNumberLayers() > 0
* The minimum number of layers is greater than 1
*         @invariant       getMinLayers() > 1
**/
public class MySprite {

        JFrame window;
        Sprite spriteCanvas;
        Sprite animationCanvas;
        ToolManagertoolManager;
        FileManagerfileManager;
        ActionManageractionsManager;
        public void PixelEditor() {...}
        /*
         *@pre:
         *@post:
         */
        public void actionPreformed(ActionEvent event) {...}
        /*
         *@pre:
         *@post:
         */
}
public class ToolManager {

        HashMap<String, Tool> tools;
        FileManagerfileManager;
        XmlReaderxmlReader;

        /**
          *Public constructor
          */
        publicToolManager(FileManagerfileManager) {...}
        /*
         *@pre: fileManager exists
         *@post: A new ToolManager is created
         */
```

```
        public void enumerateTools() {...}
        /*
         *@pre: n/a
         *@post: All possible tools are listed
         */
        public void getTools() {...}
        /*
         *@pre: The tool exists.
         *@post: Return the specified tool.
         */
}
public class Tool {

        File file;
        String name;
        ImageIcon icon;
        Queue<String> commands;

        /*
          *Public constructor
         */
        publicTool(File file) {...}
        /*
         *@pre:
         *@post:
         */
        public void parseTool() {...}
        /*
         *@pre:
         *@post:
         */
        public void getCommands() {...}
        /*
         *@pre:
         *@post:
         */
        public void getIcon() {...}
        /*
         *@pre:
         *@post:
         */
        public void getName() {...}
        /*
         *@pre:
         *@post:
         */
}
public class ActionManager {
```

```
        Queue<Action>actionsToProcess;
        Stack<Action>actionsDone;
        Stack<Action>actionsUndone;
        HashMap<String, Layer> layers;
        public void run() {...}
        /*
         *@pre:
         *@post:
         */
        public void processActions() {...}
        /*
         *@pre:
         *@post:
         */
        public void undo() {...}
        /*
         *@pre:
         *@post:
         */
        public void redo() {...}
        /*
         *@pre:
         *@post:
         */

        /*
          *Public Constructor
          */
        publicActionManager() {...}
        /*
         *@pre:
         *@post:
         */
        public void updateScreen() {...}
        /*
         *@pre:
         *@post:
         */
        public void enqueueAction(String command) {...}
        /*
         *@pre:
         *@post:
         */
}
public class Action {

        String command;
```

```
        LinkedList<String> parameters;
        State currentState;
        State previousState;
}
public class State {

        LinkedList<Pixel>changedPixels;
}
public class commands {

        public void paint(intx,inty,Colorcolor,int alpha) {...}
        /*
         *@pre:
         *@post:
         */
        public void select(vertexes?,linked list of pixels?,?) {...}
        /*
         *@pre:
         *@post:
         */
        public void addLayer(String name,intopacity,int position) {...}
        /*
         *@pre:
         *@post:
         */
        public void removeLayer(String name) {...}
        /*
         *@pre:
         *@post:
         */
        public void fill(some kind of selection,Colorcolor,int opacity) {...}
        /*
         *@pre:
         *@post:
         */
}
public class Canvas {

        Image currentImage;
        public void updateImage(BufferedImagenewImage) {...}
        /*
         *@pre:
         *@post:
         */
}

public class Sprite {
```

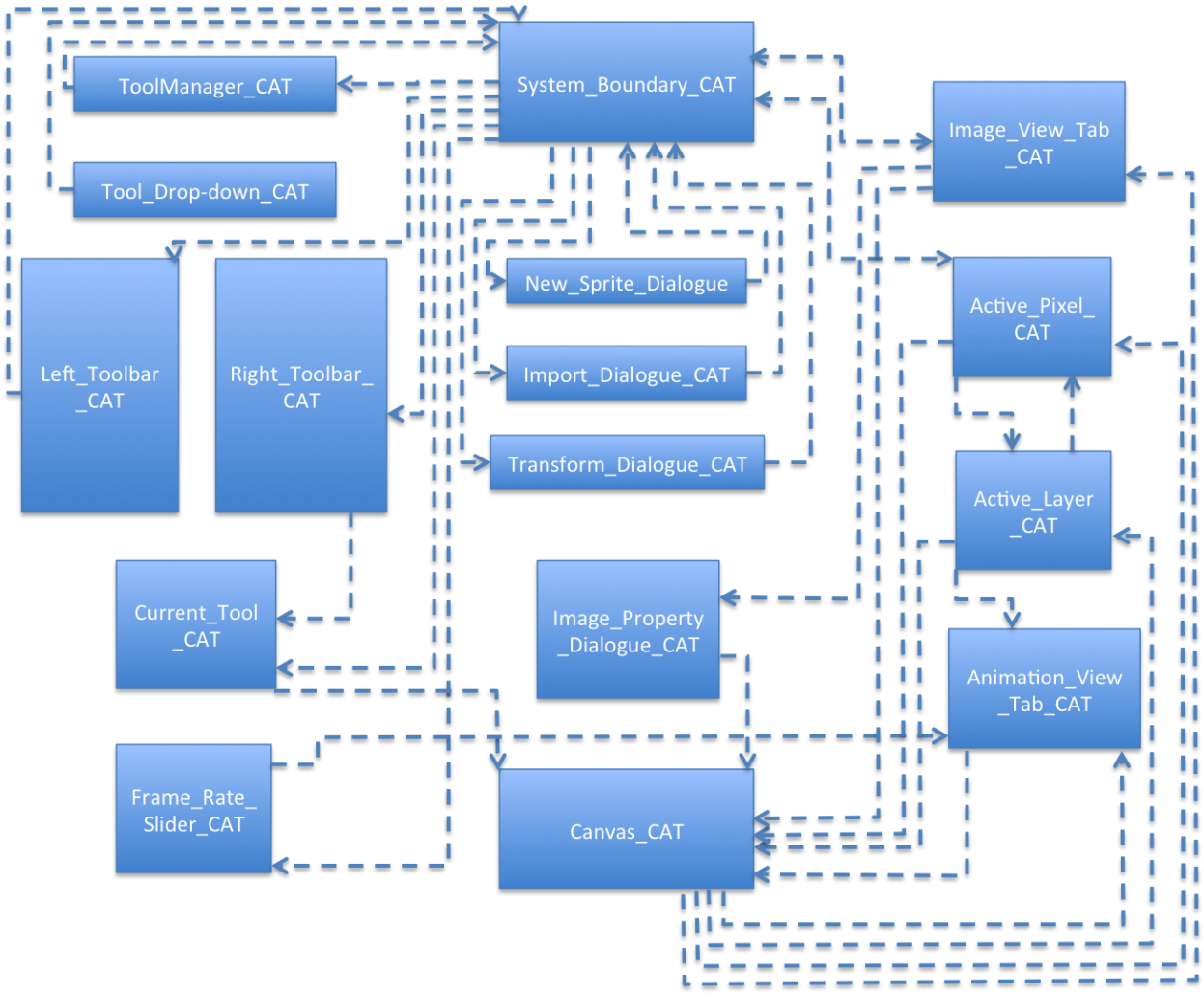
```
        ArrayList<LayeredImage> Images;
        float opacity;
        Dimension size;
        String name;
        int fps;
        LayeredImageactiveImage;
        LinkedList<Pixel>activeSelection;

        /*
         *Public constructor.  A Sprite starts with a name, opacity, size, color,
         *and fps.  These attributes cannot be changed.
         */
        public Sprite(String name,floatopacity,Dimensionsize,Colorcolor,int fps) {...}
        /*
         *@pre:
         *@post:
         */
        public void Accessors() {...}
        /*
         *@pre:
         *@post:
         */
        public void Modifiers() {...}
        /*
         *@pre:
         *@post:
         */
        public void draw(intx,inty,currentColor) {...}
        /*
         *@pre:
         *@post:
         */
        public void fill(intalpha,Color color) {...}
        /*
         *@pre:
         *@post:
         */
}
public class LayeredImage {

        ArrayList<Layer> layers;
        float opacity;
        Dimension size;
        String name;
        Layer acitveLayer;

        /*
         *Public constructor.  A LayeredImage requires a name, opacity, size, and color
```

```
         *These attributes cannot be changed.
         */
        publicLayeredImage(String name,floatopacity,Dimensionsize,Color color) {...}
        /*
         *@pre:
         *@post:
         */
        public void Accessors() {...}
        /*
         *@pre:
         *@post:
         */
        public void Modifiers() {...}
        /*
         *@pre:
         *@post:
         */
        public void draw(intx,inty,Colorcolor,int alpha) {...}
        /*
         *@pre:
         *@post:
         */
        public void fill(alpha,Color color) {...}
        /*
         *@pre:
         *@post:
         */
}
public class Layer {

        Pixel[][] canvas;
        BufferedImagefinalImage;
        String name;
        Dimension size;
        float opacity;

        /*
         *Public constructor.  A Layer requires a name, size, opacity, and color.  These
         *attributes cannot be changed.
         */
        publicLayer(name,size,opacity,color) {...}
        /*
         *@pre:
         *@post:
         */
        public void Accessors() {...}
        /*
         *@pre:
```

```
        *@post:
        */
        public void Modifiers() {...}
        /*
        *@pre:
        *@post:
        */
        public void draw(x,y,intalpha,Color color) {...}
        /*
        *@pre:
        *@post:
        */
        public void fill(alpha,Color color) {...}
        /*
        *@pre:
        *@post:
        */
}
public class Pixel {

        int alpha;
        Color color;
        float opacity;
        public void Pixel(alpha,color,float opacity) {...}
        /*
        *@pre:
        *@post:
        */
        public void Accessors() {...}
        /*
        *@pre:
        *@post:
        */
        public void Modifiers() {...}
        /*
        *@pre:
        *@post:
        */
}
public class FileManager {
        Tree<File>fileTree;
        File assets;
        /*
         *Public constructor
         */
        publicFileManager(File assets) {...}
        /*
        *@pre:
```

```
        *@post:
        */
        public void getFile(String fileName) {...}
        /*
        *@pre:
        *@post:
        */
}
public class IniReader {

        File file;
        Properties properties;
        public void IniReader(File file) {...}
        /*
        *@pre:
        *@post:
        */
}
public class XmlReader {

        File file;
        Tree<tags> tags;
        public void XmlReader(File file) {...}
        /*
        *@pre:
        *@post:
        */
        public void containsTag(String tag) {...}
        /*
        *@pre:
        *@post:
        */
        public void getTag(String tag) {...}
        /*
        *@pre:
        *@post:
        */
        public void getChildren(String tag) {...}
        /*
        *@pre:
        *@post:
        */
        public void getParent(String tag) {...}
        /*
        *@pre:
        *@post:
        */
}
```

## 14.0 Category Interaction Diagram



## 15.0 COCOMO

(Person-months)

- $Effort\ Applied\ (\boldsymbol{E}) = a(KLoC)^{\boldsymbol{b}} * Effort\ Adjustment\ Factor\ (\boldsymbol{EAF})$

(Months)

- $Development\ Time\ (\boldsymbol{D}) = \boldsymbol{cE^d}$

(People)

- $People\ Required\ (\boldsymbol{P}) = \frac{\boldsymbol{E}}{\boldsymbol{D}}$

| MySprite | a | b | c | d |
|---|---|---|---|---|
| Semi-detached | 3.0 | 1.12 | 2.5 | 0.35 |

| Product Attributes | Very Low | Low | Nominal | High | Very High | Extra High |
|---|---|---|---|---|---|---|
| **Required Software Reliability** | | | 1.00 | | | |
| **Size of Application Database** | | 0.94 | | | | |
| **Complexity of Project** | | | | *1.22 | | |
| **Hardware Attributes** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** | **Extra High** |
| **Run-time Performance Constraints** | | | 1.00 | | | |
| **Memory Constraints** | | | 1.00 | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Volatility of VM Environment** | | 0.87 | | | | |
| **Personnel Attributes** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** | **Extra High** |
| **Analyst Capabilities** | | 1.19 | | | | |
| **Application Experience** | | | 1.00 | | | |
| **Software Engineer Capability** | | | 1.00 | | | |
| **VM Experience** | | | | 0.90 | | |
| **Programming Language Experience** | | | 1.0 | | | |
| **Project Attributes** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** | **Extra High** |
| **Multisite Development** | | | | | 0.82 | |
| **Use of Software Tools** | | | 1.00 | | | |
| **Required Development Time** | | | | 1.04 | | |

$$EAF = Product of all Cost Drivers$$
$$EAF = 1.00 * 0.94 * 1.22 * 1.00 * 1.0 * 0.87 * 1.19 * 1.00 * 0.90 * 1.00 = 1.07$$
$$E = a(kLoc)^b * EAF = 3.0 * (8.500)^{1.12} * 1.07 = 35.3 person_{months}$$
$$D = cE^d = 2.5(35.3)^{0.35} = 8.7 months$$
$$P = \left\lceil \frac{E}{D} \right\rceil = \left\lceil \frac{35.3}{8.7} \right\rceil = \lceil 4.06 \rceil people = 5\ people$$
$$Cost = P * Cost_{person-month} = 5 * \$8,000 = \$40,000$$

## 16.0 COCOMO vs. Functional Point Cost Analysis

When considering the different methods of estimating cost, COCOMO versus Functional Point Cost Analysis, we derived two surprisingly different results. While the Functional Point Cost Analysis method seems to primarily focus on the expected functionality of the software, key indicators of which can be ascertained from the use cases, it seems to focus primarily on estimating functionality without regard to other aspects of the project such as the estimated total lines of code, number of and skillset of software developers to name a few. It is our opinion that the COCOMO method of analyzing/estimating overall cost is a more modern and accurate method than the Functional Point Cost Analysis.

We are sure that there are cases where the Functional Point Cost Analysis may be the preferred method of estimating cost, such as when developing software for a client with a keen sense of functionality and strict requirements of such. For a small development team, such as 5GUISE, the COCOMO method that focuses on the estimated total lines of code would better serve our development process by providing more accurate, competitive quotes to our customers. Over time, after developing software packages of varying degrees of difficulty our team will be able to better estimate the total lines of code and complexity of projects, thereby, acquiring more clients.

## 17.0 Test Cases & Rationale

### 17.1 Test Cases

**Test Case: 01**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | Create a new sprite (UC02) |
| Location | MySprite.jar |
| Input | From the menu-bar the user selects: File > New |
| Oracle | A new, blank sprite canvas is created. |
| Log | A user is able to set the width and height of the canvas, as well as the sprite name by selecting File > New. |

**Test Case: 02**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: top-down |
| Name | Modify/edit an existing sprite (UC03 & UC04) |
| Location | MySprite.jar |
| Input | From the menu-bar the user selects: Image > New Layer or Image > Delete Layer |
| Oracle | A sprite is modified by adding, removing, or changing a layer. |
| Log | **A sprite was modified by adding, removing, and changing a layer.** |

**Test Case: 03**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | User uses built in tools (UC05) |
| Location | MySprite.jar |
| Input | The user selects one of the available tools from the tool pane and makes the tool specific change to the canvas image. |
| Oracle | User is able to select and use individual tools to affect the image canvas. |
| Log | **A user can use one of many built in tools such the pencil or line tool.** |

**Test Case: 04**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | User creates a new tool (UC06) |
| Location | MySprite.jar |
| Input | From the menu-bar the user selects new tool. |
| Oracle | The user is able to create a custom tool and add it to the tool area. |
| Log | **User must use correct syntax in XML file.** |

**Test Case: 05**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | Canvas window zoom (UC07) |
| Location | MySprite.jar |
| Input | On the menu-bar the user selects: View > "Zoom In \| Zoom Out \| Zoom To" or uses the mouse wheel to zoom in and out. |
| Oracle | By using a mouse wheel or selecting zoom in/out from the menu system, the user is able to zoom the image. |
| Log | **The canvas window zoom can be done by using the mouse scroll wheel.** |

**Test Case: 06**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing |
| Name | User switches between images (UC11) |
| Location | MySprite.jar |
| Input | From the images area a user selects a new image by left-clicking with the mouse. |
| Oracle | By using a mouse, the user will be able to select a different image. |
| Log | **On both the image and animation tab a user can select any image.** |

**Test Case: 07**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing |
| Name | User switches between animations (UC12) |
| Location | MySprite.jar |
| Input | From the Animation Menu the user can “Cycle Images \| Left or Right” |
| Oracle | The user is able to switch between animations by clicking on a new one from the available list. |
| Log | **From the Animation Menu, the user was able to switch animations by selecting Cycle Images and then selecting either Left or Right.** |

**Test Case: 08**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing |
| Name | Canvas is altered by mouse-based tool interactions (UC14 & UC15) |
| Location | MySprite.jar |
| Input | The user selects a tool from the tool palette and makes changes to the canvas. |
| Oracle | The canvas changes according the tool being used. |
| Log | **By selecting a color from the palette a user is able to interact with the canvas and make changes.** |

**Test Case: 09**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | Adjust layer properties (UC16) |
| Location | MySprite.jar |
| Input | From the menu-bar the user is able to adjust a layer’s properties. |
| Oracle | The user is able to adjust the current layer properties. |
| Log | **Structure exists, but has not been implemented.** |

**Test Case: 10**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | Adjust image properties (UC17) |
| Location | MySprite.jar |
| Input | From the menu-bar the user is able to adjust an image’s properties. |
| Oracle | The user is able to adjust the current image properties. |

| | |
|---|---|
| Log | **Structure exists, but has not been implemented.** |

**Test Case: 11**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | Animation playback frame rate adjustment (UC19) |
| Location | MySprite.jar |
| Input | The user selects the "Animation" tab. The frame rate slider is located in the lower right corner and the user adjusts the number of frames per second (fps) from 0 to 60. |
| Oracle | The user is able to adjust the animation frame rate by adjusting the slider. |
| Log | **The frame rate setting adjusts to the value selected by the user, from 0 to 60 frames per second (fps).** |

**Test Case: 12**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: bottom-up |
| Name | User changes/modifies selected pixel(s) (UC22, UC23, UC24) |
| Location | MySprite.jar |
| Input | Using a tool and by selecting: View > "Zoom In \| Zoom Out \| Zoom To" the user is able to manipulate a pixel according to the type of tool selected. |
| Oracle | The user is able to change selected pixels with a tool. |
| Log | **The user is able to directly modify pixel(s) on the canvas.** |

**Test Case: 13**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: top-down |
| Name | Transform sprite (UC25) |
| Location | MySprite.jar |
| Input | From the menu bar the user selects: Modifiers > Transform > "Shift \| Flip \| Rotate \| Scale \| Stretch" |
| Oracle | The user is able to transform the sprite. |
| Log | **Structure exists, but has not been implemented.** |

**Test Case: 14**

| Attributes | Description |
|---|---|
| Test Case Type | Functional testing: top-down |
| Name | Modify layers (UC30) |
| Location | MySprite.jar |
| Input | From the menu bar the user selects: Image > "New Layer \| Delete Layer \| Merge Layer" |
| Oracle | The user is able to modify individual layers. |
| Log | **Structure exists, but has not been implemented.** |

## 17.2 Rationale for Test Cases

Given the time constraints for this project, it was decided to primarily perform functional testing for our test cases. Our test cases are derived directly from our use cases. Several of the use cases were combined due to commonality. The test cases primarily deal with the overall functionality of the GUI, testing common features such as buttons and selections made from the menu-bar. At the time of printing this document only a few of the test cases could be tested. Test cases that have not been tested are denoted with a <pending> for the Log entry. All test cases will be tested prior to our group presentation on April 12 and our final, bound document turned in on April 19 will reflect the actual results from these test cases.

## 18.0 Source Code

### 18.1 MySprite.java

```
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
//
//
//  MySprite.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//  Matt Gold        : mattbgold@gmail.com
//
//  Robert Jenkins   : rjenkins12@student.gsu.edu
//
//  Michael Burlison : mburlison@gmail.com
//
//
//
//  Main GUI, Thread Creation, file managing,  event handleing methods found here.
//
//
//
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////

import ActionManager.ActionManager;
import ActionManager.Tool;
import ActionManager.ToolManager;
import DataStructures.LayeredImage;
import DataStructures.Sprite;
import DataStructures.Stack;
import MySpriteLib.FileManager;
import MySpriteLib.GuiElement;
import MySpriteLib.GuiElement.AnimationCanvas;
import MySpriteLib.GuiElement.ImageCanvas;
import MySpriteLib.GuiElement.ImagePreview;
import MySpriteLib.IniInterface;
import org.pushingpixels.substance.api.skin.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JColorChooser;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
```

```
import javax.swing.JMenuItem;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JSpinner;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.filechooser.*;
import javax.swing.ListSelectionModel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.awt.image.BufferedImage;
import static javax.swing.ScrollPaneConstants.*;

//////////////
// MySprite //
//////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
// MySprite extends JFrame and acts as both our gui, and our main class
// It instantiates itself, spawning a concurrent thread that waits for user input
// and reacts triggering listener functions.
//////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
  public class MySprite extends JFrame {

  ///////////////
  // Constants //
  ///////////////
    private static final long       serialVersionUID   = 1L;
    private        final boolean    debug              = false;
    private static final String     INIT_TITLE         = "MySprite v0.1";
    private static final String     PREFERENCES_FILE   = "MySprite.ini";
    private static final String     DEFAULT_SKIN       = "Raven";
    private        final int        MIN_FPS            = 0;
    private        final int        MAX_FPS            = 60;
    private        final int        INIT_FPS           = 30;
    private        final Dimension  NEW_IMAGE_SIZE     = new Dimension(32, 32);
```

```
  private        final Dimension  minimumSize        = new Dimension(640, 480);
  private        final Dimension  rightSize          = new Dimension(128, 480);
  private        final Dimension  toolsGridSize      = new Dimension(256,96);
  private        final Dimension  toolsGridAxis      = new Dimension(3,8);
  private        final Dimension  animationGridSize  = new Dimension(800,600);
  private        final Dimension  animationImageSize = new Dimension(90,90);
  private        final Dimension  animationGridAxis  = new Dimension(0,6);
  private        final Dimension  fpsSliderSize      = new Dimension(128, 58);
  private        final Dimension  opacitySliderSize  = new Dimension(128, 58);

//////////////////////
// Instance Variables //
//////////////////////
  private         boolean         showLeft           = true;
  private         boolean         showCenter         = true;
  private         boolean         showRight          = true;
  private         boolean         imageTabActive     = true;
  public  static  FileManager     fileManager;
  public  static  IniInterface    preferences;
  public  static  ToolManager     toolManager;
  public  static  ActionManager   actionManager;
  public  static  MySprite        window;
  public          BorderLayout    mainLayout;
  private         JMenuBar        menuBar;
  private         Stack<JMenu>    menuStack;
  private         GuiElement[][]  menuDesign;
  public          Sprite          sprite;
  public          LayeredImage    imageClipboard;

//////////////////
// GUI Components //
//////////////////
  private         JSplitPane      left;
  private         JPanel          imageTools;
  private         JPanel          animationPreview;
  private         JPanel          imageToolsGrid;
  private         JPanel          animationGrid;
  private         JColorChooser   colorPicker;
  private         JColorChooser   animColorPicker;
  private         JTabbedPane     center;
  private         ImageCanvas     image;
  private         AnimationCanvas animation;
  private         JSplitPane      right;
  private         JPanel          images;
  private         JScrollPane     imagesScrollPane;
  private         JList           imageLayers;
  private         JPanel          animationButtons;
  private         JPanel          animationControls;
  private         JPanel          spriteButtons;
  private         JPanel          rgbaPanel;
  private         JSlider         opacitySlider;
  private         JSpinner        opacityField;

/////////////////
// Constructor //
```

```
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
  // This constructor instantiates all instance variables with their default values, It also
instantiates
  // all GUI components and adds them to the window, setting all relevant properties.
  // Code in this procedure is arranged by which screen region the component will be
occupying. Each reagion is
  // proceeded by a header giving details to it's location on the page.

