Team 2 Goofy Lights Editor
CS-383

Nick Krenowicz
Program Flow Leader

Paul Martin
GUI Leader

Tim Sonnen
File Master

Kevin Dorscher
Linked List Librarian

Joe Carter
Developer

Lise Welch
Developer

Emma Bateman
Developer

Bruce Bolden
Inspirational Teacher

April 9, 2017

Contents

1 About this Project

This project is the best project, a simply fantastic project. I know because I've worked with some of the best projects in the world. It's gonna be great. We're gonna build this project, and Mexico is gonna pay for it.

"I know words. I have the best words." -Donald

2 Design process

stuff
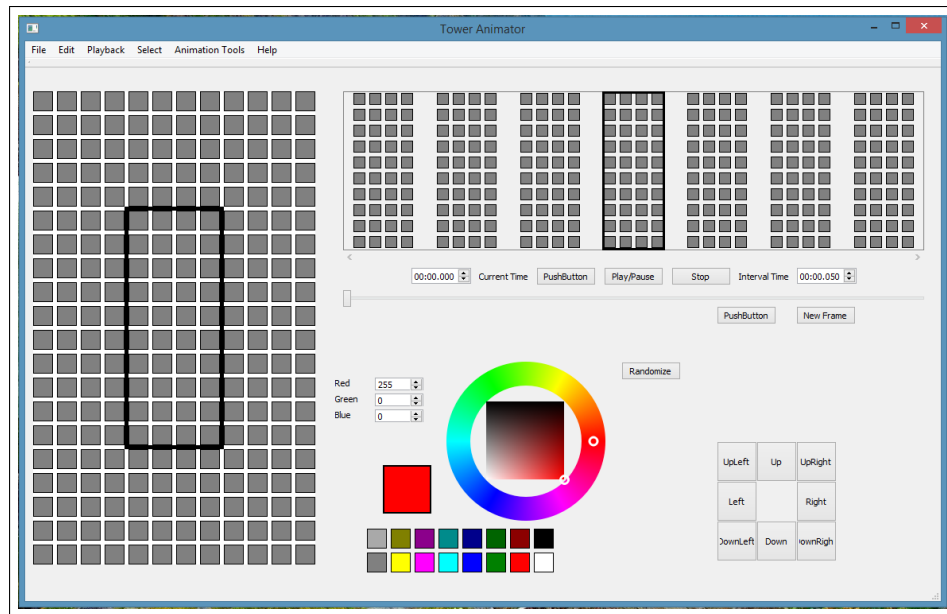
2.1 Problems encountered

stuff

3 Figures

Figure 1: Original GUI design from previous tower lights project
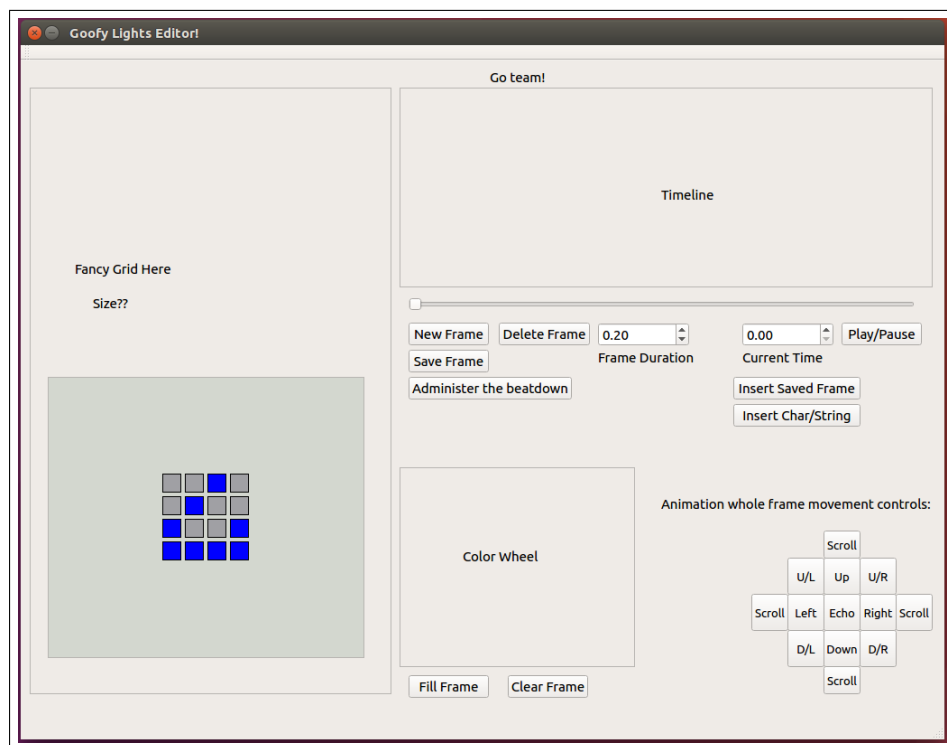