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    public MySprite ( String windowName ) {
      super(windowName);

      //////////////
      // MenuBar //

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
      // This section instantiates the menubar, commonly found at the top of an application.
      // This section reads GuiElement.menuDesign, a 2D matrix of GuiElement, into a stack
based upon the
      // enum GuiElement.MenuType. Every time a MenuType.Menu is encountered, it is pushed
onto the stack.
      // every other MenuType is added to the top element until an MenuType.EndMenu is
encountered. Then the
      // Stack is popped.
      //
      // There is an implicit EndMenu at the end of each row in the array.

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
        menuDesign = GuiElement.menuDesign;
        menuBar    = new JMenuBar();
        menuStack  = new Stack<JMenu>();

        for(GuiElement[] currentJMenu : menuDesign){
          for(GuiElement currentElement : currentJMenu){

            //////////////////////
            // MenuType.EndMenu //
            //////////////////////
              if (currentElement.type == GuiElement.MenuType.EndMenu){
                JMenu subMenu = new JMenu("Temp");
                try{
                  subMenu = menuStack.pop();
                  menuStack.peek().add(subMenu);
                } catch (NullPointerException e){
                  debugError("Invalid Menu Design", subMenu.getText(), "failed to
initialize");
                }
              }

            ///////////////////
            // MenuType.Menu //
            ///////////////////
```

```
            else if(currentElement.type == GuiElement.MenuType.Menu){
              JMenu temp = new JMenu(currentElement.value);
              temp.setMnemonic(currentElement.key);
              temp.addActionListener(new MenuActionListener());
              menuStack.push(temp);
            }

          ////////////////////////
          // MenuType.MenuItem //
          ////////////////////////
            else if (currentElement.type == GuiElement.MenuType.MenuItem){
              try{
                JMenuItem temp = new JMenuItem(currentElement.value);
                temp.setMnemonic(currentElement.key);
                temp.addActionListener(new MenuActionListener());
                menuStack.peek().add(temp);
              } catch (NullPointerException e){
                debugError("Invalid Menu Design", currentElement.value,"No parent menu");
              }
            }

          ////////////////////////
          // MenuType.Checkbox //
          ////////////////////////
            else if (currentElement.type == GuiElement.MenuType.Checkbox){
              try{
                JCheckBoxMenuItem temp = new JCheckBoxMenuItem(currentElement.value);
                temp.setMnemonic(currentElement.key);
                temp.addActionListener(new MenuActionListener());
                menuStack.peek().add(temp);
              } catch (NullPointerException e){
                debugError("Invalid Menu Design", currentElement.value,"No parent menu");
              }
            }

          //////////////////////////
          // MenuType.Radiobutton //
          //////////////////////////
            else if (currentElement.type == GuiElement.MenuType.Radiobutton){
              try{
                JRadioButtonMenuItem temp = new
JRadioButtonMenuItem(currentElement.value);
                temp.addActionListener(new MenuActionListener());
                temp.setMnemonic(currentElement.key);
                if(currentElement.value == DEFAULT_SKIN)
                {
                  temp.setSelected(true);
                }
                currentElement.group.add(temp);
                menuStack.peek().add(temp);
              } catch (NullPointerException e){
                debugError("Invalid Menu Design", currentElement.value,"No parent menu");
              }
            }

          ////////////////////////
          // MenuType.Separator //
```

```
            ////////////////////////
              else if (currentElement.type == GuiElement.MenuType.Separator){
                try{
                  menuStack.peek().addSeparator();
                } catch (NullPointerException e){
                  debugError("Invalid Menu Design", currentElement.value,"No parent menu");
                }
              }
            else {
              debugError("Invalid Menu Design", currentElement.type.toString(), "is invalid
JMenu Component");
            }
          }
          ////////////////////////
          // Implicit EndMenu //
          ////////////////////////
            JMenu menu = new JMenu("Temp");
            try{
              menu = menuStack.pop();
              menuBar.add(menu);
            } catch (NullPointerException e){
              debugError("Invalid Menu Design", menu.getText(), "failed to initialize
properly");
            }
        }
        this.setJMenuBar(menuBar);

      ////////////////
      // MainWindow //

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
      // This section instantiates all main Gui Components.
      //
      // Main Regions include:
      //   Left, Right, Center

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
        mainLayout = new BorderLayout();
        this.setLayout(mainLayout);
        sprite = new Sprite("mySprite", NEW_IMAGE_SIZE);

        //////////
        // Left //

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
        // local indentation indicates nesting level. first previous item of lower
indentation is parent.
        //
        // imageTools           : JPanel
        //   imageToolsGrid     : JPanel
        //   opacityPanel       : JPanel
        //     opacityBox       : JPanel
        //       opacityField   : JPanel
        //     opacitySlider    : JSlider
```

```
        //   rbgaPanel          : Jpanel
        //     colorPicker      : JColorChooser
        // animationTools       : JPanel
        //   animationPreview   : JPanel
        //     animation        : GuiElement.AnimationCanvas
        //   animationControl   : JPanel
        //     animationButtons : JPanel
        //     framesPerSecond  : JSlider
        //   rbgaPanel          : JPanel
        //     colorPicker      : JColorChooser

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////

          ////////////////////
          // imageToolsGrid //
          ////////////////////
          imageToolsGrid = new JPanel();
          imageToolsGrid.setLayout(new
GridLayout(toolsGridAxis.width,toolsGridAxis.height,1,1));
          imageToolsGrid.setPreferredSize(toolsGridSize);
          for(int gridx=0; gridx < toolsGridAxis.width ;gridx++){
            for(int gridy=0; gridy < toolsGridAxis.height ;gridy++){
              int index = (gridx*toolsGridAxis.width)+gridy;
              if (toolManager == null || index >= toolManager.size()) {
                JLabel blank = new JLabel();
                imageToolsGrid.add(blank);
              } else {
                JButton current = new JButton();
                try{
                  LinkedList<Tool> tools = toolManager.getTools();
                  current.addActionListener(new ToolActionListener(tools.get(index)));
                  current.setToolTipText(tools.get(index).name);
                  current.setIcon(
                    new ImageIcon(
                      fileManager.getPng(
                        tools.get(index).icon
                      ).getAbsolutePath()
                    )
                  );
                } catch (NullPointerException e) {
                  current.setText("I");
                }
                imageToolsGrid.add(current);
              }
            }
          }

          ////////////////////
          // opacitySlider //
          ////////////////////
          opacitySlider = new JSlider(JSlider.HORIZONTAL,0,100,100);
          opacitySlider.addChangeListener(new OpacitySliderListener());
          opacitySlider.setMajorTickSpacing(20);
          opacitySlider.setMinorTickSpacing(5);
          opacitySlider.setPaintTicks(true);
          opacitySlider.setPaintLabels(true);
```

```
        opacitySlider.setPreferredSize(opacitySliderSize);
        opacitySlider.setToolTipText("Opacity");

        /////////////////
        // ColorPicker //
        /////////////////
        colorPicker = new JColorChooser();
        colorPicker.getSelectionModel().addChangeListener(new ColorPickerListener());

        ///////////////
        // rbgaPanel //
        ///////////////
        rgbaPanel = new JPanel();
        rgbaPanel.setLayout(new BoxLayout(rgbaPanel,BoxLayout.Y_AXIS));

        ///////////////////
        // opacityField //
        ///////////////////
        SpinnerModel sm = new SpinnerNumberModel(100, 0, 100, 1);
        opacityField = new JSpinner(sm);
        opacityField.addChangeListener(new OpacityFieldListener());

        ////////////////
        // opacityBox //
        ////////////////
        JPanel opacityBox = new JPanel();
        opacityBox.setLayout(new BoxLayout(opacityBox,BoxLayout.Y_AXIS));
        opacityBox.add(new JLabel("Opacity"));
        opacityBox.add(opacityField);

        ///////////////////
        // opacityPanel //
        ///////////////////
        JPanel opacityPanel = new JPanel();
        opacityPanel.setLayout(new BoxLayout(opacityPanel,BoxLayout.X_AXIS));
        opacityPanel.add(opacityBox);
        opacityPanel.add(opacitySlider);
        rgbaPanel.add(opacityPanel);
        rgbaPanel.add(colorPicker);

        ///////////////
        // animation //
        ///////////////
        animation = new GuiElement.AnimationCanvas(sprite,toolsGridSize.width);

        ////////////////////////
        // animationButtons //
        ////////////////////////
        animationButtons = new JPanel();
        animationButtons.setLayout(new BoxLayout(animationButtons,BoxLayout.X_AXIS));

        JButton   temp = null;
        try{
          ImageIcon prev = new ImageIcon(fileManager.getPng("anim-
prev.png").getAbsolutePath());
          temp = new JButton(prev);
        } catch(NullPointerException e){
```

```
            temp = new JButton("<");
          }
          temp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
              animation.prevImage();
            }
          });
          temp.setPreferredSize(new Dimension(35,35));
          temp.setToolTipText("Back");
          animationButtons.add(temp);

          temp = null;
          try{
            ImageIcon stop = new ImageIcon(fileManager.getPng("anim-
stop.png").getAbsolutePath());
            temp = new JButton(stop);
          } catch(NullPointerException e){
            temp = new JButton("S");
          }
          temp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
              animation.stop();
            }
          });
          temp.setPreferredSize(new Dimension(35,35));
          temp.setToolTipText("Stop");
          animationButtons.add(temp);

          temp = null;
          try{
            ImageIcon play = new ImageIcon(fileManager.getPng("anim-
play.png").getAbsolutePath());
            temp = new JButton(play);
          } catch(NullPointerException e){
            temp = new JButton("P");
          }
          temp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
              animation.start();
            }
          });
          temp.setPreferredSize(new Dimension(35,35));
          temp.setToolTipText("Play");
          animationButtons.add(temp);

          temp = null;
          try{
            ImageIcon next = new ImageIcon(fileManager.getPng("anim-
next.png").getAbsolutePath());
            temp = new JButton(next);
          } catch(NullPointerException e){
            temp = new JButton(">");
          }
          temp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
              animation.nextImage();
            }
```

```
          });
          temp.setPreferredSize(new Dimension(35,35));
          temp.setToolTipText("Forward");
          animationButtons.add(temp);

          ////////////////////
          // framesPerSecond //
          ////////////////////
          JSlider framesPerSecond = new JSlider(JSlider.HORIZONTAL, MIN_FPS, MAX_FPS,
INIT_FPS);
          framesPerSecond.addChangeListener(new SliderChangeListener());
          framesPerSecond.setMajorTickSpacing(10);
          framesPerSecond.setMinorTickSpacing(2);
          framesPerSecond.setPaintTicks(true);
          framesPerSecond.setPaintLabels(true);
          framesPerSecond.setPreferredSize(fpsSliderSize);
          framesPerSecond.setToolTipText("FPS");

          ////////////////////////
          // animationControls //
          ////////////////////////
          animationControls = new JPanel();
          animationControls.setLayout(new BoxLayout(animationControls,BoxLayout.X_AXIS));
          animationControls.add(animationButtons);
          animationControls.add(framesPerSecond);

          imageTools = new JPanel();
          imageTools.add(imageToolsGrid);

          ////////////////////////
          // animationPreview //
          ////////////////////////
          animationPreview = new JPanel();
          animationPreview.setLayout(new BoxLayout(animationPreview,BoxLayout.Y_AXIS));
          animationPreview.add(animation);
          animationPreview.add(animationControls);

          animColorPicker = new JColorChooser();
          animColorPicker.getSelectionModel().addChangeListener(new ColorPickerListener());

        left = new JSplitPane(JSplitPane.VERTICAL_SPLIT, imageTools, rgbaPanel);
        left.setResizeWeight(1);
        if(showLeft) { this.add(left, BorderLayout.WEST); }

        ////////////
        // Center //

//////////////////////////////////////////////////////////////////////////////////////
////////////////////
        // Center is a JTabbedPane, The top two elemets are tabs, all others are nested
within the tabs
        // according to their indentation level.
        //
        // image          : GuiElement.ImageCanvas
        // animationPanel : JPanel
        //   scrollBorder : JPanel
        //     gridScroll : JScrollPane
```

```
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////

          ////////////
          // image //
          ////////////
          image = new GuiElement.ImageCanvas(actionManager, toolManager, sprite);
          image.setName("Image");

          ////////////////////
          // animationPanel //
          ////////////////////
          JPanel animationPanel = new JPanel();
          animationPanel.setName("Animation");

          //////////////////
          // scrollBorder //
          //////////////////
          JPanel scrollBorder = new JPanel();
          animationGrid = new JPanel();
          animationGrid.setLayout(new
GridLayout(animationGridAxis.width,animationGridAxis.height,8,8));
          scrollBorder.add(animationGrid);

          ////////////////
          // gridScroll //
          ////////////////
          JScrollPane gridScroll = new JScrollPane(scrollBorder);
          gridScroll.setHorizontalScrollBarPolicy(HORIZONTAL_SCROLLBAR_NEVER);
          gridScroll.setPreferredSize(animationGridSize);
          gridScroll.setName("Animation");

          animationPanel.add(gridScroll);

        center = new JTabbedPane();
        center.addTab(image.getName(), image);
        center.addTab(animationPanel.getName(), animationPanel);
        center.addChangeListener(new TabChangeListener());
        if(showCenter) { this.add(center, BorderLayout.CENTER); }

        ////////////
        // Right //

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
        // This region is actually fairly easy. No nesting involved.
        //
        // images        : JPanel
        // imageLayers   : JList
        // spriteButtons : JPanel

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////

          ////////////
          // images //
```

```
////////////
images = new JPanel();
images.setLayout(new GridLayout(0,1,0,7));
JPanel imagesHolder =new JPanel();
imagesHolder.add(images);
imagesScrollPane = new JScrollPane(imagesHolder);
imagesScrollPane.setHorizontalScrollBarPolicy(HORIZONTAL_SCROLLBAR_NEVER);
imagesScrollPane.setPreferredSize(rightSize);

/////////////////
// imageLayers //
/////////////////
String[] initialLayer = {"Background"};
imageLayers = new JList(initialLayer);
imageLayers.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
imageLayers.setLayoutOrientation(JList.VERTICAL);
imageLayers.addListSelectionListener(new LayerSelectionListener());

//////////////////
// spriteButtons //
//////////////////
spriteButtons = new JPanel();
spriteButtons.setLayout(new BoxLayout(spriteButtons,BoxLayout.Y_AXIS));
spriteButtons.add(new JLabel(" "));
temp = new JButton("Insert");
temp.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    insertImage();
  }
});
spriteButtons.add(temp);
spriteButtons.add(new JLabel(" "));
temp = new JButton("Edit");
temp.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    editImage();
  }
});
spriteButtons.add(temp);
spriteButtons.add(new JLabel(" "));
temp = new JButton("Duplicate");
temp.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    duplicateImage();
  }
});
spriteButtons.add(temp);
temp = new JButton("Copy");
temp.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    copyImage();
  }
});
spriteButtons.add(temp,BorderLayout.CENTER);
temp = new JButton("Paste");
temp.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
```

```
            pasteImage();
          }
        });
        spriteButtons.add(temp);
        temp = new JButton("Delete");
        temp.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            deleteImage();
          }
        });
        spriteButtons.add(temp);
        spriteButtons.add(new JLabel(" "));
        temp = new JButton("Move Right");
        temp.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            moveImageRight();
          }
        });
        spriteButtons.add(temp);
        temp = new JButton("Move Left");
        temp.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            moveImageLeft();
          }
        });
        spriteButtons.add(temp);

      right = new JSplitPane(JSplitPane.VERTICAL_SPLIT, imagesScrollPane, imageLayers);
      right.setResizeWeight(.75);
      right.setPreferredSize(rightSize);
      if(showRight) { this.add(right,BorderLayout.EAST); }

    ////////////
    // window //
    ////////////
    this.pack();
    this.setLocationByPlatform(true);
    this.addWindowListener(new MyWindowListener());
  }

////////////////
// debugError //
////////////////
  private void debugError(String title, String object , String message){
    if(debug){
      System.err.println("=========================================");
      System.err.println("Error: "+title);
      System.err.println("\t"+object+" "+message);
      System.err.println("=========================================");
      System.err.println();
    }
  }

////////////////////
// getMinimumSize //
////////////////////
  public Dimension getMinimumSize() {
```

```
      return this.minimumSize;
    }

  //////////
  // Main //

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
  // The main initializes all base subsystems, and creates the gui.
  // program control then passes on to the gui, and a reaction based interaction system.
  //
  // tasks the main must preform:
  //   enumerating assets
  //   loading preferences
  //   reading tools
  //   initialize actionManager
  //   create and invoke gui

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    public static void main(String[] args){

      ////////////////////
      // enumerate assets //
      ////////////////////
        fileManager = new FileManager();

      ////////////////////
      // load preferences //
      ////////////////////
        preferences = new IniInterface();
        try {
          preferences.load(new FileInputStream(fileManager.getIni(PREFERENCES_FILE)));
        } catch (Exception e) {
          System.err.println("Error: Could not load preferences");
        }

      ////////////////
      // read tools //
      ////////////////
        toolManager = null;
        try {
          toolManager = new ToolManager(fileManager.getXmls());
        } catch (NullPointerException e){
          System.err.println("Error: No tools definined. Check for valid tool xml
definitions in .\\assets");
        }

      ////////////////////////////
      // initialize actionManager //
      ////////////////////////////
        actionManager = new ActionManager();

      //////////////////////////
      // create and invoke gui //
      //////////////////////////
        JFrame.setDefaultLookAndFeelDecorated(true);
```

```
        SwingUtilities.invokeLater(new Runnable() {
          public void run() {
            try{
              UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception unused) {
              System.err.println("ERROR: System Look & Feel not supported.");
            }
            try {
              UIManager.setLookAndFeel(new SubstanceRavenLookAndFeel());
            } catch (Exception e) {
              System.err.println("ERROR: Custom Look & Feel not supported.");
            }
            window = new MySprite(INIT_TITLE);
            window.setMinimumSize(window.getMinimumSize());
            window.setExtendedState(window.getExtendedState() | JFrame.MAXIMIZED_BOTH);
            window.setVisible(true);

            window.updateSpriteComponents();
          }
        });
    }

  //////////////////////
  // MyWindowListener //

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
  // This is called whenever the window is interacted with. I.E. opened, closed, minimized,
deminimized.
  //
  // All we use it for is cleaning up when closing, but we really should sleep on iconified
to free up cpu cycles

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    private class MyWindowListener implements WindowListener{
      public void windowClosing(WindowEvent arg0) {
        try {
          preferences.save(new FileOutputStream(fileManager.getIni(PREFERENCES_FILE)));
        } catch (Exception e) {
          System.err.println("Error: Could not save preferences");
        }
        window.dispose();
        System.exit(0);
      }
      public void windowClosed      (WindowEvent arg0) {}
      public void windowActivated   (WindowEvent arg0) {}
      public void windowDeactivated (WindowEvent arg0) {}
      public void windowDeiconified (WindowEvent arg0) {}
      public void windowIconified   (WindowEvent arg0) {}
      public void windowOpened      (WindowEvent arg0) {}
    }

  //////////////////////
  // TabChangeListener //
```

```
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
  // This is called whenever the user changes from one tab to the other in Center
  // It is responsible for switching out gui components related to either tab, such as the
tools/preview panels
  //
  // Case  0: is Image tab
  // Case  1: is Animation tab
  // default: twilight zone o_O

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    private class TabChangeListener implements ChangeListener{
      public void stateChanged(ChangeEvent event) {
        switch(((JTabbedPane)event.getSource()).getSelectedIndex()){
          case 0:
            updateCurrentComponent();
            imageTabActive = true;
            window.image.repaint();

            window.remove(window.left);
            window.left = new JSplitPane(JSplitPane.VERTICAL_SPLIT, window.imageTools,
window.rgbaPanel);

            window.left.setResizeWeight(1);
            if(window.showLeft) { window.add(window.left, BorderLayout.WEST); }

            window.remove(window.right);
            window.right = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
window.imagesScrollPane, window.imageLayers);
            window.right.setResizeWeight(.75);
            right.setPreferredSize(rightSize);
            if(window.showRight) { window.add(window.right, BorderLayout.EAST); }
            break;
          case 1:
            updateCurrentComponent();
            imageTabActive = false;
            window.animation.repaint();

            window.remove(window.left);
            window.left = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
window.animationPreview,window.animColorPicker);
            window.left.setResizeWeight(1);
            if(window.showLeft) { window.add(window.left, BorderLayout.WEST); }

            window.remove(window.right);
            window.right = new JSplitPane(JSplitPane.VERTICAL_SPLIT, window.spriteButtons,
null);
            window.right.setResizeWeight(.75);
            right.setPreferredSize(rightSize);
            if(window.showRight) { window.add(window.right, BorderLayout.EAST); }
            break;
          default:
            System.err.println("You are now entering... the Twilight Zone.");
            break;
        }
```

```
    }
  }

//////////////////////////
// SlideChangeListener //
//////////////////////////
  private class SliderChangeListener implements ChangeListener{
    public void stateChanged(ChangeEvent event) {
      JSlider source = (JSlider)event.getSource();
      if (!source.getValueIsAdjusting()) {
        int fps = (int)source.getValue();
        animation.rateRefresh(fps);
      }
    }
  }

//////////////////////////
// OpacitySliderListener //
//////////////////////////
  private class OpacitySliderListener implements ChangeListener{
    public void stateChanged(ChangeEvent event) {
      JSlider source = (JSlider)event.getSource();
      if (!source.getValueIsAdjusting()) {
        int opacity = (int)source.getValue();
        opacityField.setValue(opacity);
        actionManager.setCurrentOpacity((float)opacity/100f);
      }
    }
  }

//////////////////////////
// OpacityFieldListener //
//////////////////////////
  private class OpacityFieldListener implements ChangeListener{
    public void stateChanged(ChangeEvent event) {
      JSpinner source = (JSpinner)event.getSource();
      SpinnerNumberModel sm = (SpinnerNumberModel)source.getModel();
      int opacity = sm.getNumber().intValue();
      opacitySlider.setValue(opacity);
      actionManager.setCurrentOpacity((float)opacity/100f);
    }
  }

//////////////////////////
// ToolActionListener //
//////////////////////////
  private class ToolActionListener implements ActionListener {
    private Tool tool;
    public ToolActionListener(Tool tool) {
      this.tool = tool;
    }
    public void actionPerformed(ActionEvent e) {
      MySprite.toolManager.currentTool = tool;
    }
  }

//////////////////////////
```

```
// ImagesPreviewListener //
//////////////////////////
  public class ImagePreviewListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      Component sourceButton = (Component) e.getSource();
      Component[] allButtons = sourceButton.getParent().getParent().getComponents();
      for(int i=0; i < allButtons.length ; i++ ) {
        if(!allButtons[i].equals(sourceButton.getParent())){
          allButtons[i].setBackground(null);
        } else {
          allButtons[i].setBackground(Color.WHITE);
          actionManager.setCurrentImage(i);
        }
      }
    }
  }

//////////////////////////
// LayerSelectionListener //
//////////////////////////
  public class LayerSelectionListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent event) {
      if (!event.getValueIsAdjusting()) {
        JList list = (JList)event.getSource();
        Object[] selected = list.getSelectedValues();
        for (int i=0; i<selected.length; i++) {
          //Object sel = selected[i];
        }
      }
    }
  }

//////////////////////////
// ColorPickerListener //
//////////////////////////
  public class ColorPickerListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
      if (imageTabActive) {
        actionManager.setCurrentColor(window.colorPicker.getColor());
      }
      else {
        animation.setBackgroundColor(window.animColorPicker.getColor());
      }
    }
  }

//////////////////////////
// MenuActionListener //
//////////////////////////
  private class MenuActionListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      String command =  e.getActionCommand();
      for (int i=0; i< menuDesign.length; i++){
        for( int j=0; j<menuDesign[i].length; j++){
          if(menuDesign[i][j].value == command)
          {
            if (i==0)               //File
```

```
{
  if (command.equals("New"))
    mFile_New();
  else if (command.equals("Open"))
    mFile_Open();
  else if (command.equals("Save"))
    mFile_Save();
  else if (command.equals("Save As"))
    mFile_SaveAs();
  else if (command.equals("Overwrite Current"))
    mFile_Import(false);
  else if (command.equals("Add to Current"))
    mFile_Import(true);
  else if (command.equals("Images"))
    mFile_ExportAsImages();
  else if (command.equals("Sheet"))
    mFile_ExportAsSheet();
  else if (command.equals("Close"))
    mFile_Close();
  else if (command.equals("Exit"))
    mFile_Exit();

}
else if (i==1)              //Edit
{
  if (command.equals("Undo"))
    mEdit_Undo();
  else if (command.equals("Redo"))
    mEdit_Redo();
  else if (command.equals("Cut"))
    mEdit_Cut();
  else if (command.equals("Copy"))
    mEdit_Copy();
  else if (command.equals("Paste"))
    mEdit_Paste();
  else if (command.equals("Delete"))
    mEdit_Delete();
  else if (command.equals("Preferences"))
    mEdit_Preferences();
}
else if (i==2)              //Select
{
  if (command.equals("Deselect"))
    mSelect_Deselect();
  else if (command.equals("Select All"))
    mSelect_SelectAll();
  else if (command.equals("Invert Selection"))
    mSelect_InvertSelection();
  else if (command.equals("Expand"))
    mSelect_Expand();
  else if (command.equals("Shrink"))
    mSelect_Shrink();
  else if (command.equals("To Layer"))
    mSelect_ToLayer();
}
else if (i==3)              //View
{
```

```
    if (command.equals("Zoom In"))
      mView_ZoomIn();
    else if (command.equals("Zoom Out"))
      mView_ZoomOut();
    else if (command.equals("Show Grid"))
      mView_ShowGrid();
    else if (command.equals("Transparency Image"))
      mView_TransparencyImage();
    else if (command.equals("Next Image"))
      mView_AnimationGhostNext();
    else if (command.equals("Previous Image"))
      mView_AnimationGhostPrev();
    else if (command.contains("%"))
      mView_ZoomTo(Integer.parseInt(command.replaceAll("%","")));
  }
  else if (i==4)              //Image
  {
    if (command.equals("Next") || command.equals("Previous"))
      mImage_Goto(command);
    else if (command.equals("Delete"))
      mImage_Delete();
    else if (command.equals("Canvas Size"))
      mImage_CanvasSize();
    else if (command.equals("New Layer"))
      mImage_NewLayer();
    else if (command.equals("Delete Layer"))
      mImage_DeleteLayer();
    else if (command.equals("Up") || command.equals("Down"))
      mImage_MergeLayer(command);
  }
  else if (i==5)              //Modifiers
  {
    if (command.equals("Shift"))
      mMods_Shift();
    else if (command.equals("Flip Sprite Horizontally"))
      mMods_FlipSpriteHorizontal();
    else if (command.equals("Flip Image Horizontally"))
      mMods_FlipImageHorizontal();
    else if (command.equals("Flip Layer Horizontally"))
      mMods_FlipLayerHorizontal();
    else if (command.equals("Flip Sprite Vertically"))
      mMods_FlipSpriteVertical();
    else if (command.equals("Flip Image Vertically"))
      mMods_FlipImageVertical();
    else if (command.equals("Flip Layer Vertically"))
      mMods_FlipLayerVertical();
    else if (command.equals("Rotate Sprite by Degrees"))
      mMods_RotateSpriteByDegrees();
    else if (command.equals("Rotate Sprite 90\u00b0 Clockwise"))
      mMods_RotateSprite90CW();
    else if (command.equals("Rotate Sprite 90\u00b0 Counter-clockwise"))
      mMods_RotateSprite90CCW();
    else if (command.equals("Rotate Image"))
      mMods_RotateImage();
    else if (command.equals("Rotate Layer"))
      mMods_RotateLayer();
    else if (command.equals("Scale Sprite"))
```

```
            mMods_ScaleSprite();
          else if (command.equals("Stretch Sprite"))
            mMods_StretchSprite();
          else if (command.equals("Hue/Saturation"))
            mMods_HSV();
          else if (command.equals("Opacity"))
            mMods_Opacity();
          else if (command.equals("Invert"))
            mMods_Invert();
          else if (command.equals("Erase Color"))
            mMods_EraseColor();
          else if (command.equals("Anti-Alias"))
            mMods_AntiAlias();
          else if (command.equals("Crop To Selection"))
            mMods_CropToSelection();
          else if (command.equals("Auto-Crop"))
            mMods_CropAuto();
        }
        else if (i==6)              //Animation
        {
          if (command.equals("Left") || command.equals("Right"))
            mAnimation_Cycle(command);
          else if (command.equals("Reverse"))
            mAnimation_Reverse();
          else if (command.equals("Even") || command.equals("Odd"))
            mAnimation_AddReverse(command);
          else if (command.equals("Stretch"))
            mAnimation_Stretch();
          else if (command.equals("Set Length"))
            mAnimation_SetLength();
        }
        else if (i==7)            //Window
        {
          if (j>=2 && j<=4)
            mWindow_ToggleToolBar(command);
          else if (j>=8 && j<=34)
            mWindow_ChangeSkin(command);
          else if (command.equals("About"))
            mWindow_About();
          else if (command.equals("Help"))
            mWindow_Help();
        }
        break;
      }
    }
  }

  }
}

////////////////////////////////////////////////////////////
////*************  BUTTON HANDLER METHODS  ***************////
////////////////////////////////////////////////////////////
private void insertImage()
{
  int index = actionManager.getCurrentImage();
  sprite.insertImage(index,new LayeredImage(sprite.getName() + index, sprite.getSize()));
```

```
    updateSpriteComponents();
  }
  private void editImage()
  {
    center.setSelectedIndex(0);

    updateCurrentComponent();
    imageTabActive = true;
    window.image.repaint();

    window.remove(window.left);
    window.left = new JSplitPane(JSplitPane.VERTICAL_SPLIT, window.imageTools,
window.rgbaPanel);

    window.left.setResizeWeight(1);
    if(window.showLeft) { window.add(window.left, BorderLayout.WEST); }

    window.remove(window.right);
    window.right = new JSplitPane(JSplitPane.VERTICAL_SPLIT, window.imagesScrollPane,
window.imageLayers);
    window.right.setResizeWeight(.75);
    right.setPreferredSize(rightSize);
    window.add(window.right, BorderLayout.EAST);
  }
  private void duplicateImage()
  {
    int index = actionManager.getCurrentImage();
    LayeredImage img = new LayeredImage(sprite.getImages().get(index));
    sprite.insertImage(index,img);
    updateSpriteComponents();
  }
  private void copyImage()
  {
    imageClipboard = new
LayeredImage(sprite.getImages().get(actionManager.getCurrentImage()));
  }
  private void pasteImage()
  {
    if (imageClipboard != null) {
      int index = actionManager.getCurrentImage();
      sprite.insertImage(index,imageClipboard);
      updateSpriteComponents();
    }
  }
  private void moveImageRight()
  {
    if (sprite.getImages().size() > 1) {
      int index = actionManager.getCurrentImage();
      LayeredImage img = new LayeredImage(sprite.getImages().get(index));
      sprite.deleteImage(index);
      index = index+1>(sprite.getImages().size()) ? 0 : index+1;
      actionManager.setCurrentImage(index);
      sprite.insertImage(index, img);
      updateSpriteComponents();
    }
  }
  private void moveImageLeft()
```

```
  {
    if (sprite.getImages().size() > 1) {
      int index = actionManager.getCurrentImage();
      LayeredImage img = new LayeredImage(sprite.getImages().get(index));
      sprite.deleteImage(index);
      index = index-1<0 ? sprite.getImages().size() : index-1;
      actionManager.setCurrentImage(index);
      sprite.insertImage(index, img);
      updateSpriteComponents();
    }
  }
  private void deleteImage()
  {
    if (sprite.getImages().size() > 1) {
      int index = actionManager.getCurrentImage();
        sprite.deleteImage(index);
        actionManager.setCurrentImage(Math.max(actionManager.getCurrentImage()-1,0));
        updateSpriteComponents();
    }
  }
    ////////////////////////////////////////////////////////
    ////*************  MENU HANDLER METHODS  ***************////
    ////////////////////////////////////////////////////////

    ////////////////////////////////////////////
    //-----------------FILE----------------------//
    ////////////////////////////////////////////
    private void mFile_New()
    {
      JTextField nameField = new JTextField(7);
      nameField.setText("newSprite");
      SpinnerNumberModel smw = new SpinnerNumberModel(32,1, 1000, 1);
      SpinnerNumberModel smh = new SpinnerNumberModel(32,1, 1000, 1);
      JSpinner widthField    = new JSpinner(smw);
      JSpinner heightField   = new JSpinner(smh);
      JPanel pnBoxes         = new JPanel();
      JPanel pnFields        = new JPanel();
      JPanel pnLabels        = new JPanel();
      pnLabels.setLayout(new BoxLayout(pnLabels,BoxLayout.LINE_AXIS));
      pnLabels.add(new JLabel("Name:              "));
      pnLabels.add(new JLabel("Width(px):         "));
      pnLabels.add(new JLabel("Height(px):        "));
      pnFields.setLayout(new FlowLayout());
      pnFields.add(nameField);
      pnFields.add(widthField);
      pnFields.add(heightField);
      pnBoxes.setLayout(new BoxLayout(pnBoxes,BoxLayout.Y_AXIS));
      pnBoxes.add(pnLabels);
      pnBoxes.add(pnFields);

      Object[] params = {"Enter sprite name, width, and height.",pnBoxes};
      int exportVal = JOptionPane.showConfirmDialog(this,params,"New
Sprite",JOptionPane.OK_CANCEL_OPTION);

      if(exportVal == JOptionPane.OK_OPTION) {
        int    width  = smw.getNumber().intValue();
        int    height = smh.getNumber().intValue();
```

```
        String name   = nameField.getText();
        actionManager.setCurrentImage(0);
        sprite = new Sprite(name, new Dimension(width,height));
        image.updateSprite(sprite);

        animation.updateSprite(sprite);
        updateSpriteComponents();
      }
    }
    private void mFile_Open()
    {
      JFileChooser chooser = new JFileChooser();
      FileNameExtensionFilter filter = new FileNameExtensionFilter(".spr","spr");
      chooser.setFileFilter(filter);
      int returnVal = chooser.showOpenDialog(this);

      if(returnVal == JFileChooser.APPROVE_OPTION) {
        //TODO: Michael H add here to open spr file
      }
    }
    private void mFile_Save()
    {
      //if new file
      mFile_SaveAs();
      //else ...
    }
    private void mFile_SaveAs()
    {
      //
    }
    private void mFile_Import(boolean additive) {
      JFileChooser chooser = new JFileChooser();
      FileNameExtensionFilter filter = new
FileNameExtensionFilter(".png,.jpg,.gif","png","jpg","gif");
      chooser.setFileFilter(filter);
      chooser.setMultiSelectionEnabled(true);
      chooser.setDialogTitle("Select same-sized images to import");
      int returnVal = chooser.showOpenDialog(this);

      if(returnVal == JFileChooser.APPROVE_OPTION) {

        String name = chooser.getSelectedFiles()[0].getName();//the first name
        BufferedImage[] files = new BufferedImage[chooser.getSelectedFiles().length];
        Dimension lastSize;
        try {
          for(int i=0; i<files.length; i++)
          {
            files[i] = ImageIO.read(chooser.getSelectedFiles()[i]);

            if (i>0){
              lastSize = new Dimension(files[i-1].getWidth(),files[i-1].getHeight());
              if (!lastSize.equals(new Dimension(files[i].getWidth(),files[i].getHeight())))
{
                files[i] = null;
                //change the length of the array
                BufferedImage[] temp = new BufferedImage[i];
                for(int j=0; j<temp.length;j++) {
```

```
                temp[i]=files[i];
              }
              files = temp;
              break;
              //stop the loop and remove the bad image
            }
          }
        }
        if (!additive) {
          sprite = new Sprite(
            name.substring(
              0,
              name.lastIndexOf('.')
            ),
            new Dimension(
              files[0].getWidth(),
              files[0].getHeight()
            )
          );
          sprite.setImages(files);
          images.removeAll();
          animationGrid.removeAll();
        } else {
          sprite.addImages(files);
        }
        image.updateSprite(sprite);
        animation.updateSprite(sprite);
        for(BufferedImage image : files) {
          if (new Dimension(image.getWidth(),image.getHeight()).equals(sprite.size)) {
            ImagePreview imagePreview = new
ImagePreview(images.getParent().getParent().getWidth()*0.7, image);
            imagePreview.setButtonActionListener(new ImagePreviewListener());
            images.add(imagePreview);

            ImagePreview animationPreview = new
ImagePreview(animationImageSize.getWidth(),image);
            animationPreview.setButtonActionListener(new ImagePreviewListener());

            animationGrid.add(animationPreview);

          } else {
            JOptionPane.showMessageDialog(
              this,
              "Imported images must be the same size as the current sprite.",
              "Error",
              0
            );
            break;
          }
        }
        imagesScrollPane.validate();
      } catch (IOException e) {
        System.err.println("Error: Unable to read Image file");
      }
    }
  }
  private void mFile_ExportAsImages()
```

```
    {
      LinkedList<String> checkedBoxes = new LinkedList<String>();
      String[] ext;

      JCheckBox bxPng = new JCheckBox("png",true);
      JCheckBox bxGif = new JCheckBox("gif");
      JCheckBox bxJpg = new JCheckBox("jpg");
      JPanel pnBoxes = new JPanel();
      pnBoxes.setLayout(new FlowLayout());
      pnBoxes.add(bxPng);
      pnBoxes.add(bxGif);
      pnBoxes.add(bxJpg);
      Object[] params = {"Select file types to export.",pnBoxes};
      int exportVal = JOptionPane.showConfirmDialog(this,params,"Export as
Images",JOptionPane.OK_CANCEL_OPTION);

      if(exportVal == JOptionPane.OK_OPTION) {
        if(bxPng.isSelected())
          checkedBoxes.add("png");
        if(bxGif.isSelected())
          checkedBoxes.add("gif");
        if(bxJpg.isSelected())
          checkedBoxes.add("jpg");
        ext = checkedBoxes.toArray(new String[0]);

        JFileChooser chooser = new JFileChooser();
          FileNameExtensionFilter filter = new
FileNameExtensionFilter(".spr,.png,.jpg,.gif","spr","png","jpg","gif");
          chooser.setFileFilter(filter);
          chooser.setSelectedFile(new File(sprite.getName()));
          chooser.setDialogTitle("Export Images");
        int returnVal = chooser.showSaveDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION) {
          File saveFile;

          for(int i=0; i<ext.length; i++) {

            int j=0;
              for (LayeredImage image : sprite.getImages()) {
                try {
                  if(chooser.getSelectedFile().getName().contains(".")) {
                  saveFile = new File(
                    chooser.getSelectedFile().getAbsolutePath().substring(
                      0,
                      chooser.getSelectedFile().getAbsolutePath().lastIndexOf('.')
                    )+j+"."+ext[i]
                  );
                }
                else
                  saveFile=new File(chooser.getSelectedFile().getAbsolutePath() +j+
"."+ext[i]);
                    ImageIO.write(image.image(),ext[i], saveFile);

            } catch (IOException e) {
              System.err.println("Error: Unable to write image file");
            }
                j++;
```

```
            }
          }
        }
      }

    }
    private void mFile_ExportAsSheet()
    {
      LinkedList<String> checkedBoxes = new LinkedList<String>();
      String[] ext;

      Dimension sprSize = sprite.getSize();
      int sprLength = sprite.getImages().size();
      System.out.println("sprLength:" + sprLength);
      JCheckBox bxPng = new JCheckBox("png",true);
      JCheckBox bxGif = new JCheckBox("gif");
      JCheckBox bxJpg = new JCheckBox("jpg");

      SpinnerNumberModel sm = new SpinnerNumberModel(5,1, 100, 1);
      JSpinner columnsField = new JSpinner(sm);
      JLabel columnsLabel = new JLabel("Columns");

      JPanel pnBoxes = new JPanel();
      pnBoxes.setLayout(new FlowLayout());
      pnBoxes.add(bxPng);
      pnBoxes.add(bxGif);
      pnBoxes.add(bxJpg);
      pnBoxes.add(columnsLabel);
      pnBoxes.add(columnsField);
      Object[] params = {"Select file types to export.",pnBoxes};
      int exportVal = JOptionPane.showConfirmDialog(this,params,"Export as
Sheet",JOptionPane.OK_CANCEL_OPTION);

      if(exportVal == JOptionPane.OK_OPTION) {
        if(bxPng.isSelected())
          checkedBoxes.add("png");
        if(bxGif.isSelected())
          checkedBoxes.add("gif");
        if(bxJpg.isSelected())
          checkedBoxes.add("jpg");

        int columns = sm.getNumber().intValue();
        int rows = (int)Math.ceil((double)sprLength/(double)columns);
        int width = Math.min(sprSize.width*columns,sprSize.width*sprLength);
          int height = sprSize.height*rows;
          System.out.println("--------------"+rows +"/" + columns + "--"+width + "/" +
height );

        BufferedImage sheet = new BufferedImage(width,height,BufferedImage.TYPE_4BYTE_ABGR);

        ext = checkedBoxes.toArray(new String[0]);

        JFileChooser chooser = new JFileChooser();
          FileNameExtensionFilter filter = new
FileNameExtensionFilter(".spr,.png,.jpg,.gif","spr","png","jpg","gif");
          chooser.setFileFilter(filter);
          chooser.setSelectedFile(new File(sprite.getName()));
```

```
        chooser.setDialogTitle("Export as Sheet");
      int returnVal = chooser.showSaveDialog(this);
      if(returnVal == JFileChooser.APPROVE_OPTION) {
        File saveFile;

        //get sprite images array
        BufferedImage[][] bImages = new BufferedImage[rows][columns];
        for(int i =0; i<rows;i++) {
          for(int k=0; k<columns; k++) {
            if((columns*i)+k > sprLength-1) {
              break;
            }

            bImages[i][k] = sprite.getImages().get((columns*i)+k).image();
            System.out.println("---addingImage to[" + i+"]["+k+"]:" + ((columns*i)+k));
          }
          }

        //construct sheet
        for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                  int imageY =
(int)Math.floor((double)x/(double)bImages[0][0].getWidth());
                  int imageX =
(int)Math.floor((double)y/(double)bImages[0][0].getHeight());
                  int offsetX = x%bImages[0][0].getWidth();
                  int offsetY = y%bImages[0][0].getHeight();


                  if (bImages[imageX][imageY] == null) {
                    continue;
                  }

                    sheet.setRGB( x, y, bImages[imageX][imageY].getRGB(offsetX,offsetY));
               }
            }

        for(int i=0; i<ext.length; i++) {
          //write the sheet file(s) to disk
          if(chooser.getSelectedFile().getName().contains(".")) {
          saveFile = new File(
            chooser.getSelectedFile().getAbsolutePath().substring(
              0,
              chooser.getSelectedFile().getAbsolutePath().lastIndexOf('.')
            ) + "."+ext[i]
          );
        }
        else
          saveFile=new File(chooser.getSelectedFile().getAbsolutePath() + "."+ext[i]);
          try {
            ImageIO.write(sheet,ext[i], saveFile);
          } catch (IOException e) {
            System.err.println("Error: Unable to write sprite sheet");
          }
        }
      }
    }
```

```
}
private void mFile_Close()
{/*
  sprite = new Sprite("mySprite", NEW_IMAGE_SIZE);
  actionManager.setCurrentImage(0);
  //image
  updateSpriteComponents();
  window.repaint();
  */
}
private void mFile_Exit()
{
  System.exit(0);
}


////////////////////////////////////////////////
//-----------------EDIT----------------------//
////////////////////////////////////////////////
private void mEdit_Undo()
{
  //not implemented
}
private void mEdit_Redo()
{
  //not implemented
}
private void mEdit_Cut()
{
  //not implemented
}
private void mEdit_Copy()
{
  //not implemented
}
private void mEdit_Paste()
{
  //not implemented
}
private void mEdit_Delete()
{
  //not implemented
}
private void mEdit_Preferences()
{
  //not implemented
}


////////////////////////////////////////////////
//-----------------SELECT--------------------//
////////////////////////////////////////////////
private void mSelect_Deselect()
{
  //not implemented
}
private void mSelect_SelectAll()
```

```
    {
      //not implemented
    }
    private void mSelect_InvertSelection()
    {
      //not implemented
    }
    private void mSelect_Expand()
    {
      //not implemented
    }
    private void mSelect_Shrink()
    {
      //not implemented
    }
    private void mSelect_ToLayer()
    {
      //not implemented
    }


    ////////////////////////////////////////////
    //-----------------VIEW---------------------//
    ////////////////////////////////////////////
    private void mView_ZoomIn()
    {
      //not implemented
    }
    private void mView_ZoomOut()
    {
      //not implemented
    }
    private void mView_ZoomTo(int percent)
    {
      System.out.println("Zoom To " + percent + "% - Not implemented");
    }
    private void mView_ShowGrid()
    {
      //not implemented
    }
    private void mView_TransparencyImage()
    {
      //not implemented
    }
    private void mView_AnimationGhostNext()
    {/*
      int index = actionManager.getCurrentImage();
      index=index+1>sprite.getImages().size()-1 ? 0 : index+1;

      Layer ghost = new Layer(sprite.getName() +
"ghost",sprite.getImages().get(index).image()); //neds to be reference
      ghost.updateOpacity(0.25f);
      sprite.getImages().get(actionManager.getCurrentImage()).addLayer(ghost,0);*/

    }
    private void mView_AnimationGhostPrev()
    {
```

```
  //not implemented
}


////////////////////////////////////////////
//-----------------IMAGE--------------------//
////////////////////////////////////////////
private void mImage_Goto(String which)
{
  if (sprite.getImages().size()<=1)
    return;
  int index = actionManager.getCurrentImage();
  if (which.equals("Next")) {
    index=index+1>sprite.getImages().size()-1 ? 0 : index+1;
  }
  else  {
    index=index-1<0 ? sprite.getImages().size()-1 : index-1;
  }
  actionManager.setCurrentImage(index);
  setSelectedImagePreview(index);
}
private void mImage_Delete()
{
  deleteImage();
}
private void mImage_CanvasSize()
{
  //not implemented
}
private void mImage_NewLayer()
{
  //not implemented
}
private void mImage_DeleteLayer()
{
  //not implemented
}
private void mImage_MergeLayer(String direction)
{
  //direction = "Up" or "Down"
}


////////////////////////////////////////////
//-----------------MODIFIERS----------------//
////////////////////////////////////////////
private void mMods_Shift()
{
  //not implemented
}
private void mMods_FlipSpriteVertical()
{
  //flip sprite
  sprite.flipVertical();
  image.updateSprite(sprite);
  animation.updateSprite(sprite);
```

```
  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_FlipImageVertical()
{
  //flip image
  sprite.getImages().get(actionManager.getCurrentImage()).flipVertical();

  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_FlipLayerVertical()
{
  //flip layer
  sprite.getImages().get(actionManager.getCurrentImage()).getLayers()
                    .get(actionManager.getCurrentLayer()).flipVertical();
  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_FlipSpriteHorizontal()
{
  //flip sprite
  sprite.flipHorizontal();
  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_FlipImageHorizontal()
{
  //flip image
  sprite.getImages().get(actionManager.getCurrentImage()).flipHorizontal();

  image.updateSprite(sprite);
```

```
      animation.updateSprite(sprite);

      int length = sprite.getImages().size();
      BufferedImage[] spriteImages = new BufferedImage[length];
      for(int i=0;i<length;i++){
        spriteImages[i] = sprite.getImages().get(i).image();
      }
      window.setSpriteAndUpdate(spriteImages);
    }
    private void mMods_FlipLayerHorizontal()
    {
      //flip layer
      sprite.getImages().get(actionManager.getCurrentImage()).getLayers()
                        .get(actionManager.getCurrentLayer()).flipHorizontal();
      image.updateSprite(sprite);
      animation.updateSprite(sprite);

      int length = sprite.getImages().size();
      BufferedImage[] spriteImages = new BufferedImage[length];
      for(int i=0;i<length;i++){
        spriteImages[i] = sprite.getImages().get(i).image();
      }
      window.setSpriteAndUpdate(spriteImages);
    }
    private void mMods_RotateSpriteByDegrees()
    {
      SpinnerNumberModel sm = new SpinnerNumberModel(0, 0, 360, 1);
      JSpinner degreesField = new JSpinner(sm);
      JLabel degreesLabel = new JLabel("Degrees");

      JPanel pnBoxes = new JPanel();
      pnBoxes.setLayout(new FlowLayout());
      pnBoxes.add(degreesLabel);
      pnBoxes.add(degreesField);
      int exportVal = JOptionPane.showConfirmDialog(this,pnBoxes,"Rotate by
Degrees",JOptionPane.OK_CANCEL_OPTION);

      if(exportVal == JOptionPane.OK_OPTION)
      {
        double degrees = sm.getNumber().intValue();
        double radians = Math.toRadians(degrees);

        sprite.rotate(radians);
        image.updateSprite(sprite);
        animation.updateSprite(sprite);

        int length = sprite.getImages().size();
        BufferedImage[] spriteImages = new BufferedImage[length];
        for(int i=0;i<length;i++){
          spriteImages[i] = sprite.getImages().get(i).image();
        }
        window.setSpriteAndUpdate(spriteImages);
      }
    }
    private void mMods_RotateSprite90CW()
    {
      sprite.rotate(Math.toRadians(90));
```

```
  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_RotateSprite90CCW()
{
  sprite.rotate(Math.toRadians(-90));
  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_RotateImage()
{
  sprite.getImages().get(actionManager.getCurrentImage()).rotate(Math.toRadians(90));

  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_RotateLayer()
{
  sprite.getImages().get(actionManager.getCurrentImage()).getLayers()
                    .get(actionManager.getCurrentLayer()).rotate(Math.toRadians(90));
  image.updateSprite(sprite);
  animation.updateSprite(sprite);

  int length = sprite.getImages().size();
  BufferedImage[] spriteImages = new BufferedImage[length];
  for(int i=0;i<length;i++){
    spriteImages[i] = sprite.getImages().get(i).image();
  }
  window.setSpriteAndUpdate(spriteImages);
}
private void mMods_ScaleSprite()
{
  //not implemented
}
private void mMods_StretchSprite()
{
```

```
  //not implemented
}
private void mMods_HSV()
{
  //not implemented
}
private void mMods_Opacity()
{
  //not implemented
}
private void mMods_Invert()
{
  //not implemented
  //rgb = abs(current - 255)
}
private void mMods_EraseColor()
{
  //not implemented
}
private void mMods_AntiAlias()
{
  //not implemented
}
private void mMods_CropToSelection()
{
  //not implemented
}
private void mMods_CropAuto()
{
  //not implemented
}


////////////////////////////////////////////////
//-----------------ANIMATION-----------------//
////////////////////////////////////////////////
private void mAnimation_Cycle(String direction)
{
  int numImages = sprite.getImages().size();

  BufferedImage[] cycled = new BufferedImage[numImages];

  if (direction.equals("Left")) {
      //Go ahead and grab first image and assign to index [length-1].
      cycled[numImages-1] = sprite.getImages().get(0).image();

      for(int i = 0; i < numImages-1; i++) {
       cycled[i] = sprite.getImages().get(i+1).image();
      }
    }
    else if (direction.equals("Right")) {
      //Go ahead and grab last image and assign to index [0].
      cycled[0] = sprite.getImages().get(numImages-1).image();

      for(int i = 1; i < numImages; i++) {
       cycled[i] = sprite.getImages().get(i-1).image();
      }
```

```
    }
  setSpriteAndUpdate(cycled);
}
private void mAnimation_Reverse()
{
  int numImages = sprite.getImages().size();
  BufferedImage[] reversed = new BufferedImage[numImages];
  for(int i = 0; i < numImages; i++){
    reversed[i] = sprite.getImages().get((numImages-1)-i).image();
  }
  setSpriteAndUpdate(reversed);
}

private void mAnimation_AddReverse(String reverseType)
{
  int numImages = sprite.getImages().size();
  int index = 0;

  BufferedImage[] revAdd = new BufferedImage[numImages*2];

    //populate first half : same order
    for(index = 0; index < numImages; index++) {
      revAdd[index] = sprite.getImages().get(index).image();
    }
  //populate second half : reverse order
    for(int j = index; index > 0; j++) {
    revAdd[j] = sprite.getImages().get(index-1).image();
    index--;
  }
  setSpriteAndUpdate(revAdd);
  //delete duplicate at middle.
  if(reverseType.equals("Odd")) {
    sprite.deleteImage(numImages-1);
    updateSpriteComponents();
  }
}

private void mAnimation_Stretch()
{
  //not implemented
}
private void mAnimation_SetLength()
{
  //not implemented
}


////////////////////////////////////////////////
//-----------------WINDOW--------------------//
////////////////////////////////////////////////
private void mWindow_ToggleToolBar(String window)
{
  //not implemented
}
private void mWindow_ChangeSkin(String lookAndFeel)
{
  try
```

```
      {
        UIManager.setLookAndFeel(
          "org.pushingpixels.substance.api.skin.Substance"
          + lookAndFeel.replaceAll("\\s", "")
          + "LookAndFeel"
        );
      }
      catch (Exception ex) {
        System.err.println("ERROR: Custom Look & Feel not supported.");
      }
      window.repaint();
    }
    private void mWindow_About()
    {
      JOptionPane.showMessageDialog(this, "MySprite Image Editor\n v0.0.1\n© 5GUISE
2012","About",1);
    }
    private void mWindow_Help()
    {
      //not implemented
    }


    ////////////////////
    //  misc methods     //
    ////////////////////
    private void setSpriteAndUpdate(BufferedImage[] newImages) {

      images.removeAll();
      animationGrid.removeAll();

      sprite.setImages(newImages);
      animation.updateSprite(sprite);

      for(BufferedImage image : newImages) {
      //update gui elements to reflect sprite structure

        ImagePreview imagePreview = new
ImagePreview(images.getParent().getParent().getWidth()*0.7, image);
        imagePreview.setButtonActionListener(new ImagePreviewListener());
        images.add(imagePreview);

        ImagePreview animationPreview = new
ImagePreview(animationImageSize.getWidth(),image);
        animationPreview.setButtonActionListener(new ImagePreviewListener());

        animationGrid.add(animationPreview);
      }
      setSelectedImagePreview(actionManager.getCurrentImage());
      imagesScrollPane.validate();
      animationGrid.validate();
    }

    private void updateSpriteComponents() {

      BufferedImage[] newImages = new BufferedImage[sprite.getImages().size()];
      for(int i =0; i<newImages.length;i++) {
```

```
        newImages[i] = sprite.getImages().get(i).image();
      }

      images.removeAll();
      animationGrid.removeAll();

      animation.updateSprite(sprite);

      for(BufferedImage image : newImages) {
        ImagePreview imagePreview = new
ImagePreview(images.getParent().getParent().getWidth()*0.7, image);
        imagePreview.setButtonActionListener(new ImagePreviewListener());
        images.add(imagePreview);

        ImagePreview animationPreview = new
ImagePreview(animationImageSize.getWidth(),image);
        animationPreview.setButtonActionListener(new ImagePreviewListener());

        animationGrid.add(animationPreview);
      }
      setSelectedImagePreview(actionManager.getCurrentImage());
      imagesScrollPane.validate();
      animationGrid.validate();
      animationGrid.repaint();
    }
    private void updateCurrentComponent() {
      int index = actionManager.getCurrentImage();
      BufferedImage newImage = sprite.getImages().get(index).image();
      ImagePreview preview = (ImagePreview)images.getComponent(index);
      preview.updateImage(preview.width, newImage);
      preview.setButtonActionListener(new ImagePreviewListener());

      preview = (ImagePreview)animationGrid.getComponent(index);
      preview.updateImage(preview.width, newImage);
      preview.setButtonActionListener(new ImagePreviewListener());

      setSelectedImagePreview(index);
      imagesScrollPane.validate();
      animationGrid.validate();
      animationGrid.repaint();
    }
    private void setSelectedImagePreview(int index) {
      Component[] allButtons = animationGrid.getComponents();
      Component[] allButtons2 = images.getComponents();

      for(int i=0; i < allButtons.length ; i++ ) {
        if(!(i==index)){
          allButtons[i].setBackground(null);
          allButtons2[i].setBackground(null);
        } else {
          allButtons[i].setBackground(Color.WHITE);
          allButtons2[i].setBackground(Color.WHITE);
        }
      }
    }
}
```

## 18.2 Action.java

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
//  Action.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  checks syntax and parses out literals from <action> tag in xml tool
definiions                                          //
//
//
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

package ActionManager;
import java.util.HashMap;
import java.util.Set;

////////////
// Action //
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
// This class verifies syntax of actions in xml files
//
// each instance must have command and parameters to match a Commands
enumerated type.
// The constructor requires a command and parameter to be present. They are
matched against the enumerated class
// to confirm valid syntax. The parameters values are then checked for
correctness, and given default values if not.
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
  public class Action {

    //////////////
    // Commands //
    //////////////
      public enum Commands{

        //////////////////////
        // Enumerated Types //
        //////////////////////
          PAINT  ( "paint",  new String[] { "x",          "y",
"size", "opacity",    "color"} ),
          FILL   ( "fill",   new String[] { "selection",
"opacity",    "color"} ),
          LINE   ( "line",   new String[] { "x",          "y",
"size", "opacity",    "color"} );

        ////////////////////////
        // Instance Variables //
        ////////////////////////
```

```
        private String   command;
        private String[] parameters;

      /////////////////
      // Constructor //
      /////////////////
        private Commands(String command, String[] parameters){
          this.command = command;
          this.parameters = parameters;
        }

      ///////////////////
      // matchCommand //
      ///////////////////
        public static Commands matchCommand(String command){
          for( Commands current : values() ){
            if(current.command.equalsIgnoreCase(command)){
              return current;
            }
          }
          return null;
        }

      ////////////////////
      // matchParameters //
      ////////////////////
        public static boolean matchParameters(Commands command, Set<String>
parameters){
          if(parameters.size() != command.parameters.length){
          System.err.println("not correct length");
            return false;
          }
          for(String current : command.parameters){
            if(!current.contains(current)){
              System.err.println(command.command+"::"+current);
              return false;
            }
          }
          return true;
        }
    }

  ////////////////////////
  // Instance Variables //
  ////////////////////////
    public String                  command;
    public HashMap<String,String> parameters;

  /////////////////
  // Constructor //
  /////////////////
    public Action(String command, HashMap<String,String> parameters){
      this.parameters = new HashMap<String,String>();
      if(!this.validateSyntax(command, parameters)){
        System.err.println("Error: Invalid action definition.");
      }
    }
```

```
    ////////////////////
    // validateSyntax //

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
    // This function takes the input command and parameters and validates the
syntax.
    //
    // If the command does not match any Commands.command then false is
returned.
    // If the command does match, then the parameters.keys are matched to the
Commands.parameters.keys
    // if they keys do not match then false is returned
    // if they do match, check the value of the keys.
    // if the values are correct, only then do we accept and return true.

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      public boolean validateSyntax(String command, HashMap<String,String>
parameters) {

        ///////////////////////////////////
        // check which command they want //
        ///////////////////////////////////
          switch(Commands.matchCommand(command)){

            ////////////
            // paint //
            ////////////
              case PAINT:
                this.command = "paint";
                if(Commands.matchParameters(Commands.PAINT,
parameters.keySet())){

                  ///////
                  // X //
                  ///////
                    if(parameters.containsKey("x")){
                      if(parameters.get("x").equalsIgnoreCase("currentx")){
                        this.parameters.put("x", "currentx");
                      } else {
                        try{
                          Integer.parseInt(parameters.get("x"));
                          this.parameters.put("x", parameters.get("x"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter x: "+parameters.get("x"));
                          this.parameters.put("x", "0");
                        }
                      }
                    } else {
                      System.err.println("Error: No action parameter x");
                      this.parameters.put("x", "0");
                    }

                  ///////
```

```
                    // Y //
                    ///////
                      if(parameters.containsKey("y")){
                        if(parameters.get("y").equalsIgnoreCase("currenty")){
                          this.parameters.put("y", "currenty");
                        } else {
                          try{
                                Integer.parseInt(parameters.get("y"));
                                this.parameters.put("y", parameters.get("y"));
                              } catch(NumberFormatException e){
                                System.err.println("Error: Invalid value for
action parameter y: "+parameters.get("y"));
                                this.parameters.put("y", "0");
                              }
                        }
                      } else {
                        System.err.println("Error: No action parameter y");
                        this.parameters.put("y", "0");
                      }

                    /////////
                    // size //
                    /////////
                      if(parameters.containsKey("size")){

if(parameters.get("size").equalsIgnoreCase("currentsize")){
                          this.parameters.put("size", "currentsize");
                        } else {
                        try{
                            Integer.parseInt(parameters.get("size"));
                            this.parameters.put("size",
parameters.get("size"));
                          } catch(NumberFormatException e){
                            System.err.println("Error: Invalid value for action
parameter size: "+parameters.get("size"));
                            this.parameters.put("size", "0");
                          }
                        }
                      } else {
                        System.err.println("Error: No action parameter size");
                        this.parameters.put("size", "1");
                      }

                    ///////////
                    // opacity //
                    ///////////
                      if(parameters.containsKey("opacity")){

if(parameters.get("opacity").equalsIgnoreCase("currentopacity")){
                          this.parameters.put("opacity", "currentopacity");
                        } else {
                        try{
                            Integer.parseInt(parameters.get("opacity"));
                            this.parameters.put("opacity",
parameters.get("opacity"));
                          } catch(NumberFormatException e){
```

```
                        System.err.println("Error: Invalid value for action
parameter opacity: "+parameters.get("opacity"));
                        this.parameters.put("opacity", "0");
                      }
                    }
                  } else {
                    System.err.println("Error: No action parameter
opacity");
                    this.parameters.put("opacity", "0");
                  }

                  ////////////
                  // color //
                  ////////////
                    if(parameters.containsKey("color")){

if(parameters.get("color").equalsIgnoreCase("currentcolor")){
                        this.parameters.put("color", "currentcolor");
                      } else {
                      try{
                          Integer.parseInt(parameters.get("color"),16);
                          this.parameters.put("color",
parameters.get("color"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter color: "+parameters.get("color"));
                        this.parameters.put("color", "0");
                      }
                    }
                  } else {
                    System.err.println("Error: No action parameter color");
                    this.parameters.put("color", "0");
                  }
                } else {
                  System.out.println("parameters don't match.");
                }
                break;

            ///////////
            // fill //
            ///////////
              case FILL:
                this.command = "fill";
                if(Commands.matchParameters(Commands.FILL,
parameters.keySet())){

                  ///////////////
                  // selection //
                  ///////////////
                    if(parameters.containsKey("selection")){

if(parameters.get("selection").equalsIgnoreCase("currentselection")){
                        this.parameters.put("selection", "currentselection");
                      } else {
                        System.err.println("Error: Invalid value for action
parameter selection: "+parameters.get("selection"));
                      }
```

```
                    } else {
                      System.err.println("Error: No action parameter
selection");
                      this.parameters.put("selection", "currentselection");
                    }

                  ////////////
                  // opacity //
                  ////////////
                    if(parameters.containsKey("opacity")){

if(parameters.get("opacity").equalsIgnoreCase("currentopacity")){
                        this.parameters.put("opacity", "currentopacity");
                      } else {
                      try{
                          Integer.parseInt(parameters.get("opacity"));
                          this.parameters.put("opacity",
parameters.get("opacity"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter opacity: "+parameters.get("opacity"));
                          this.parameters.put("opacity", "0");
                        }
                      }
                    } else {
                      System.err.println("Error: No action parameter
opacity");
                      this.parameters.put("opacity", "0");
                    }

                  ///////////
                  // color //
                  ///////////
                    if(parameters.containsKey("color")){
                      if(parameters.get("color").equalsIgnoreCase("color")){
                        this.parameters.put("color", "currentcolor");
                      } else {
                      try{
                          Integer.parseInt(parameters.get("color"),16);
                          this.parameters.put("color",
parameters.get("color"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter color: "+parameters.get("color"));
                          this.parameters.put("color", "0");
                        }
                      }
                    } else {
                      System.err.println("Error: No action parameter color");
                      this.parameters.put("color", "0");
                    }
                } else {
                  System.out.println("parameters don't match.");
                }
                break;

            //////////
```

```
            // line //
            //////////
              case LINE:
                this.command = "line";
                if(Commands.matchParameters(Commands.LINE,
parameters.keySet())){

                  ///////
                  // X //
                  ///////
                    if(parameters.containsKey("x")){
                      if(parameters.get("x").equalsIgnoreCase("currentx")){
                        this.parameters.put("x", "currentx");
                      } else {
                        try{
                          Integer.parseInt(parameters.get("x"));
                          this.parameters.put("x", parameters.get("x"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter x: "+parameters.get("x"));
                          this.parameters.put("x", "0");
                        }
                      }
                    } else {
                      System.err.println("Error: No action parameter x");
                      this.parameters.put("x", "0");
                    }

                  ///////
                  // Y //
                  ///////
                    if(parameters.containsKey("y")){
                      if(parameters.get("y").equalsIgnoreCase("currenty")){
                        this.parameters.put("y", "currenty");
                      } else {
                        try{
                          Integer.parseInt(parameters.get("y"));
                          this.parameters.put("y", parameters.get("y"));
                        } catch(NumberFormatException e){
                          System.err.println("Error: Invalid value for action
parameter y: "+parameters.get("y"));
                          this.parameters.put("y", "0");
                        }
                      }
                    } else {
                      System.err.println("Error: No action parameter y");
                      this.parameters.put("y", "0");
                    }

                  //////////
                  // size //
                  //////////
                    if(parameters.containsKey("size")){

if(parameters.get("size").equalsIgnoreCase("currentsize")){
                          this.parameters.put("size", "currentsize");
                        } else {
```

```
                    try{
                        Integer.parseInt(parameters.get("size"));
                        this.parameters.put("size",
parameters.get("size"));
                      } catch(NumberFormatException e){
                        System.err.println("Error: Invalid value for action
parameter size: "+parameters.get("size"));
                        this.parameters.put("size", "0");
                      }
                    }
                  } else {
                    System.err.println("Error: No action parameter size");
                    this.parameters.put("size", "1");
                  }

                 //////////////
                 // opacity //
                 //////////////
                   if(parameters.containsKey("opacity")){

if(parameters.get("opacity").equalsIgnoreCase("currentopacity")){
                      this.parameters.put("opacity", "currentopacity");
                    } else {
                    try{
                        Integer.parseInt(parameters.get("opacity"));
                        this.parameters.put("opacity",
parameters.get("opacity"));
                      } catch(NumberFormatException e){
                        System.err.println("Error: Invalid value for action
parameter opacity: "+parameters.get("opacity"));
                        this.parameters.put("opacity", "0");
                      }
                    }
                  } else {
                    System.err.println("Error: No action parameter
opacity");
                    this.parameters.put("opacity", "0");
                  }

                 ////////////
                 // color //
                 ////////////
                   if(parameters.containsKey("color")){

if(parameters.get("color").equalsIgnoreCase("currentcolor")){
                      this.parameters.put("color", "currentcolor");
                    } else {
                    try{
                        Integer.parseInt(parameters.get("color"),16);
                        this.parameters.put("color",
parameters.get("color"));
                      } catch(NumberFormatException e){
                        System.err.println("Error: Invalid value for action
parameter color: "+parameters.get("color"));
                        this.parameters.put("color", "0");
                      }
                    }
```

```
                    } else {
                      System.err.println("Error: No action parameter color");
                      this.parameters.put("color", "0");
                    }
                } else {
                  System.out.println("parameters don't match.");
                }
                break;

            //////////////
            // default //
            //////////////
              default:
                System.err.println("Error: Invalid command: "+command);
                return false;
          }
        return true;
      }
}
```

## 18.3 Action Manager.java

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
// ActionManager.java
//
// Michael Hardeman : mhardeman2@student.gsu.edu
//
//
//
// Interperates actions read from xml files into operations on the Sprite
DataStructure                                    //
//
//
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package ActionManager;
import java.awt.Color;
import DataStructures.Stack;
import MySpriteLib.GuiElement.ImageCanvas;

//////////////////
// ActionManager //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
// This class acts as a bridge between the xml parsing system, the canvas,
and DataStructures.Sprite
//
// When the user interacts with the canvas, the methods mouseDown, mouseDrag,
and mouseUp are called.
// The current tool is retrieved, and the actions stored in it are done to
the canvas.
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
  @SuppressWarnings("unused")
  public class ActionManager {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public  boolean       ready;
      public  boolean       line;
      private int           currentImage;
      private int           currentLayer;
      private Color         currentColor;
      private float         currentOpacity;
      private int           currentSize;
      private int           x;
      private int           y;
      private int           previousX;
      private int           previousY;
      private boolean       mouseDown;
      private boolean       mouseDrag;
```

```
      private boolean       mouseUp;
      private ToolManager   toolManager;
      private ImageCanvas   imageCanvas;
      private Stack<Action> actionsDone;
      private Stack<Action> actionsUndone;

    /////////////////
    // Constructor //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // When the actionManager is first instantiated, it's not ready to be
used yet. It still requires a valid
    // imageCanvas to work.
    //
    // To make the actionManager ready, you have to initialize() it

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public ActionManager() {
        this.ready = false;
      }

    ////////////////
    // initialize //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // A delayed constructor that is called when the imageCanvas is ready.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public void initalize(ToolManager toolManager, ImageCanvas
imageCanvas){
        this.ready          = true;
        this.line           = false;
        this.currentImage   = 0;
        this.currentLayer   = 0;
        this.currentColor   = Color.white;
        this.currentOpacity = 1.0f;
        this.currentSize    = 1;
        this.x              = 0;
        this.y              = 0;
        this.previousX      = 0;
        this.previousY      = 0;
        this.mouseDown      = false;
        this.mouseDrag      = false;
        this.mouseUp        = false;
        this.toolManager    = toolManager;
        this.imageCanvas    = imageCanvas;
        this.actionsDone    = new Stack<Action>();
        this.actionsUndone  = new Stack<Action>();
      }

    //////////////////////
    // getCurrentImage //
    //////////////////////
```

```
      public int getCurrentImage(){
        return currentImage;
      }

    ////////////////////
    // getCurrentLayer //
    ////////////////////
      public int getCurrentLayer(){
        return currentLayer;
      }

    ////////////////////
    // getCurrentColor //
    ////////////////////
      public Color getCurrentColor(){
        return currentColor;
      }

    ////////////////////
    // setCurrentImage //
    ////////////////////
      public void setCurrentImage(int currentImage){
        if(currentImage < 0 || currentImage >
imageCanvas.sprite.getImages().size()){
          System.err.println("Error: Tried to access image not in sprite");
          currentImage = 0;
        }
        this.currentImage = currentImage;
        this.imageCanvas.imageIsUpdated();
      }

    ////////////////////
    // setCurrentLayer //
    ////////////////////
      public void setCurrentLayer(int currentLayer){
        if(currentLayer > 0 || currentLayer <
imageCanvas.sprite.getImages().get(currentImage).getLayers().size()){
          System.err.println("Error: Tried to access layer not in image");
          currentLayer = 0;
        }
        this.currentLayer = currentLayer;
      }

    ////////////////////
    // setCurrentColor //
    ////////////////////
      public void setCurrentColor(Color color){
        this.currentColor = color;
      }

    ////////////////////////
    // setCurrentOpacity //
    ////////////////////////
      public void setCurrentOpacity(float opacity) {
        this.currentOpacity = opacity;
      }
```

```
    ///////////////
    // mouseDown //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // called when a mouse button is depressed.
    //
    // This procedure checks the current tool to see if there are actions to
be done,
    // if so it preforms them on the canvas, if not it does nothing

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public void mouseDown(int x, int y){
        if(this.ready){

          ///////////////////////////////////
          // check if actions are to be done //
          ///////////////////////////////////
            if(this.toolManager.currentTool.downActions.size() != 0){

              ////////////////////////
              // save mouse position //
              ////////////////////////
                this.previousX = this.x;
                this.previousY = this.y;
                this.x         = x;
                this.y         = y;

              ///////////////////////////////////
              //  interperate actions to be done //
              ///////////////////////////////////
                for(Action current : toolManager.currentTool.downActions){

                  ///////////
                  // paint //
                  ///////////
                    if(current.command.equals("paint")){
                      int X = 0;
                      int Y = 0;
                      int S = 1;
                      int A = 255;
                      Color CurrentColor = null;
                      if(current.parameters.get("x").equals("currentx")){
                        X = x;
                      } else {
                        X = Integer.parseInt(current.parameters.get("x"));
                      }
                      if(current.parameters.get("y").equals("currenty")){
                        Y = y;
                      } else {
                        Y = Integer.parseInt(current.parameters.get("y"));
                      }

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
```

```
                        S = Integer.parseInt(current.parameters.get("size"));
                      }

if(current.parameters.get("opacity").equals("currentopacity")){
                        A = (int)(currentOpacity*255);
                      } else {
                        A =
Integer.parseInt(current.parameters.get("opacity"));
                      }

if(current.parameters.get("color").equals("currentcolor")){
                        CurrentColor = new Color(
                          currentColor.getRed(),
                          currentColor.getGreen(),
                          currentColor.getBlue(),
                          A
                        );
                      } else {
                        int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                        int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                        int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                        CurrentColor = new Color(R,G,B,A);
                      }
                      for(int row = 0; row < S; row++){
                        for(int col = 0; col < S; col++){

imageCanvas.sprite.getImages().get(currentImage).paint(
                            currentLayer,
                            X+row,
                            Y+col,
                            CurrentColor
                          );
                        }
                      }

                    //////////
                    // fill //
                    //////////
                      } else if(current.command.equals("fill")){
                        //don't know what to do yet.
                        //selections aren't implimented in the canvas

                    //////////
                    // line //
                    //////////
                      } else if(current.command.equals("line")){
                        int S = 1;
                        int A = 255;
                        Color CurrentColor = null;
                        if(!line){
                          if(!current.parameters.get("x").equals("currentx")){
                            x = Integer.parseInt(current.parameters.get("x"));
                          }
                          if(!current.parameters.get("y").equals("currenty")){
```

```
                        y = Integer.parseInt(current.parameters.get("y"));
                      }
                      this.line = true;
                    } else {

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
                        S =
Integer.parseInt(current.parameters.get("size"));
                      }

if(current.parameters.get("opacity").equals("currentopacity")){
                        A = (int)(currentOpacity*255);
                      } else {
                        A =
Integer.parseInt(current.parameters.get("opacity"));
                      }

if(current.parameters.get("color").equals("currentcolor")){
                        CurrentColor = new Color(
                          currentColor.getRed(),
                          currentColor.getGreen(),
                          currentColor.getBlue(),
                          A
                        );
                      } else {
                        int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                        int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                        int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                        CurrentColor = new Color(R,G,B,A);
                      }

imageCanvas.sprite.getImages().get(currentImage).line(
                        currentLayer,
                        x,
                        y,
                        previousX,
                        previousY,
                        S,
                        CurrentColor
                      );
                      this.line = false;
                    }
                  } else {
                    System.err.println("Error: Invalid Command. This should
never happen...");
                  }
                }
              }
            imageCanvas.imageIsUpdated();
          }
        }
```

```
    ///////////////
    // mouseDrag //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // called when a mouse button is depressed, and the mouse is moving.
    //
    // This procedure checks the current tool to see if there are actions to
be done,
    // if so it preforms them on the canvas, if not it does nothing

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public void mouseDrag(int x, int y){
        if(this.ready){

          ///////////////////////////////////
          // check if actions are to be done //
          ///////////////////////////////////
            if(this.toolManager.currentTool.dragActions.size() != 0){

              ////////////////////////
              // save mouse position //
              ////////////////////////
                this.previousX = this.x;
                this.previousY = this.y;
                this.x         = x;
                this.y         = y;

              ///////////////////////////////////
              // interperate actions to be done //
              ///////////////////////////////////
                for(Action current : toolManager.currentTool.dragActions){

                  ///////////
                  // paint //
                  ///////////
                    if(current.command.equals("paint")){
                      int X = 0;
                      int Y = 0;
                      int S = 1;
                      int A = 255;
                      Color CurrentColor = null;
                      if(current.parameters.get("x").equals("currentx")){
                        X = x;
                      } else {
                        X = Integer.parseInt(current.parameters.get("x"));
                      }
                      if(current.parameters.get("y").equals("currenty")){
                        Y = y;
                      } else {
                        Y = Integer.parseInt(current.parameters.get("y"));
                      }

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
```

```
                        S = Integer.parseInt(current.parameters.get("size"));
                      }

if(current.parameters.get("opacity").equals("currentopacity")){
                        A = (int)(currentOpacity*255);
                      } else {
                        A =
Integer.parseInt(current.parameters.get("opacity"));
                      }

if(current.parameters.get("color").equals("currentcolor")){
                        CurrentColor = new Color(
                          currentColor.getRed(),
                          currentColor.getGreen(),
                          currentColor.getBlue(),
                          A
                        );
                      } else {
                        int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                        int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                        int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                        CurrentColor = new Color(R,G,B,A);
                      }
                      for(int row = 0; row < S; row++){
                        for(int col = 0; col < S; col++){

imageCanvas.sprite.getImages().get(currentImage).paint(
                            currentLayer,
                            X+row,
                            Y+col,
                            CurrentColor
                          );
                        }
                      }

                    //////////
                    // fill //
                    //////////
                      } else if(current.command.equals("fill")){
                        //don't know what to do yet.
                        //selections aren't implimented in the canvas

                    //////////
                    // line //
                    //////////
                      } else if(current.command.equals("line")){
                        int S = 1;
                        int A = 255;
                        Color CurrentColor = null;
                        if(!line){
                          if(!current.parameters.get("x").equals("currentx")){
                            x = Integer.parseInt(current.parameters.get("x"));
                          }
                          if(!current.parameters.get("y").equals("currenty")){
```

```
                        y = Integer.parseInt(current.parameters.get("y"));
                      }
                      this.line = true;
                    } else {

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
                        S =
Integer.parseInt(current.parameters.get("size"));
                      }

if(current.parameters.get("opacity").equals("currentopacity")){
                        A = (int)(currentOpacity*255);
                      } else {
                        A =
Integer.parseInt(current.parameters.get("opacity"));

}if(current.parameters.get("color").equals("currentcolor")){
                        CurrentColor = new Color(
                          currentColor.getRed(),
                          currentColor.getGreen(),
                          currentColor.getBlue(),
                          A
                        );
                      } else {
                        int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                        int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                        int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                        CurrentColor = new Color(R,G,B,A);
                      }

imageCanvas.sprite.getImages().get(currentImage).line(
                        currentLayer,
                        x,
                        y,
                        previousX,
                        previousY,
                        S,
                        CurrentColor
                      );
                      this.line = false;
                    }
                  } else {
                    System.err.println("Error: Invalid Command. This should
never happen...");
                  }
                }
              }
            imageCanvas.imageIsUpdated();
          }
        }

    //////////////
```

```
    // mouseUp //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
    // called when a mouse button is released.
    //
    // This procedure checks the current tool to see if there are actions to
be done,
    // if so it preforms them on the canvas, if not it does nothing

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      public void mouseUp(int x, int y){
        if(this.ready){

          ////////////////////////////////////
          // check if actions are to be done //
          ////////////////////////////////////
            if(this.toolManager.currentTool.upActions.size() != 0){

              ////////////////////////
              // save mouse position //
              ////////////////////////
                this.previousX = this.x;
                this.previousY = this.y;
                this.x         = x;
                this.y         = y;

              ////////////////////////////////////
              //  interperate actions to be done //
              ////////////////////////////////////
                for(Action current : toolManager.currentTool.upActions){

                  ///////////
                  // paint //
                  ///////////
                    if(current.command.equals("paint")){
                      int X = 0;
                      int Y = 0;
                      int S = 1;
                      int A = 255;
                      Color CurrentColor = null;
                      if(current.parameters.get("x").equals("currentx")){
                        X = x;
                      } else {
                        X = Integer.parseInt(current.parameters.get("x"));
                      }
                      if(current.parameters.get("y").equals("currenty")){
                        Y = y;
                      } else {
                        Y = Integer.parseInt(current.parameters.get("y"));
                      }

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
                        S = Integer.parseInt(current.parameters.get("size"));
```

```
                    }

if(current.parameters.get("opacity").equals("currentopacity")){
                      A = (int)(currentOpacity*255);
                    } else {
                      A =
Integer.parseInt(current.parameters.get("opacity"));
                    }

if(current.parameters.get("color").equals("currentcolor")){
                      CurrentColor = new Color(
                        currentColor.getRed(),
                        currentColor.getGreen(),
                        currentColor.getBlue(),
                        A
                      );
                    } else {
                      int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                      int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                      int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                      CurrentColor = new Color(R,G,B,A);
                    }
                    for(int row = 0; row < S; row++){
                      for(int col = 0; col < S; col++){

imageCanvas.sprite.getImages().get(currentImage).paint(
                          currentLayer,
                          X+row,
                          Y+col,
                          CurrentColor
                        );
                      }
                    }

                  //////////
                  // fill //
                  //////////
                    } else if(current.command.equals("fill")){
                      //don't know what to do yet.
                      //selections aren't implimented in the canvas

                  //////////
                  // line //
                  //////////
                    } else if(current.command.equals("line")){
                      int S = 1;
                      int A = 255;
                      Color CurrentColor = null;
                      if(!line){
                        if(!current.parameters.get("x").equals("currentx")){
                          x = Integer.parseInt(current.parameters.get("x"));
                        }
                        if(!current.parameters.get("y").equals("currenty")){
                          y = Integer.parseInt(current.parameters.get("y"));
```

```
                        }
                        this.line = true;
                      } else {

if(current.parameters.get("size").equals("currentsize")){
                        S = currentSize;
                      } else {
                        S =
Integer.parseInt(current.parameters.get("size"));
                      }

if(current.parameters.get("opacity").equals("currentopacity")){
                        A = (int)(currentOpacity*255);
                      } else {
                        A =
Integer.parseInt(current.parameters.get("opacity"));
                      }

if(current.parameters.get("color").equals("currentcolor")){
                        CurrentColor = new Color(
                          currentColor.getRed(),
                          currentColor.getGreen(),
                          currentColor.getBlue(),
                          A
                        );
                      } else {
                        int R =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 16);
                        int G =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 8 );
                        int B =
(byte)(Long.parseLong(current.parameters.get("color"),16) >>> 0 );
                        CurrentColor = new Color(R,G,B,A);
                      }

imageCanvas.sprite.getImages().get(currentImage).line(
                        currentLayer,
                        x,
                        y,
                        previousX,
                        previousY,
                        S,
                        CurrentColor
                      );
                      this.line = false;
                    }
                  } else {
                    System.err.println("Error: Invalid Command. This should
never happen...");
                  }
                }
              }
            imageCanvas.imageIsUpdated();
          }
        }
    }
```

## 18.4 Tool.java

```
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  ToolManager.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Initializes all tools in assets\tools
//
//
//
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package ActionManager;
import java.io.File;
import java.util.HashMap;
import java.util.LinkedList;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import MySpriteLib.XmlInterface;
import javax.management.modelmbean.XMLParseException;
import javax.xml.parsers.ParserConfigurationException;

//////////
// Tool //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
// Stores name, description, icon, and actions parsed from xml tool
definitions in the assets folder.
//
// These objects will be instantiated at the programs start, and the
appropriate
// data structures initialized. This each valid xml tool definition will
spawn an instance of Tool to track it's
// information, and generate buttons in the toolbar for user access to that
tool.
//
// on mouse canvas interactions, these actions will be be retrieved and
processed by the action manager.
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
  public class Tool {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public File   file;
      public String name        = "default name.";
      public String description = "default description.";
```

```
      public String icon        = "default.png";
      public LinkedList<Action> downActions;
      public LinkedList<Action> dragActions;
      public LinkedList<Action> upActions;

    /////////////////
    // Constructor //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // throws exceptions to notify the ToolManager that this tool isn't
usable.
    // ParserConfigurationException is thrown when the parser breaks
    // XMLParseException is thrown when there is an invalid tool definition

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public Tool(File file) throws XMLParseException,
ParserConfigurationException{

        this.downActions = new LinkedList<Action>();
        this.dragActions = new LinkedList<Action>();
        this.upActions   = new LinkedList<Action>();

        ////////////////////////////
        // parse tool definition //
        ////////////////////////////
        this.file = file;
        XmlInterface xmlInterface = new XmlInterface();
        xmlInterface.parse(file);
        if(xmlInterface.getRootTag().equalsIgnoreCase("tool")){
          //////////
          // name //
          //////////
            try{
              name = xmlInterface.getTagValue("name");
            } catch (NullPointerException e){
              System.err.println("No 'name' definied in "+file.getName());
            }

          /////////////////
          // description //
          /////////////////
            try{
              description = xmlInterface.getTagValue("description");
            } catch (NullPointerException e){
              System.err.println("No 'description' definied in
"+file.getName());
            }

          //////////
          // icon //
          //////////
            try{
              icon = xmlInterface.getTagValue("icon");
            } catch (NullPointerException e){
              System.err.println("No 'icon' definied in "+file.getName());
```

```
            }

          ///////////////
          // mousedown //
          ///////////////
            try{
              Node mousedown = xmlInterface.getNodeWithTag("mousedown");
              LinkedList<Node> downActionNodes =
xmlInterface.getAllNodesWithTag("action", mousedown);
              for(Node current : downActionNodes){
                NodeList children = current.getChildNodes();
                String command = null;
                HashMap<String,String> parameters = new
HashMap<String,String>();
                boolean isCommand = true;
                for(int i=0; i < children.getLength(); i++ ){
                  if(children.item(i).getNodeType() == Node.ELEMENT_NODE){
                    if(isCommand){
                      command = xmlInterface.getNodeValue(children.item(i));
                      isCommand = false;
                    } else {
                      parameters.put(children.item(i).getNodeName(),
xmlInterface.getNodeValue(children.item(i)));
                    }
                  }
                }
                if(command != null && parameters.size() != 0){
                  downActions.add(new Action(command, parameters));
                }
              }
            } catch (NullPointerException e){
              System.err.println("No 'mousedown' definied in
"+file.getName());
            }

          ///////////////
          // mousedrag //
          ///////////////
            try{
              Node mousedrag = xmlInterface.getNodeWithTag("mousedrag");
              LinkedList<Node> dragActionNodes =
xmlInterface.getAllNodesWithTag("action", mousedrag);
              for(Node current : dragActionNodes){
                NodeList children = current.getChildNodes();
                String command = new String();
                HashMap<String,String> parameters = new
HashMap<String,String>();
                boolean isCommand = true;
                for(int i=0; i < children.getLength(); i++ ){
                  if(children.item(i).getNodeType() == Node.ELEMENT_NODE){
                    if(isCommand){
                      command = xmlInterface.getNodeValue(children.item(i));
                      isCommand = false;
                    } else {
                      parameters.put(children.item(i).getNodeName(),
xmlInterface.getNodeValue(children.item(i)));
                    }
```

```
                }
              }
              if(command != null && parameters.size() != 0){
                dragActions.add(new Action(command, parameters));
              }
            }
          } catch (NullPointerException e){
            System.err.println("No 'mousedrag' definied in
"+file.getName());
          }

        ////////////
        // mouseup //
        ////////////
          try{
            Node mouseup = xmlInterface.getNodeWithTag("mouseup");
            LinkedList<Node> upActionNodes =
xmlInterface.getAllNodesWithTag("action", mouseup);
            for(Node current : upActionNodes){
              NodeList children = current.getChildNodes();
              String command = new String();
              HashMap<String,String> parameters = new
HashMap<String,String>();
              boolean isCommand = true;
              for(int i=0; i < children.getLength(); i++ ){
                if(children.item(i).getNodeType() == Node.ELEMENT_NODE){
                  if(isCommand){
                    command = xmlInterface.getNodeValue(children.item(i));
                    isCommand = false;
                  } else {
                    parameters.put(children.item(i).getNodeName(),
xmlInterface.getNodeValue(children.item(i)));
                  }
                }
              }
              if(command != null && parameters.size() != 0){
                upActions.add(new Action(command, parameters));
              }
            }
          } catch (NullPointerException e){
            System.err.println("No 'mouseup' definied in "+file.getName());
          }

      } else {
        System.err.println("Error: invalid tool definition
"+file.getName());
        throw new XMLParseException();
      }
    }
  }
```

## 18.5 ToolManager.java

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
//  ToolManager.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Initializes all tools in assets\tools
//
//
//
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

package ActionManager;
import java.io.File;
import java.util.LinkedList;
import javax.management.modelmbean.XMLParseException;
import javax.xml.parsers.ParserConfigurationException;

/////////////////
// ToolManager //
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
// on program start, this is initialized with all xml files in assets
// it creates a tool for all valid tool definitions in assets
// and throws a null pointer if there aren't any tool definitions.
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
  public class ToolManager {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public  Tool             currentTool;
      private LinkedList<Tool> tools;

    /////////////////
    // Constructor //
    /////////////////
      public ToolManager(LinkedList<File> xmlFiles) throws
NullPointerException{
        tools = new LinkedList<Tool>();
        for(File current : xmlFiles){
          try {
            Tool temp;
            temp = new Tool(current);
            tools.add(temp);
          } catch (XMLParseException e) {
            //invalid tool definition. ignore it
          } catch (ParserConfigurationException e) {
```

```
          //something wrong occurred. ignore it
        }
      }
      if(tools.size() != 0){
        currentTool = tools.element();
      } else {
        throw new NullPointerException();
      }
    }

  ///////////////
  // getTools //
  ///////////////
    public LinkedList<Tool> getTools(){
      return tools;
    }

  ///////////
  // size //
  ///////////
    public int size(){
      return tools.size();
    }
}
```

## 18.6 Layer.java

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  Layer.java
//
//  Michael Hardeman
//
//  Mhardeman2@student.gsu.edu
//
//
//
//  Represents a matrix of pixels in our Sprite.LayerdImage data structure
using a BufferedImage                        //
//
//
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package DataStructures;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.WritableRaster;
import java.nio.ByteBuffer;

///////////
// Layer //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
// Stores pixel information on the current image in the form of a
BufferedImage
//
// RGBA is a 4byte integer with 8 bits for each field.
// Operations are preformed on this data by procedures at the bottom.
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
  public class Layer {

    ///////////////
    // Constants //
    ///////////////
      public final int     MIN_COLOR_RANGE     = 0;
      public final int     MAX_COLOR_RANGE     = 255;
      public final int     MIN_DATA_RANGE      = 0;
      public final float   MIN_OPACITY_RANGE   = 0.0f;
      public final float   MAX_OPACITY_RANGE   = 1.0f;
      public final String  ERROR_MESSAGE       = "Error:  Illegal ";
      public final String  ERROR_DATA_RANGE    = "data range ";
```

```
  public final String  ERROR_ALPHA_RANGE   = "alpha  range ";
  public final String  ERROR_OPACITY_RANGE = "opacity  range ";
  public final String  ERROR_COLOR_RANGE   = "color range ";

////////////////////////
// Instance Variables //
////////////////////////
  public  String        name;
  private float         opacity;
  private BufferedImage data;

/////////////////
// Constructor //
/////////////////
  public Layer( String name, Dimension size ){
    this.name      = name;
    this.opacity   = MAX_OPACITY_RANGE;
    this.data      = new BufferedImage(
      size.width,
      size.height,
      BufferedImage.TYPE_4BYTE_ABGR
    );
    Graphics2D g2d = data.createGraphics();
    g2d.setColor(Color.white);
    g2d.fillRect(0, 0, size.width, size.height);
  }

/////////////////
// Constructor //
/////////////////
  public Layer( String name, BufferedImage data ){
    this.name    = name;
    this.opacity = MAX_OPACITY_RANGE;
    this.data    = data;
  }

/////////////////
// Constructor //
/////////////////
  public Layer( Layer layer ){
    this.name    = layer.name;
    this.opacity = layer.opacity;
    this.data    = deepCopy(layer.data);
  }

//////////////
// deepCopy //
//////////////
  private static BufferedImage deepCopy(BufferedImage bi) {
    ColorModel cm = bi.getColorModel();
    boolean isAlphaPremultiplied = cm.isAlphaPremultiplied();
    WritableRaster raster = bi.copyData(null);
    return new BufferedImage(cm, raster, isAlphaPremultiplied, null);
  }

////////////////
// updateData //
```

```
    ////////////////
      public void updateData(BufferedImage data){
        if( ( this.data.getWidth() != data.getWidth() ) || (
this.data.getHeight() != data.getHeight() )){
          System.err.println(ERROR_MESSAGE + ERROR_DATA_RANGE + "(" +
data.getWidth() + "," + data.getHeight() + ")");
        } else {
          this.data = data;
        }
      }

    ///////////////////
    // updateOpacity //
    ///////////////////
      public void updateOpacity(float opacity){
        assert(( opacity >= MIN_OPACITY_RANGE ) && ( opacity <=
MAX_OPACITY_RANGE )):
          ERROR_MESSAGE + ERROR_OPACITY_RANGE + opacity;
        float previousOpacity = this.opacity;
        this.opacity = opacity;
        for( int row = 0; row < data.getWidth(); row++ ){
          for( int col = 0; col < data.getHeight(); col++ ){
            byte red      = (byte)(data.getRGB(row, col) >>> 24);
            byte green    = (byte)(data.getRGB(row, col) >>> 16);
            byte blue     = (byte)(data.getRGB(row, col) >>> 8 );
            byte alpha    = (byte)(data.getRGB(row, col) >>> 0 );
            byte newAlpha = (byte)((alpha/previousOpacity)*opacity);
            data.setRGB( row, col, ByteBuffer.wrap(new
byte[]{red,green,blue,newAlpha}).getInt());
          }
        }
      }

    ////////////////
    // getOpacity //
    ////////////////
      public float getOpacity(){
        return this.opacity;
      }

    ///////////
    // image //
    ///////////
      public BufferedImage getData(){
        return data;
      }

    ///////////////////
    // flipVertical //
    ///////////////////
      public void flipVertical(){
        AffineTransform transformation = AffineTransform.getScaleInstance(1,
-1);
        transformation.translate(0, -data.getHeight());
        AffineTransformOp transformOperation = new
AffineTransformOp(transformation, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
        data = transformOperation.filter(data, null);
```

```
      }

    ////////////////////
    // flipHorizontal //
    ////////////////////
      public void flipHorizontal(){
        AffineTransform transformation = AffineTransform.getScaleInstance(-1,
1);
        transformation.translate(-data.getWidth(),0);
        AffineTransformOp transformOperation = new
AffineTransformOp(transformation, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
        data = transformOperation.filter(data, null);
      }

    ////////////
    // rotate //
    ////////////
      public void rotate(double radians){
        AffineTransform transformation = new AffineTransform();
        transformation.rotate(radians, data.getWidth()/2,
data.getHeight()/2);
        AffineTransformOp transformationOperation = new
AffineTransformOp(transformation, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
        data = transformationOperation.filter(data, null);
      }

    ///////////
    // paint //
    ///////////
      public void paint( int x, int y, Color color ){
        Graphics2D g2d = data.createGraphics();
        g2d.setColor(color);
        g2d.setStroke(new BasicStroke(1));
        g2d.drawLine(x, y, x, y);
        g2d.dispose();
      }

    //////////
    // line //
    //////////
      public void line( int x, int y, int x2, int y2, int s, Color color){
        Graphics2D g2d = data.createGraphics();
        g2d.setColor(color);
        g2d.setStroke(new BasicStroke(s));
        g2d.drawLine(x, y, x2, y2);
        g2d.dispose();
      }
  }
```

## 18.7 LayeredImage.java

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
//  LayeredImage.java
//
//  Michael Hardeman
//
//  Mhardeman2@student.gsu.edu
//
//
//
//  Represents an ArrayList() of Sprite.Layers in our Sprite data structure
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

package DataStructures;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.image.BufferedImage;

//////////////////
// LayeredImage //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
// Stores information on the layers that comprise an Image.
// in a linear arrayList.
//
// This class also contains procedures that manage data manipulation on all
layer levels.
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
  public class LayeredImage {

    ///////////////
    // Constants //
    ///////////////
      public final int              MIN_DATA_RANGE      = 0;
      public final float            MIN_OPACITY_RANGE   = 0.0f;
      public final float            MAX_OPACITY_RANGE   = 1.0f;
      public final String           DEFAULT_LAYER_NAME  = "Background";
      public final String           NEW_LAYER_NAME      = "New Layer";
      public final String           ERROR_MESSAGE       = "Error: Illegal ";
      public final String           ERROR_LAYER_RANGE   = "layer range ";
      public final String           ERROR_OPACITY_RANGE = "opacity range ";

    ////////////////////////
    // Instance Variables //
    ////////////////////////
```

```
  private      String           name;
  private      Dimension        size;
  private      float            opacity;
  private      ArrayList<Layer> layers;

/////////////////
// Constructor //
/////////////////
  public LayeredImage( String name, Dimension size ){
    this.name     = name;
    this.opacity  = MAX_OPACITY_RANGE;
    this.size     = size;
    this.layers   = new ArrayList<Layer>();
    this.layers.add(new Layer(DEFAULT_LAYER_NAME, size));
  }

/////////////////
// Constructor //
/////////////////
  public LayeredImage( String name, BufferedImage data ){
    this.name     = name;
    this.opacity  = MAX_OPACITY_RANGE;
    this.size     = new Dimension(data.getWidth(), data.getHeight());
    this.layers   = new ArrayList<Layer>();
    this.layers.add(new Layer(DEFAULT_LAYER_NAME, data));
  }

/////////////////
// Constructor //
/////////////////
  public LayeredImage(LayeredImage image) {
  this.name = image.name;
  this.opacity = image.opacity;
  this.size = image.size;
  this.layers = new ArrayList<Layer>();
  for(Layer layer : image.layers) {
    layers.add(new Layer(layer));
  }
  }

///////////////
// addLayer //
///////////////
  public void addLayer( int position ){
    this.layers.add(position, new Layer(NEW_LAYER_NAME, this.size));
  }

///////////////
// addLayer //
///////////////
  public void addLayer( String name, int position ){
    this.layers.add(position, new Layer(name, this.size));
  }
///////////////
// addLayer //
///////////////
  public void addLayer( Layer layer, int position ){
```

```
        this.layers.add(position, layer);
      }

    /////////////////
    // removeLayer //
    /////////////////
      public void removeLayer( int position ){
        this.layers.remove(position);
      }

    /////////////////
    // moveLayerTo //
    /////////////////
      public void moveLayerTo( int position, int destination ){
        this.layers.add(destination, layers.remove(position));
      }

    ////////////////
    // updateName //
    ////////////////
      public void updateName(String name){
        this.name = name;
      }

    ////////////////////
    // updateOpacity //
    ////////////////////
      public void updateOpacity(float opacity){
        if(( opacity < MIN_OPACITY_RANGE ) || ( opacity > MAX_OPACITY_RANGE
)){
          System.err.println(ERROR_MESSAGE + ERROR_OPACITY_RANGE + opacity);
        } else {
        float previousOpacity = this.opacity;
        this.opacity = opacity;
        for(Layer current : layers){
          if(previousOpacity != MIN_OPACITY_RANGE){
            current.updateOpacity(Math.max(current.getOpacity() -
previousOpacity,MIN_OPACITY_RANGE));
          }
          current.updateOpacity(Math.min(current.getOpacity() + opacity,
MAX_OPACITY_RANGE));
        }
        }
      }

    /////////////
    // getName //
    /////////////
      public String getName(){
        return this.name;
      }

    ////////////////
    // getOpacity //
    ////////////////
      public float getOpacity(){
        return this.opacity;
```

```
    }

  ////////////////////
  // getLayerNames //
  ////////////////////
    public String[] getLayerNames(){
      String[] output = new String[layers.size()];
      for(int i=0; i<layers.size(); i++){
        output[i] = layers.get(i).name;
      }
      return output;
    }

  ////////////////////
  // numberOfLayers //
  ////////////////////
    public ArrayList<Layer> getLayers(){
      return layers;
    }

  ////////////
  // image //
  ////////////
    public BufferedImage image(){
      BufferedImage image = layers.get(0).getData();
      for( int i = 1; i < layers.size(); i++ ){
        image = makeComposite(layers.get(i).getData(), image);
      }
      return image;
    }

  ////////////////////
  // makeComposite //
  ////////////////////
    private BufferedImage makeComposite(BufferedImage S, BufferedImage D){
      assert (S.getWidth() == D.getWidth()) && (S.getHeight() ==
D.getHeight()):
        "Error";
      BufferedImage theta = new BufferedImage(
        D.getWidth(),
        D.getHeight(),
        BufferedImage.TYPE_4BYTE_ABGR
      );
      for(int x = 0; x < theta.getWidth(); x++){
        for(int y = 0; y < theta.getHeight(); y++){
          float alpha   = ((byte)(S.getRGB(x, y) >>> 0 ))/255.0F;
          float S_red   = ((byte)(S.getRGB(x, y) >>> 24))/255.0F;
          float S_green = ((byte)(S.getRGB(x, y) >>> 16))/255.0F;
          float S_blue  = ((byte)(S.getRGB(x, y) >>> 8 ))/255.0F;
          float D_red   = ((byte)(D.getRGB(x, y) >>> 24))/255.0F;
          float D_green = ((byte)(D.getRGB(x, y) >>> 16))/255.0F;
          float D_blue  = ((byte)(D.getRGB(x, y) >>> 8 ))/255.0F;
          // T = S*a + D*(1 - a)
          float theta_red   = (S_red   * alpha) + (D_red   * (1-alpha));
          float theta_green = (S_green * alpha) + (D_green * (1-alpha));
          float theta_blue  = (S_blue  * alpha) + (D_blue  * (1-alpha));
          byte r = (byte) (theta_red   * 255.0F);
```

```
            byte g = (byte) (theta_green * 255.0F);
            byte b = (byte) (theta_blue  * 255.0F);
            byte a = (byte) 0xff;
            theta.setRGB( x, y, ByteBuffer.wrap(new
byte[]{r,g,b,a}).getInt());
          }
        }
        return theta;
      }

    //////////////////
    // flipVertical //
    //////////////////
      public void flipVertical(){
        for(Layer current : layers){
          current.flipVertical();
        }
      }

    ////////////////////
    // flipHorizontal //
    ////////////////////
      public void flipHorizontal(){
        for(Layer current : layers){
          current.flipHorizontal();
        }
      }

    ////////////
    // rotate //
    ////////////
      public void rotate(double radians){
        for(Layer current : layers){
          current.rotate(radians);
        }
      }

    ///////////
    // paint //
    ///////////
      public void paint( int selectedLayer, int x, int y, Color color ){
        this.layers.get(selectedLayer).paint(x,y,color);
      }

    //////////
    // line //
    //////////
      public void line( int selectedLayer, int x, int y, int x2, int y2, int
size, Color color){
        this.layers.get(selectedLayer).line(x,y,x2,y2,size,color);
      }
  }
```

## 18.8 Node.java

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
//
//
//  Node.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Basic Linked List data component. Generic.
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

package DataStructures;

//////////
// Node //
//////////
  public class Node<E> {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      private E       data;
      private Node<E> next;

    /////////////////
    // Constructor //
    /////////////////
      public Node(){
        this.data = null;
        this.next = null;
      }
      public Node(E data, Node<E> next){
        this.data = data;
        this.next = next;
      }

    /////////////
    // getData //
    /////////////
      public E getData(){
        return this.data;
      }

    /////////////
    // getNext //
    /////////////
      public Node<E> getNext(){
        return this.next;
```

```
    }

  //////////////
  // setData //
  //////////////
    public void setData(E data){
      this.data = data;
    }

  //////////////
  // setNext //
  //////////////
    public void setNext(Node<E> next){
      this.next = next;
    }
}
```

## 18.9 Queue.java

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  Queue.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  It's a queue. Generic.
//
//
//
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package DataStructures;

///////////
// Queue //
///////////
  public class Queue<E>{

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public  int     size;
      private Node<E> top;
      private Node<E> bottom;

    /////////////////
    // Constructor //
    /////////////////
      public Queue(){
        size = 0;
        top = null;
        bottom = null;
      }

    /////////////
    // enqueue //
    /////////////
      public void enqueue(E input){
        if(size == 0){
          top = new Node<E>(input, null);
          bottom = top;
        } else {
          bottom.setNext(new Node<E>(input,null));
          bottom = bottom.getNext();
        }
        size++;
      }
```

```
  //////////////
  // dequeue //
  //////////////
    public E dequeue() throws NullPointerException {
      if (size == 0)
        throw new NullPointerException();
      Node<E> temp = top;
      top = top.getNext();
      size--;
      return temp.getData();
    }

  ///////////
  // peek //
  ///////////
    public E peek() throws NullPointerException {
      return top.getData();
    }

  /////////
  // get //
  /////////
    public E get(int index) throws NullPointerException {
      Node<E> current = top;
      for( int i = 0; i < index; i++ ){
        current = current.getNext();
      }
      return current.getData();
    }
}
```

## 18.10 Sprite.java

```
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////
//
//
//  Sprite.java
//
//  Michael Hardeman
//
//  Mhardeman2@student.gsu.edu
//
//
//
//  The whole shebang, represents an ArrayList<LayeredImages>
//
//
//
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////

package DataStructures;
import java.util.ArrayList;
import java.awt.Dimension;
import java.awt.image.BufferedImage;

////////////
// Sprite //
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////
// Stores data on the LayeredImages that comprise a sprite in a linear
// arrayList of LayeredImages
//
// This class also contains procedures that manages data manipulation on all
the layeredImages
// in the sprite.
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////
  public class Sprite {

    ///////////////
    // Constants //
    ///////////////
      public final int    MIN_COLOR_RANGE        = 0;
      public final int    MAX_COLOR_RANGE        = 255;
      public final float  MIN_OPACITY_RANGE      = 0.0f;
      public final float  MAX_OPACITY_RANGE      = 1.0f;
      public final String DEFAULT_IMAGE_NAME     = "New Frame";
      public final String ERROR_MESSAGE       = "Error: Illegal ";
      public final String ERROR_LAYER_RANGE   = "layer range ";
      public final String ERROR_OPACITY_RANGE = "opacity range ";

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      private      String                  name;
```

```
      public       Dimension               size;
      private      float                   opacity;
      private      ArrayList<LayeredImage> images;

    /////////////////
    // Constructor //
    /////////////////
      public Sprite( String name, Dimension size ){
        this.name     = name;
        this.opacity  = MAX_OPACITY_RANGE;
        this.size     = size;
        this.images   = new ArrayList<LayeredImage>();
        this.images.add(new LayeredImage(DEFAULT_IMAGE_NAME, size));
      }

    /////////////////
    // Constructor //
    /////////////////
      public Sprite(Sprite sprite) {
        this.name = sprite.getName();
        this.size = sprite.getSize();
        this.opacity = sprite.getOpacity();
        this.images = new ArrayList<LayeredImage>();
        for(LayeredImage image : sprite.getImages()){
          this.images.add(new LayeredImage(image));
        }
      }

    ////////////////
    // updateName //
    ////////////////
      public void updateName(String name){
        this.name = name;
      }

    ///////////////////
    // updateOpacity //
    ///////////////////
      public void updateOpacity(float opacity){
        if(( opacity < MIN_OPACITY_RANGE ) || ( opacity > MAX_OPACITY_RANGE
)){
          System.err.println(ERROR_MESSAGE + ERROR_OPACITY_RANGE + opacity);
        }
        float previousOpacity = this.opacity;
        this.opacity = opacity;
        for(LayeredImage current : images){
          if(previousOpacity != MIN_OPACITY_RANGE){
            current.updateOpacity(Math.max(current.getOpacity() -
previousOpacity,MIN_OPACITY_RANGE));
          }
          current.updateOpacity(Math.min(current.getOpacity() + opacity,
MAX_OPACITY_RANGE));
        }
      }

    ///////////////
    // getImages //
```

```
    ///////////////
      public ArrayList<LayeredImage> getImages(){
        return images;
      }

    //////////////
    // getName //
    //////////////
      public String getName(){
        return name;
      }

    ////////////////
    // getOpacity //
    ////////////////
      public float getOpacity(){
        return opacity;
      }

    //////////////
    // getSize //
    //////////////
      public Dimension getSize(){
        return size;
      }

    //////////////
    //setImages//
    //////////////
      public void setImages(BufferedImage[] images){
        if (!this.size.equals( new
Dimension(images[0].getWidth(),images[0].getHeight())))
          System.out.println("---------Image size does not match sprite. Try
setting sprite size first!--------");
        else {
          this.images.clear();
          for(int i=0; i<images.length; i++) {
            this.images.add(new LayeredImage(DEFAULT_IMAGE_NAME + i,
images[i]));
          }
        }
      }
    //////////////
    //addImages//
    //////////////
      public void addImages(BufferedImage[] images){
        if (!this.size.equals( new
Dimension(images[0].getWidth(),images[0].getHeight())))
          System.out.println("---------Image size does not match sprite. Try
setting sprite size first!--------");
        else {
          for(int i=0; i<images.length; i++) {
            this.images.add(new LayeredImage(DEFAULT_IMAGE_NAME + i,
images[i]));
          }
        }
      }
```

```
  ////////////////////
  // flipVertical //
  ////////////////////
   public void flipVertical(){
      for(LayeredImage current : images){
        current.flipVertical();
      }
   }

  ////////////////////
  // flipHorizontal //
  ////////////////////
    public void flipHorizontal(){
      for(LayeredImage current : images){
        current.flipHorizontal();
      }
    }

  ////////////
  // rotate //
  ////////////
    public void rotate(double radians){
      for(LayeredImage current : images){
        current.rotate(radians);
      }
    }

  ///////////////
  //insertImage//
  ///////////////
    public void insertImage(int index, LayeredImage image) {
      this.images.add(index, image);
    }
    public void insertImages(int index, LayeredImage[] images)  {
      for (LayeredImage image : images) {
        insertImage(index, image);
      }
    }

  ///////////////
  //deleteImage//
  ///////////////
    public void deleteImage(int index) {
      this.images.remove(index);
    }
}
```

## 18.11 Stack.java

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  Stack.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  It's a Stack. Generic.
//
//
//
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package DataStructures;

///////////
// Stack //
///////////
  public class Stack<E>{

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public  int     size;
      private Node<E> top;

    /////////////////
    // Constructor //
    /////////////////
      public Stack(){
        size = 0;
        top = null;
      }

    ///////////
    // push //
    ///////////
      public void push(E input){
        top = new Node<E>(input,top);
        size++;
      }

    /////////
    // pop //
    /////////
      public E pop() throws NullPointerException {
        if (size == 0)
          throw new NullPointerException();
        Node<E> temp = top;
        top = top.getNext();
```

```
      size--;
      return temp.getData();
    }

  //////////
  // peek //
  //////////
    public E peek() throws NullPointerException {
      return top.getData();
    }
}
```

## 18.12 FileManager.java

```
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  FileManager.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Reads all ini, xml, png, and spr files in .\assets into default data
structures                                          //
//
//
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package MySpriteLib;
import java.io.File;
import java.util.LinkedList;
import java.util.ListIterator;

/////////////////
// FileManager //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
// This file is pretty self explanitory. It traverses all files in TOP_DIR
and if they match criterion specified
// in searchFolders, it adds them to the data structures ini, xml, png, or
spr.
//
// the program can then retrieve the data by using .getIni(), .getXml(),
.getPng(), and .getSpr()
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
  public class FileManager {

    ///////////////
    // Constants //
    ///////////////
      public final String TOP_DIR = ".";

    //////////////////////
    // Data Structures //
    //////////////////////
      public       LinkedList<File> ini;
      public       LinkedList<File> xml;
      public       LinkedList<File> png;
      public       LinkedList<File> spr;

    /////////////////
    // Constructor //
    /////////////////
      public FileManager() {
```

```
        ini = new LinkedList<File>();
        xml = new LinkedList<File>();
        png = new LinkedList<File>();
        spr = new LinkedList<File>();
        File start = new File(TOP_DIR);
        if(start.exists() && start.isDirectory()){
          searchFolders(start);
        }
      }

    ////////////////////
    // searchFolders //
    ////////////////////
      public void searchFolders(File current) {
        if(current.isDirectory()){
          String subDirectories[] = current.list();
          for(int i=0; i<subDirectories.length; i++){
            searchFolders(new File(current.getAbsolutePath() + File.separator
+ subDirectories[i]));
          }
        } else {
          String file = current.getName().trim().toLowerCase();
          if(file.endsWith(".ini")){
            ini.add(current);
          } else if (file.endsWith(".xml")) {
            xml.add(current);
          } else if (file.endsWith(".png")) {
            png.add(current);
          } else if (file.endsWith(".spr")) {
            spr.add(current);
          }
        }
      }

    //////////////
    // getIni //
    //////////////
      public File getIni(String name) {
        ListIterator<File> iterator = ini.listIterator();
        while(iterator.hasNext()){
          File current = iterator.next();
          if(current.getName().equals(name)){
            return current;
          }
        }
        return null;
      }

    //////////////
    // getXml //
    //////////////
      public File getXml(String name) {
        ListIterator<File> iterator = xml.listIterator();
        while(iterator.hasNext()){
          File current = iterator.next();
          if(current.getName().equals(name)){
            return current;
```

```
        }
      }
      return null;
    }

  ////////////
  // getPng //
  ////////////
    public File getPng(String name) {
      ListIterator<File> iterator = png.listIterator();
      while(iterator.hasNext()){
        File current = iterator.next();
        if(current.getName().equals(name)){
          return current;
        }
      }
      return null;
    }

  ////////////
  // getSpr //
  ////////////
    public File getSpr(String name) {
      ListIterator<File> iterator = spr.listIterator();
      while(iterator.hasNext()){
        File current = iterator.next();
        if(current.getName().equals(name)){
          return current;
        }
      }
      return null;
    }

/////////////
// getInis //
/////////////
  public LinkedList<File> getInis() {
    return ini;
  }

/////////////
// getXmls //
/////////////
  public LinkedList<File> getXmls() {
    return xml;
  }



/////////////
// getPngs //
/////////////
  public LinkedList<File> getPngs() {
    return png;
  }

/////////////
```

```
  // getSprs //
  /////////////
    public LinkedList<File> getSprs() {
      return spr;
    }
}
```

## 18.13 GuiElement.java

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
//  GuiElement.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//  Matt Gold        : mattbgold@gmail.com
//
//  Robert Jenkins   : rjenkins12@student.gsu.edu
//
//  Michael Burlison : mburlison@gmail.com
//
//  David Milum      : d.milum@yahoo.com
//
//  MySprite Specific Gui Componets and data structures
//
//
//
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

package MySpriteLib;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.event.*;
import java.awt.BasicStroke;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.Timer;
import ActionManager.ActionManager;
import ActionManager.ToolManager;
import DataStructures.Sprite;
import java.lang.Math;

////////////////
// GuiElement //
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
// This class contains custom written Gui Components, and a weird data
structure
// that is used to generate the menubar.
// That's about it.
```

```
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
  public class GuiElement {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      public String      value;
      public MenuType    type;
      public ButtonGroup group;
      public int         key;

    //////////////
    // MenuType //
    //////////////
      public enum MenuType {
        Menu,
        MenuItem,
        Checkbox,
        Radiobutton,
        Separator,
        EndMenu;
      }

    /////////////////
    // Constructor //
    /////////////////
      public GuiElement(MenuType type){
        this.value = null;
        this.type  = type;
        this.group = null;
        this.key   = 0;
      }

    /////////////////
    // Constructor //
    /////////////////
      public GuiElement(String value, MenuType type){
        this.value = value;
        this.type  = type;
        this.group = null;
        this.key   = 0;
      }

  /////////////////
  // Constructor //
  /////////////////
    public GuiElement(String value, MenuType type, int key){
      this.value = value;
      this.type  = type;
      this.group = null;
      this.key   = key;
    }

  /////////////////
  // Constructor //
  /////////////////
```

```
    public GuiElement(String value, MenuType type, ButtonGroup group){
      this.value = value;
      this.type  = type;
      this.group = group;
    }

  ////////////////////
  // Button Groups //
  ////////////////////
    static ButtonGroup skinButtonGroup = new ButtonGroup();

  ////////////////
  // MenuDesign //

//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
  // This data structure utilizes the constructors above to store
information.
  // The information is then read during program initialization by Menu
Building code in the MySprite.java
  // constructor to create a JMenuBar for the GUI.
  //
  // Every time a MenuType.Menu is encountered, it is pushed onto the stack.
  // every other MenuType is added to the top element until an
MenuType.EndMenu is encountered. Then the
  // Stack is popped.
  //
  // There is an implicit EndMenu at the end of each row in the array.

//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    public static GuiElement[][]  menuDesign =
    {
      {
        new GuiElement( "File",
MenuType.Menu,        KeyEvent.VK_F   ),
        new GuiElement( "New",
MenuType.MenuItem,    KeyEvent.VK_N   ),
        new GuiElement( "Open",
MenuType.MenuItem,    KeyEvent.VK_O   ),
        new GuiElement( "Import",
MenuType.Menu,        KeyEvent.VK_I   ),
        new GuiElement( "Overwrite Current",
MenuType.MenuItem,    KeyEvent.VK_O   ),
        new GuiElement( "Add to Current",
MenuType.MenuItem,    KeyEvent.VK_A   ),
        new GuiElement(
MenuType.EndMenu                      ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "Save",
MenuType.MenuItem,    KeyEvent.VK_S   ),
        new GuiElement( "Save As",
MenuType.MenuItem,    KeyEvent.VK_A   ),
        new GuiElement( "Export As",
MenuType.Menu,        KeyEvent.VK_E   ),
```

```
        new GuiElement( "Images",
MenuType.MenuItem,    KeyEvent.VK_I   ),
        new GuiElement( "Sheet",
MenuType.MenuItem,    KeyEvent.VK_H   ),
        new GuiElement(
MenuType.EndMenu                      ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "Close",
MenuType.MenuItem,    KeyEvent.VK_C   ),
        new GuiElement( "Exit",
MenuType.MenuItem,    KeyEvent.VK_X   ),
      },

      {
        new GuiElement( "Edit",
MenuType.Menu,        KeyEvent.VK_E   ),
        new GuiElement( "Undo",
MenuType.MenuItem,    KeyEvent.VK_U   ),
        new GuiElement( "Redo",
MenuType.MenuItem,    KeyEvent.VK_R   ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "Cut",
MenuType.MenuItem,    KeyEvent.VK_T   ),
        new GuiElement( "Copy",
MenuType.MenuItem,    KeyEvent.VK_C   ),
        new GuiElement( "Paste",
MenuType.MenuItem,    KeyEvent.VK_P   ),
        new GuiElement( "Delete",
MenuType.MenuItem,    KeyEvent.VK_D   ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "Preferences",
MenuType.MenuItem,    KeyEvent.VK_F   ),
      },

      {
        new GuiElement( "Select",
MenuType.Menu,        KeyEvent.VK_S   ),
        new GuiElement( "Deselect",
MenuType.MenuItem,    KeyEvent.VK_D   ),
        new GuiElement( "Select All",
MenuType.MenuItem,    KeyEvent.VK_A   ),
        new GuiElement( "Invert Selection",
MenuType.MenuItem,    KeyEvent.VK_I   ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "Expand",
MenuType.MenuItem,    KeyEvent.VK_E   ),
        new GuiElement( "Shrink",
MenuType.MenuItem,    KeyEvent.VK_S   ),
        new GuiElement( "To Layer",
MenuType.MenuItem,    KeyEvent.VK_L   ),
      },

      {
```

```
        new GuiElement( "View",
MenuType.Menu,         KeyEvent.VK_V   ),
        new GuiElement( "Zoom In",
MenuType.MenuItem,     KeyEvent.VK_I   ),
        new GuiElement( "Zoom Out",
MenuType.MenuItem,     KeyEvent.VK_O   ),
        new GuiElement( "Zoom To",
MenuType.Menu,         KeyEvent.VK_Z   ),
        new GuiElement( "100%",
MenuType.MenuItem,     KeyEvent.VK_1   ),
        new GuiElement( "200%",
MenuType.MenuItem,     KeyEvent.VK_2   ),
        new GuiElement( "300%",
MenuType.MenuItem,     KeyEvent.VK_3   ),
        new GuiElement( "400%",
MenuType.MenuItem,     KeyEvent.VK_4   ),
        new GuiElement( "500%",
MenuType.MenuItem,     KeyEvent.VK_5   ),
        new GuiElement( "1000%",
MenuType.MenuItem,     KeyEvent.VK_0   ),
        new GuiElement(
MenuType.EndMenu                       ),
        new GuiElement(
MenuType.Separator                     ),
        new GuiElement( "Show Grid",
MenuType.Checkbox,     KeyEvent.VK_G   ),
        new GuiElement( "Transparency Image",
MenuType.MenuItem,     KeyEvent.VK_T   ),
        new GuiElement(
MenuType.Separator                     ),
        new GuiElement( "Animation Ghosting",
MenuType.Menu,         KeyEvent.VK_A   ),
        new GuiElement( "Next Image",
MenuType.Checkbox,     KeyEvent.VK_N   ),
        new GuiElement( "Previous Image",
MenuType.Checkbox,     KeyEvent.VK_P   ),
        new GuiElement(
MenuType.EndMenu                       ),
      },

      {
        new GuiElement( "Image",
MenuType.Menu,         KeyEvent.VK_I   ),
        new GuiElement( "Goto",
MenuType.Menu,         KeyEvent.VK_G   ),
        new GuiElement( "Next",
MenuType.MenuItem,     KeyEvent.VK_N   ),
        new GuiElement( "Previous",
MenuType.MenuItem,     KeyEvent.VK_P   ),
        new GuiElement(
MenuType.EndMenu                       ),
        new GuiElement(
MenuType.Separator                     ),
        new GuiElement( "Delete",
MenuType.MenuItem,     KeyEvent.VK_D   ),
        new GuiElement( "Canvas Size",
MenuType.MenuItem,     KeyEvent.VK_C   ),
```

```
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "New Layer",
MenuType.MenuItem,    KeyEvent.VK_L   ),
        new GuiElement( "Delete Layer",
MenuType.MenuItem,    KeyEvent.VK_T   ),
        new GuiElement( "Merge Layer",
MenuType.Menu,        KeyEvent.VK_M   ),
        new GuiElement( "Up",
MenuType.MenuItem,    KeyEvent.VK_U   ),
        new GuiElement( "Down",
MenuType.MenuItem,    KeyEvent.VK_D   ),
        new GuiElement(
MenuType.EndMenu                      ),
      },

      {
          new GuiElement( "Modifiers",
MenuType.Menu,        KeyEvent.VK_M   ),
          new GuiElement( "Transform",
MenuType.Menu,        KeyEvent.VK_T   ),
          new GuiElement( "Sprite",
MenuType.Menu                         ),
          new GuiElement( "Shift",
MenuType.MenuItem,    KeyEvent.VK_S   ),
          new GuiElement( "Flip Sprite Vertically",
MenuType.MenuItem                     ),
          new GuiElement( "Flip Sprite Horizontally",
MenuType.MenuItem                     ),
          new GuiElement( "Rotate Sprite by Degrees",
MenuType.MenuItem                     ),
          new GuiElement( "Rotate Sprite 90\u00b0 Clockwise",
MenuType.MenuItem                     ),
          new GuiElement( "Rotate Sprite 90\u00b0 Counter-clockwise",
MenuType.MenuItem                     ),
          new GuiElement( "Scale Sprite",
MenuType.MenuItem,    KeyEvent.VK_C   ),
          new GuiElement( "Stretch Sprite",
MenuType.MenuItem,    KeyEvent.VK_T   ),
          new GuiElement(
MenuType.EndMenu                      ),
          new GuiElement( "Image",
MenuType.Menu                         ),
          new GuiElement( "Shift Image",
MenuType.MenuItem                     ),
          new GuiElement( "Flip Image Vertically",
MenuType.MenuItem                     ),
          new GuiElement( "Flip Image Horizontally",
MenuType.MenuItem                     ),
          new GuiElement( "Rotate Image",
MenuType.MenuItem                     ),
          new GuiElement(
MenuType.EndMenu                      ),
          new GuiElement( "Layer",
MenuType.Menu                         ),
          new GuiElement( "Shift Layer",
MenuType.MenuItem                     ),
```

```
          new GuiElement( "Flip Layer Vertically",
MenuType.MenuItem                       ),
          new GuiElement( "Flip Layer Horizontally",
MenuType.MenuItem                       ),
          new GuiElement( "Rotate Layer",
MenuType.MenuItem                       ),
          new GuiElement(
MenuType.EndMenu                        ),
          new GuiElement(
MenuType.Separator                      ),
          new GuiElement( "Hue/Saturation",
MenuType.MenuItem,    KeyEvent.VK_H     ),
          new GuiElement( "Opacity",
MenuType.MenuItem,    KeyEvent.VK_O     ),
          new GuiElement( "Invert",
MenuType.MenuItem,    KeyEvent.VK_I     ),
          new GuiElement(
MenuType.Separator                      ),
          new GuiElement( "Erase Color",
MenuType.MenuItem,    KeyEvent.VK_E     ),
          new GuiElement( "Anti-Alias",
MenuType.MenuItem,    KeyEvent.VK_A     ),
          new GuiElement(
MenuType.Separator                      ),
          new GuiElement( "Crop To Selection",
MenuType.MenuItem,    KeyEvent.VK_C     ),
          new GuiElement( "Auto-Crop",
MenuType.MenuItem,    KeyEvent.VK_U     ),
      },

      {
          new GuiElement( "Animation",
MenuType.Menu,        KeyEvent.VK_A     ),
          new GuiElement( "Cycle Images",
MenuType.Menu,        KeyEvent.VK_C     ),
          new GuiElement( "Left",
MenuType.MenuItem,    KeyEvent.VK_L     ),
          new GuiElement( "Right",
MenuType.MenuItem,    KeyEvent.VK_R     ),
          new GuiElement(
MenuType.EndMenu                        ),
          new GuiElement( "Reverse",
MenuType.MenuItem,    KeyEvent.VK_R     ),
          new GuiElement( "Add Reverse",
MenuType.Menu,        KeyEvent.VK_A     ),
          new GuiElement( "Even",
MenuType.MenuItem,    KeyEvent.VK_E     ),
          new GuiElement( "Odd",
MenuType.MenuItem,    KeyEvent.VK_O     ),
          new GuiElement(
MenuType.EndMenu                        ),
          new GuiElement( "Add Reverse",
MenuType.MenuItem,    KeyEvent.VK_A     ),
          new GuiElement( "Stretch",
MenuType.MenuItem,    KeyEvent.VK_S     ),
          new GuiElement( "Set Length",
MenuType.MenuItem,    KeyEvent.VK_L     ),
```

```
      },


      {
        new GuiElement( "Window",
MenuType.Menu,        KeyEvent.VK_W   ),
        new GuiElement( "Toolbars",
MenuType.Menu,        KeyEvent.VK_T   ),
        new GuiElement( "Instant Tools",
MenuType.Checkbox,    KeyEvent.VK_T   ),
        new GuiElement( "Color Picker",
MenuType.Checkbox,    KeyEvent.VK_C   ),
        new GuiElement( "Pallet",
MenuType.Checkbox,    KeyEvent.VK_P   ),
        new GuiElement(
MenuType.EndMenu                      ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "GUI Skin",
MenuType.Menu,        KeyEvent.VK_G   ),
        new GuiElement( "Autumn",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Business Black Steel",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Business Blue Steel",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Business",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Challenger Deep",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Creme Coffee",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Creme",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Dust Coffee",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Dust",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Emerald Dusk",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Gemini",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Graphite Aqua",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Graphite Glass",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Graphite",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Magellan",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Mariner",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Mist Aqua",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Mist Silver",
MenuType.Radiobutton, skinButtonGroup ),
```

```
        new GuiElement( "Moderate",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Nebula Brick Wall",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Nebula",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Office Black 2007",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Office Blue 2007",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Office Silver 2007",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Raven",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Sahara",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement( "Twilight",
MenuType.Radiobutton, skinButtonGroup ),
        new GuiElement(
MenuType.EndMenu                      ),
        new GuiElement(
MenuType.Separator                    ),
        new GuiElement( "About",
MenuType.MenuItem,    KeyEvent.VK_A   ),
        new GuiElement( "Help",
MenuType.MenuItem,    KeyEvent.VK_H   ),
      },
    };

  /////////////////
  // ImageCanvas //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
  // Interface between DataStrucutre.Sprite, ActionManager.ActionManager, and
the user
  //
  // in the paint() procedure, the canvas is drawn, and appropriate gridlines
are added
  // this procedure is invoked by using component.repaint().
  //
  // this class also contains mouse listening procedures that monitor mouse
movement and interaction.
  // these procedures fire procedures mouseDown(), mouseDrag(), and mouseUp()
in actionManager at the appropriate times
  // allowing DataStructure.Sprite to be changed, and handle mouse scrolling
for zooming.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    public static class ImageCanvas
      extends
        JPanel
      implements
        MouseMotionListener,
        MouseListener,
        MouseWheelListener
```

```
    {
      ///////////////
      // Constants //
      ///////////////
        private static final long          serialVersionUID = 1L;
        private        final float         MAXIMUM_ZOOM     = 64.0f;
        private        final float         MINIMUM_ZOOM     = 1.0f;
        private        final float         INITIAL_ZOOM     = 10.0f;
        private        final float         ZOOM_INCRIMENTS  = 1.0f;
        private        final float         GRID_THREASHOLD  = 4.0f;

      ////////////////////////
      // Instance Variables //
      ////////////////////////
        private       int           displacementX;
        private       int           displacementY;
        private       float         currentZoom;
        private       boolean       rightMouseDown;
        private       boolean       leftMouseDown;
        private       int           mousePreviousX;
        private       int           mousePreviousY;
        private       int           mouseCurrentX;
        private       int           mouseCurrentY;
        private       int           canvasWidth;
        private       int           canvasHeight;
        private       int           pixelWidth;
        private       int           pixelHeight;
        public        boolean       updated;
        public        Dimension     size;
        public        Sprite        sprite;
        public        ActionManager actionManager;

      /////////////////
      // Constructor //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      // Initializes all instance variables and sets them to default values.
      // Initializes actionManager, and passes relevant parameters.

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public ImageCanvas(ActionManager actionManager, ToolManager
toolManager, Sprite sprite){
          this.setDoubleBuffered (true);
          this.setFocusable      (true);
          this.setBackground     (Color.gray);

          this.displacementX  = Integer.MIN_VALUE;
          this.displacementY  = Integer.MIN_VALUE;
          this.currentZoom    = INITIAL_ZOOM;
          this.rightMouseDown = false;
          this.leftMouseDown  = false;
          this.mousePreviousX = 0;
          this.mousePreviousY = 0;
          this.mouseCurrentX  = 0;
          this.mouseCurrentY  = 0;
```

```
          this.canvasWidth    = 0;
          this.canvasHeight   = 0;
          this.pixelWidth     = 0;
          this.pixelHeight    = 0;
          this.updated        = true;

          this.sprite         = sprite;
          this.size           = sprite.size;
          this.actionManager  = actionManager;
          this.actionManager.initalize(toolManager, this);

          this.addMouseListener       (this);
          this.addMouseMotionListener (this);
          this.addMouseWheelListener  (this);
        }

      //////////////////
      // updateSprite //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      // This is called by MySprite.java every time the sprite is changed.
      // This resets zoom, and canvas position values. When displacementX and
displacementY are
      // set to Integer.MIN_VALUE, the canvas is centered by paint().

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public void updateSprite(Sprite sprite){
          this.displacementX = Integer.MIN_VALUE;
          this.displacementY = Integer.MIN_VALUE;
          this.currentZoom   = INITIAL_ZOOM;
          this.sprite        = sprite;
          this.size          = sprite.size;

this.actionManager.setCurrentImage(actionManager.getCurrentImage());

this.actionManager.setCurrentLayer(actionManager.getCurrentLayer());
          this.updated       = true;
          this.repaint();
        }

      ////////////////////
      // imageIsUpdated //
      ////////////////////
        public void imageIsUpdated(){
          this.updated = true;
          this.repaint();
        }

      ////////////
      // paint //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      // invoked with component.repaint();
      //
```

```
      // calculates all displcements, widths and heights,
      // sets all variables, then paints the sprite, then the grid overlay.

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public void paint(Graphics g){
          ////////////////
          // initialize //
          ////////////////
            super.paint(g);
            Graphics2D g2d    = (Graphics2D)g;
            this.canvasWidth  = (int)(this.size.getWidth()*currentZoom);
            this.canvasHeight = (int)(this.size.getHeight()*currentZoom);
            this.pixelWidth   = canvasWidth/this.size.width;
            this.pixelHeight  = canvasHeight/this.size.height;
            // center canvas
            if(this.displacementX == Integer.MIN_VALUE)
              this.displacementX   = (this.getWidth()/2 )-(canvasWidth/2 );
            if(this.displacementY == Integer.MIN_VALUE)
              this.displacementY   = (this.getHeight()/2)-(canvasHeight/2);

          /////////////////
          // draw sprite //
          /////////////////
            g2d.setBackground(Color.gray);
            g2d.drawImage
            (

sprite.getImages().get(actionManager.getCurrentImage()).image(),
              displacementX,
              displacementY,
              canvasWidth,
              canvasHeight,
              null
            );

          ///////////////////////
          // draw gird overlay //
          ///////////////////////
            g2d.setColor(Color.BLACK);
            g2d.setStroke(new BasicStroke(1));
            if(currentZoom >= GRID_THREASHOLD){
              for(int i=0; i < this.size.width+1; i++){
                g2d.drawLine
                (
                  displacementX + i*pixelWidth,
                  displacementY,
                  displacementX + i*pixelWidth,
                  displacementY + canvasHeight
                );
              }
              for(int j=0; j < this.size.height+1; j++){
                g2d.drawLine
                (
                  displacementX,
                  displacementY + j*pixelHeight,
                  displacementX + canvasWidth,
```

```
                displacementY + j*pixelHeight
              );
            }
          }

          ////////////////
          // clean exit //
          ////////////////
            g.dispose();
            g2d.dispose();
        }

      /////////////////
      // mouseToPixel //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
      // This function converts the mouse coordinates to sprite pixels.
      // input is mouse coordinates relative to the top left corner of the
canvas
      // output is pixel coordinates relative to the top left corner of the
sprite.
      // output is null if the mouse is not over a pixel.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
        public Dimension mouseToPixel(int mouseX, int mouseY){
          /////////////////
          // check bounds //
          /////////////////
            if(mouseX < displacementX || mouseX > displacementX+canvasWidth)
              return null;
            if(mouseY < displacementY || mouseY > displacementY+canvasHeight)
              return null;
          ////////////////////////////////
          // calculate cell coordinates //
          ////////////////////////////////
            int pixelX = (mouseX - displacementX)/pixelWidth;
            int pixelY = (mouseY - displacementY)/pixelHeight;

          return new Dimension(pixelX, pixelY);
        }

      ////////////////////
      // mouseWheelMoved //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
      // Invoked on mouse scroll
      //
      // if the mouse wheel is moved, calculate the new zoom
      //
      // ignore the commented out lines, i was playing with making the canvas
zoom to the pixel you were over with your
      // mouse
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public void mouseWheelMoved(MouseWheelEvent event) {
          if(event.getWheelRotation() == 1){
            this.currentZoom = Math.max(MINIMUM_ZOOM, currentZoom -
ZOOM_INCRIMENTS);
//            int oldPixelWidth   = pixelWidth;
//            int oldPixelHeight  = pixelHeight;//
//            this.canvasWidth  = (int)(this.size.getWidth()*currentZoom);
//            this.canvasHeight = (int)(this.size.getHeight()*currentZoom);
//            this.pixelWidth   = canvasWidth/this.size.width;
//            this.pixelHeight  = canvasHeight/this.size.height;
//            if(mouseCurrentX < displacementX+canvasWidth/2 && mouseCurrentY
< displacementY+canvasWidth/2){
//              displacementX -= (oldPixelWidth  - pixelWidth
)*this.size.width;
//              displacementY -= (oldPixelHeight -
pixelHeight)*this.size.height;
//            } else if(mouseCurrentX < displacementX+canvasWidth/2 &&
mouseCurrentY >= displacementY+canvasWidth/2){
//              displacementX -= (oldPixelWidth -
pixelWidth)*this.size.width;
//              displacementY += (oldPixelHeight -
pixelHeight)*this.size.height;
//            } else if(mouseCurrentX >= displacementX+canvasWidth/2 &&
mouseCurrentY < displacementY+canvasWidth/2){
//              displacementX += (oldPixelWidth -
pixelWidth)*this.size.width;
//              displacementY -= (oldPixelHeight -
pixelHeight)*this.size.height;
//            } else if(mouseCurrentX >= displacementX+canvasWidth/2 &&
mouseCurrentY >= displacementY+canvasWidth/2){
//              displacementX += (oldPixelWidth -
pixelWidth)*this.size.width;
//              displacementY += (oldPixelHeight -
pixelHeight)*this.size.height;
//            } else {
//              //should never happen
//            }
          }else{
            this.currentZoom = Math.min(MAXIMUM_ZOOM, currentZoom +
ZOOM_INCRIMENTS);
//            int oldPixelWidth   = pixelWidth;
//            int oldPixelHeight  = pixelHeight;
//            this.canvasWidth  = (int)(this.size.getWidth()*currentZoom);
//            this.canvasHeight = (int)(this.size.getHeight()*currentZoom);
//            this.pixelWidth   = canvasWidth/this.size.width;
//            this.pixelHeight  = canvasHeight/this.size.height;
//            if(mouseCurrentX < displacementX+canvasWidth/2 && mouseCurrentY
< displacementY+canvasWidth/2){
//              displacementX -= (pixelWidth  - oldPixelWidth
)*this.size.width;
//              displacementY -= (pixelHeight -
oldPixelHeight)*this.size.height;
//            } else if(mouseCurrentX < displacementX+canvasWidth/2 &&
mouseCurrentY >= displacementY+canvasWidth/2){
```

```
//              displacementX -= (pixelWidth  - oldPixelWidth
)*this.size.width;
//              displacementY += (pixelHeight -
oldPixelHeight)*this.size.height;
//            } else if(mouseCurrentX >= displacementX+canvasWidth/2 &&
mouseCurrentY < displacementY+canvasWidth/2){
//              displacementX += (pixelWidth  - oldPixelWidth
)*this.size.width;
//              displacementY -= (pixelHeight -
oldPixelHeight)*this.size.height;
//            } else if(mouseCurrentX >= displacementX+canvasWidth/2 &&
mouseCurrentY >= displacementY+canvasWidth/2){
//              displacementX += (pixelWidth  - oldPixelWidth
)*this.size.width;
//              displacementY += (pixelHeight -
oldPixelHeight)*this.size.height;
//            } else {
//              //should never happen
//            }
          }
          this.repaint();
        }

      ////////////////
      // mousePressed //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
      // Invoked on mouse button down
      //
      // if the user right clicks, don't do anything yet
      //
      // if the user left clicks, then inform the actionManager a tool action
needs to happen

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
        public void mousePressed(MouseEvent event) {

          this.mousePreviousX = this.mouseCurrentX;
          this.mousePreviousY = this.mouseCurrentY;
          this.mouseCurrentX  = event.getX();
          this.mouseCurrentY  = event.getY();

          if (SwingUtilities.isLeftMouseButton(event)){
            this.leftMouseDown = true;
            Dimension pixel = mouseToPixel(mouseCurrentX, mouseCurrentY);
            if(pixel != null){
              this.updated = true;
              actionManager.mouseDown(pixel.width, pixel.height);
            }
          } else if (SwingUtilities.isRightMouseButton(event)){
            this.rightMouseDown = true;
          } else {
            // middle mouse button ignore for now
          }
        }
```

```
      ////////////////////
      // mouseReleased //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      // Invoked on mouse button up
      //
      // if it's the right mouse button, don't do anything yet
      //
      // if it's the left mouse button, inform the actionManager a tool
action needs to happen

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public void mouseReleased(MouseEvent event) {

          this.mousePreviousX = this.mouseCurrentX;
          this.mousePreviousY = this.mouseCurrentY;
          this.mouseCurrentX  = event.getX();
          this.mouseCurrentY  = event.getY();

          if (SwingUtilities.isLeftMouseButton(event)){
            this.leftMouseDown = false;
            Dimension pixel = mouseToPixel(mouseCurrentX, mouseCurrentY);
            if(pixel != null){
              this.updated = true;
              actionManager.mouseUp(pixel.width, pixel.height);
            }
          } else if (SwingUtilities.isRightMouseButton(event)){
            this.rightMouseDown = false;
          } else {
            // middle mouse button ignore for now
          }
        }

      ////////////////////
      // mosueDragged //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
      // Invoked on mouse movement while a button is depressed :'(
      //
      // if it's the right mouse button, move the canvas relative to the
mouse movement
      //
      // if it's the left mouse button, don't do anything yet

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
        public void mouseDragged(MouseEvent event) {

          this.mousePreviousX = this.mouseCurrentX;
          this.mousePreviousY = this.mouseCurrentY;
          this.mouseCurrentX  = event.getX();
          this.mouseCurrentY  = event.getY();
```

```
          if(leftMouseDown && !rightMouseDown){
            Dimension previousCell = mouseToPixel(mousePreviousX,
mousePreviousY);
            Dimension currentCell  = mouseToPixel(mouseCurrentX,
mouseCurrentY );

            if(previousCell != null && currentCell != null){
              if(previousCell.width != currentCell.width ||
previousCell.height != currentCell.height){
                this.updated = true;
                actionManager.mouseDrag(currentCell.width,
currentCell.height);
              }
            }
          } else if(!leftMouseDown && rightMouseDown){
            this.displacementX += (mouseCurrentX - mousePreviousX);
            this.displacementY += (mouseCurrentY - mousePreviousY);
          } else if(leftMouseDown && rightMouseDown){
            // ignored for now
          } else {
            // should never happen
          }

          this.repaint();
        }

      ////////////
      // unused //
      ////////////
        public void mouseClicked (MouseEvent event) {}
        public void mouseEntered (MouseEvent event) {}
        public void mouseExited  (MouseEvent event) {}
        public void mouseMoved   (MouseEvent event) {}
    }

    ////////////////////
    // AnimationCanvas //
    ////////////////////
    public static class AnimationCanvas extends JPanel {

      ///////////////
      // Constants //
      ///////////////
        private static final long serialVersionUID = 1L;

      ////////////////////////
      // Instance Variables //
      ////////////////////////
        private BufferedImage[] images;
        private int             current;
        private boolean         frozen;
        private Timer           animationTimer;
        private int             width;

      /////////////////
      // Constructor //
      /////////////////
```

```
        public AnimationCanvas(Sprite sprite, int width) {
          this.width=width;
          this.setBackground(Color.GRAY);
          this.setDoubleBuffered(true);
          this.current = 0;
          this.frozen = true;
          this.animationTimer =
            new Timer
            (
              (int)(1000/30),
              new ActionListener()
              {
                public void actionPerformed(ActionEvent e){
                  nextImage();
                }
              }
            );
          updateSprite(sprite);
        }

      ////////////////////////
      // setBackgroundColor //
      ////////////////////////
        public void setBackgroundColor(Color c) {
          this.setBackground(c);
        }

      ///////////////////
      // updateSprite //
      ///////////////////
        public void updateSprite(Sprite images){
          int length = images.getImages().size();
          BufferedImage[] spriteImages = new BufferedImage[length];
          for(int i=0;i<length;i++){
            spriteImages[i] = images.getImages().get(i).image();
          }
          this.images = spriteImages;
        }

      //////////////////
      // rateRefresh //
      //////////////////
        public void rateRefresh(int fps) {
          if (fps>0) {
            animationTimer.setDelay((int)Math.ceil(1000/fps));
            if (!animationTimer.isRunning() && frozen == false) {
              animationTimer.start(); //this starts again after setting fps
to 0 without clicking stop button
            }
          }
          else{
            animationTimer.stop(); //it is imperative we keep frozen = false
in this case
          }
        }

      ///////////
```

```
// stop //
//////////
  public void stop() {
    if (frozen == true){
      current = 0;
    } else {
      frozen = true;
        animationTimer.stop();
    }
  }

//////////
// start //
//////////
  public void start() {
    if (frozen == false){
      current = 0;
    } else {
      frozen = false;
      animationTimer.start();
    }
  }

//////////////
// nextImage //
//////////////
  public void nextImage(){
    if(current >= images.length - 1){
      current = 0;
    }  else {
      current++;
    }
    repaint();
  }

//////////////
// prevImage //
//////////////
  public void prevImage(){
    if(current == 0){
      current = images.length - 1;
    }  else {
      current--;
    }
    repaint();
    System.out.println(current);
  }

//////////
// paint //
//////////
  public void paint(Graphics g){
    super.paint(g);
    Graphics2D g2d = (Graphics2D)g;
    double sc =width/(double)images[current].getWidth();
    if (images != null) {
```

```
            g2d.drawImage(images[current], 0, 0, width,
(int)(images[current].getHeight()*sc), this);
          }

          g2d.dispose();
        }
    }


    //////////////////
    // ImagePreview //
    //////////////////
      public static class ImagePreview extends JPanel {

      ///////////////
      // Constants //
      ///////////////
        private static final long serialVersionUID = 1L;

        ////////////////////////
        // Instance Variables //
        ////////////////////////
          private JPanel  icon;
          private JButton wrapper;
          public  double  width;

        /////////////////
        // Constructor //
        /////////////////
          public ImagePreview(double width, BufferedImage image) {
            this.width = width;

            this.icon  = new JPanel();
            this.icon.setLayout(new GridLayout(0,1));
            double iscale = ((double)width/(double)image.getWidth());
            this.icon.add(new JLabel(scale(image,iscale)));

            this.wrapper = new JButton();
            this.wrapper.add                  (icon);
            this.wrapper.setOpaque            (false);
            this.wrapper.setBorderPainted     (false);
            this.wrapper.setContentAreaFilled (false);
            this.add(wrapper);
          }

        ////////////////////////////
        // setButtonActionListener //
        ////////////////////////////
          public void setButtonActionListener(ActionListener actionListener){
            wrapper.addActionListener(actionListener);
          }

        /////////////////
        // updateImage //
        /////////////////
          public void updateImage(double width, BufferedImage image) {
            this.removeAll();
```

```
            this.width = width;

            this.icon = new JPanel();
            this.icon.setLayout(new GridLayout(0,1));
            double iscale = ((double)width/(double)image.getWidth());
            this.icon.add(new JLabel(scale(image,iscale)));

            this.wrapper = new JButton();
            this.wrapper.add                   (icon);
            this.wrapper.setOpaque             (false);
            this.wrapper.setBorderPainted      (false);
            this.wrapper.setContentAreaFilled (false);
            this.add(wrapper);
          }

        ///////////
        // scale //
        ///////////
          private ImageIcon scale(Image src, double scale) {
            int w = (int)(scale*src.getWidth(this));
            int h = (int)(scale*src.getHeight(this));
            int type = BufferedImage.TYPE_INT_RGB;
            BufferedImage dst = new BufferedImage(w, h, type);
            Graphics2D g2d = dst.createGraphics();
            g2d.drawImage(src, 0, 0, w, h, this);
            g2d.dispose();
            return new ImageIcon(dst);
          }

    }
}
```

## 18.14 IniInterface.java

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//
//
//  IniInterface.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Just a front end for properties. I didn't actually have to do much.
Thought it would be harder.                    //
//
//
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

package MySpriteLib;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

//////////////////
// IniInterface //
//////////////////
  public class IniInterface{

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      private Properties properties;

    /////////////////
    // Constructor //
    /////////////////
      public IniInterface(){
        this.properties = new Properties();
      }

    ///////////
    // load //
    ///////////
      public void load(FileInputStream inStream){
        try {
          this.properties.load(inStream);
          inStream.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }

    ///////////
    // save //
```

```
    //////////
      public void save(FileOutputStream outStream){
        try {
          this.properties.store(outStream, "Auto-Generated Configuration
File");
          outStream.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }

    //////////////
    // getValue //
    //////////////
      public String getValue(String key){
        if(this.properties.containsKey(key)){
          return (String)this.properties.get(key);
        } else {
          return "Key not found";
        }
      }

    ////////////
    // addKey //
    ////////////
      public void addKey(String key, String value){
        if(!this.properties.containsKey(key)){
          this.properties.put(key, value);
        } else {
          setKey(key, value);
        }
      }

    ////////////
    // setKey //
    ////////////
      public void setKey(String key, String value){
        if(this.properties.containsKey(key)){
          this.properties.setProperty(key, value);
        }
      }

    ///////////////
    // removeKey //
    ///////////////
      public void removeKey(String key){
        if(this.properties.containsKey(key)){
          this.properties.remove(key);
        }
      }

    ///////////
    // clear //
    ///////////
      public void clear(){
        this.properties.clear();
      }
```

```
  ////////////
  // print //
  ////////////
    public void print(){
      System.out.println();
      this.properties.list(System.out);
      System.out.println();
    }
}
```

## 18.15 XmlInterface.java

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
//  XmlInterface.java
//
//  Michael Hardeman : Mhardeman2@student.gsu.edu
//
//
//
//  Abstracts XmlParsing Library with standard function calls
//
//
//
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

package MySpriteLib;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

//////////////////
// XmlInterface //
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
// I am not a fan of java's docmuent Builder.
// I'm baffled by it's obtuse design, and wonder why it parses out such
usless information as
// blank spaces and new lines, and why it's structure is so nested and nodes
don't have relevant data in them,
// the data is stored in children nodes along with other tags and blank
spaces and junk.
//
// This tries to remedy that by making getting tags, and info out of tags
easier.
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
  public class XmlInterface {

    ////////////////////////
    // Instance Variables //
    ////////////////////////
      private Node                  rootNode;
      private Document              currentDocument;
      private DocumentBuilder       documentBuilder;
```

```
      private DocumentBuilderFactory documentBuilderFactory;

    /////////////////
    // Constructor //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // God damit, who thought that it was a good idea to have a
documentBuilder facotry, that you use to build
    // documentBuilders, that you then use to buildDocuments... That's some
of the most non-sensical design i've
    // ever seen.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public XmlInterface() throws ParserConfigurationException{
        try {
          rootNode        = null;
          currentDocument = null;
          documentBuilderFactory = DocumentBuilderFactory.newInstance();
          documentBuilder = documentBuilderFactory.newDocumentBuilder();
        } catch (ParserConfigurationException e) {
          System.err.println("Error: Parser unable to be configured");
          throw e; // notify calling program of error
        }
      }

    ///////////
    // parse //

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // Asswiper myAssWiper = new Asswiper(this.ass);
    // myAssWiper.wipeAss();
    //
    // that's basically the jist of this function. Goddamit java, you make
things too easy. Nobody will ever learn
    // anything using you because you do everything for them.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
      public void parse(File file) {
        try {
          if(file.exists() && file.canRead() && file.isFile() &&
file.getName().endsWith(".xml")) {
            currentDocument = documentBuilder.parse(file);
            NodeList nodeList = currentDocument.getChildNodes();
            for(int i=0; i<nodeList.getLength(); i++){
              Node current = nodeList.item(i);
              if(current.getNodeType() == Node.ELEMENT_NODE){
                rootNode = current;
                break;
              }
            }
          }
        } catch(IOException e){
          e.printStackTrace();
```

```
        } catch (SAXException e) {
          e.printStackTrace();
        }
      }


////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
    // These are getters that try and extract data from the retarded tree
structure that the dom parser makes
    // They attempt to discards dummy data, which for some reason, is
included in the tree; and
    // they extract the info contained in the tags from the obtuse design.

////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////

    ////////////////
    // getRootTag //
    ////////////////
      public String getRootTag(){
        return rootNode.getNodeName();
      }

    ////////////////////////
    // getAllNodesWithTag //
    ////////////////////////
      public LinkedList<Node> getAllNodesWithTag(String tag){
        LinkedList<Node> output = new LinkedList<Node>();
        search(output, tag, rootNode);
        return output;
      }

    ////////////////////////
    // getAllNodesWithTag //
    ////////////////////////
      public LinkedList<Node> getAllNodesWithTag(String tag, Node start){
          LinkedList<Node> output = new LinkedList<Node>();
        search(output, tag, start);
        return output;
      }

    ////////////
    // search //
    ////////////
      public void search(LinkedList<Node> nodes, String tag, Node current){
        NodeList children = current.getChildNodes();
        for(int i=0; i<children.getLength(); i++){
          if(children.item(i).getNodeType() == Node.ELEMENT_NODE){
            if(children.item(i).getNodeName().equals(tag)){
              nodes.add(children.item(i));
            }
            search(nodes, tag, children.item(i));
          }
        }
      }
```

```
    ////////////////////
    // getNodeWithTag //
    ////////////////////
      public Node getNodeWithTag(String tag, Node start){
        return search(tag, start);
      }

    ////////////////////
    // getNodeWithTag //
    ////////////////////
      public Node getNodeWithTag(String tag){
        return search(tag, rootNode);
      }

    ////////////
    // search //
    ////////////
      public Node search(String tag, Node current){
        NodeList children = current.getChildNodes();
        for(int i=0; i<children.getLength(); i++){
          if(children.item(i).getNodeType() == Node.ELEMENT_NODE){
            if(children.item(i).getNodeName().equals(tag)){
              return children.item(i);
            }
            search(tag, children.item(i));
          }
        }
        return null;
      }

    /////////////////
    // getTagValue //
    /////////////////
      public String getTagValue(String tag) {
        NodeList list =
((Element)rootNode).getElementsByTagName(tag).item(0).getChildNodes();
        Node value = (Node) list.item(0);
        return value.getNodeValue();
      }

    /////////////////
    // getTagValue //
    /////////////////
      public String getTagValue(String tag, Node node) {
        Element element = (Element) node;
        NodeList list =
element.getElementsByTagName(tag).item(0).getChildNodes();
        Node value = (Node) list.item(0);
        return value.getNodeValue();
      }

    ///////////////////
    // getNodeValue //
    ///////////////////
      public String getNodeValue(Node node){
        Element element = (Element) node;
        NodeList list = element.getChildNodes();
```

```
        Node value = (Node) list.item(0);
        return value.getNodeValue();
      }

    ///////////////
    // printNode //
    ///////////////
      public void printNode(Node node){
        System.out.println(node.getNodeName()+" : "+getNodeValue(node));
      }
}
```

## 19.0 Project Legacy

There were many things about this project, that our group enjoyed overcoming. Working on a non trivial project, implementing interesting image algorithms, learning to handle errors with grace to name a few, however, It was not the things we enjoyed that taught us the most; it was those things we disliked doing that were the best for us. One hard lesson we learned early on, was that things will never turn out exactly as you design it. Our project initially included some interesting features like multi-threading and a java OpenGL binding known as JOGL. While cool, neither of these features made it into the final product, as both added unnecessary complexity, without adding speed or reactivity. We learned that we had to keep our designs fluid from this, and began to change them as needed throughout the implementation of our final program.

The second lesson we pulled from this project, is to keep it simple stupid. In the first design and implementation of the Sprite data structure, there were functions that exported the sprite at different zoom levels, with and without grids. This was backwards design, a data structure should be simple, and should contain only the data, and functions that perform operations on it. Formatting the data for display is something better handled by the gui. Keep it simple.

## 20.0 Glossary

| Term | Definition |
|---|---|
| animation | The rapid display of a sequence of images of 2-D or 3-D artwork or model positions in order to create an illusion of movement. |
| canvas | The visible area or workspace of an image. |
| color pallet | A given, finite set of colors for the management of digital images. |
| color picker | A utility used to preview/choose colors or create color schemes. |
| coordinates | Any of a set of numbers used in specifying the location of a pixel on the canvas. |
| flip | flip image on either a horizontal or vertical axis |
| frame | A frame is one of the many still images which compose a complete moving sprite |
| frame rate | The rate at which individual images (frames) in a sprite sequence are displayed when animated. Frame rate is measured in frames per second (fps). |
| graphical user interface | a user interface based on graphics (icons and pictures and menus) instead of text; uses a mouse as well as a keyboard as an input device |
| GUI | see Graphical User Interface |
| independent game | Video games created by individuals or small teams without video game publisher financial support. |
| indie game | See *independent game* |
| JFileChooser | provides a simple mechanism for the user to choose a file. |
| layer | a discrete container for pieces of an image which can be superimposed on other layers in any user specified order to create one final image. |
| matrix | A data structure with corresponding number values in which pixel data is stored. |
| MPSE | MyPixel Sprite Editor |
| opacity | The condition of lacking transparency or translucence. |
| pallet | |
| pixel | A minute, discreet, and often square area of an image. A pixel can be assigned a RGB value to determine its color when displayed on screen. |
| pixel art | a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level. |
| raster graphic | A data structure representing a generally rectangular grid of pixels, or points of color, viewable via a monitor, paper, or other display medium. |
| RGB value | Stands for the red, green, blue color space where colors are specified as triples of numbers typically between 0 and 100. The number triple gives the intensities for the red, green, and blue components of a color. The RGB value (100, 0, 0) would indicate red, for example. |
| rotate | rotate image clockwise a user specified amount of degrees |
| scale | scale image to a user specified percentage |

| | |
|---|---|
| shift | shift image by a user specified amount of pixels in a user specified direction |
| sprite | A computer graphic that may be moved on-screen and otherwise manipulated as a single entity. |
| sprite animation | See *animation* |
| square matrix | A two-dimensional matrix. |
| toolbar | A strip of GUI buttons labeled with descriptive icons used to perform specific functions. |
| transparency | An adjustable level of image visibility that ranges from invisible to 100% visible. |
| x,y coordinates | See *coordinates* |
| xml | A flexible text format for creating structured computer documents in machine-readable form |
| xml file | see XML |

## 21.0 Appendix

*User Manual*

*Prototype Demo*

*Software Engineering Course Project Contract*