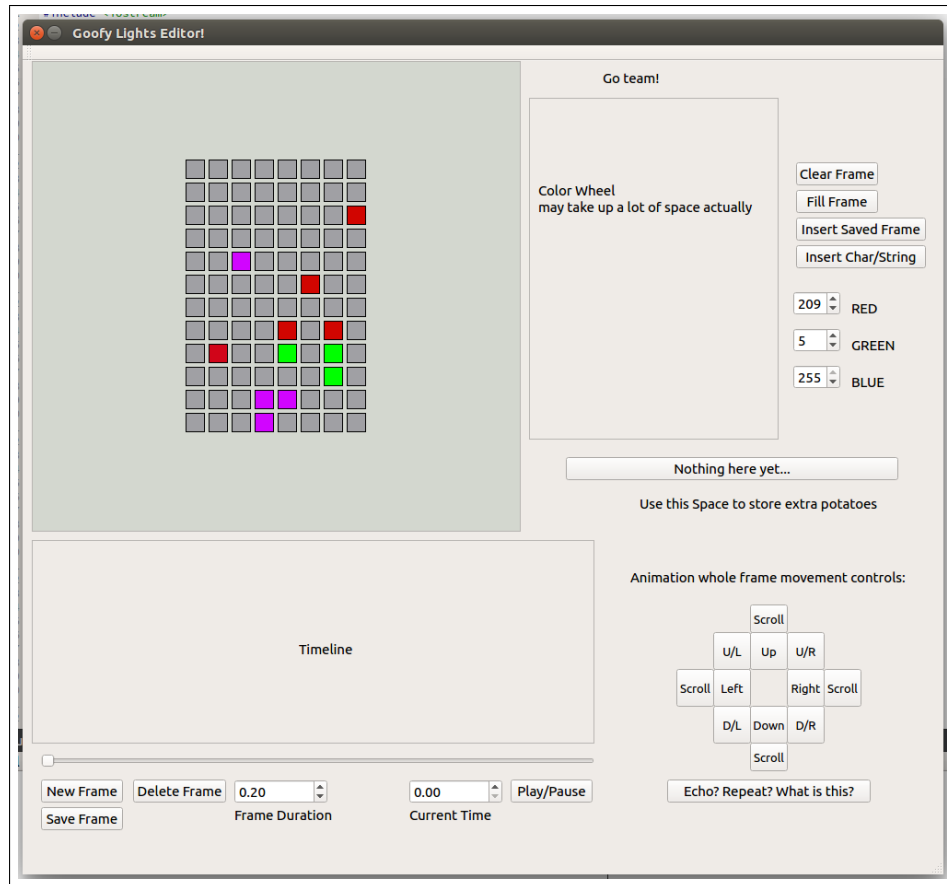Figure 2: Our initial GUI after first sprint

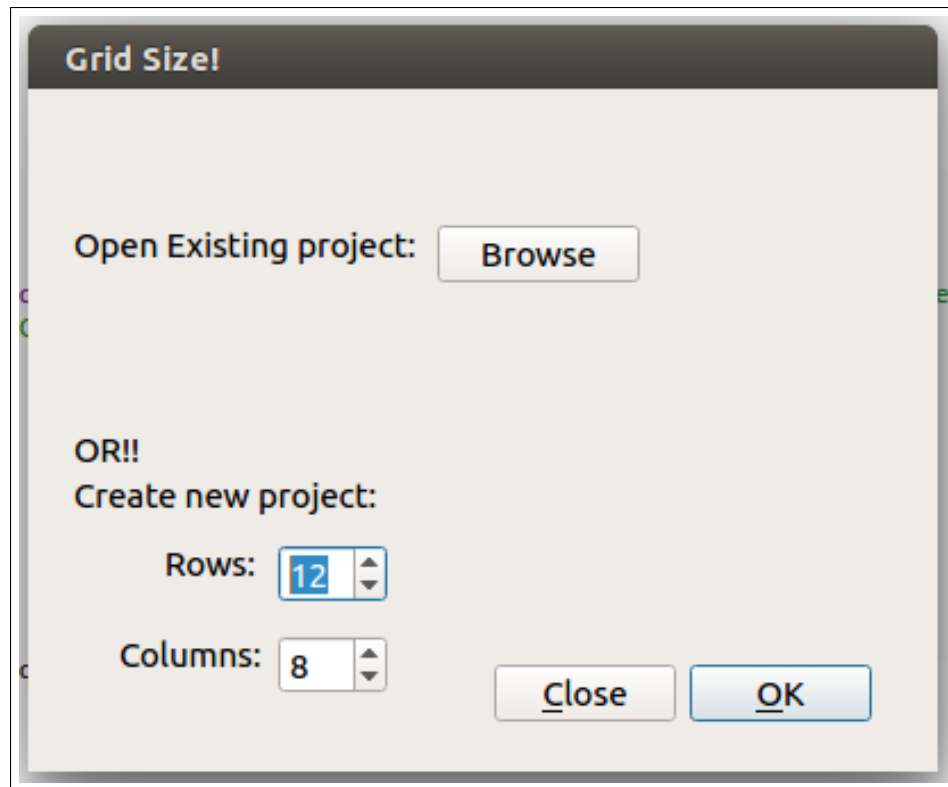Figure 3: GUI after second sprint, RGB functionality added
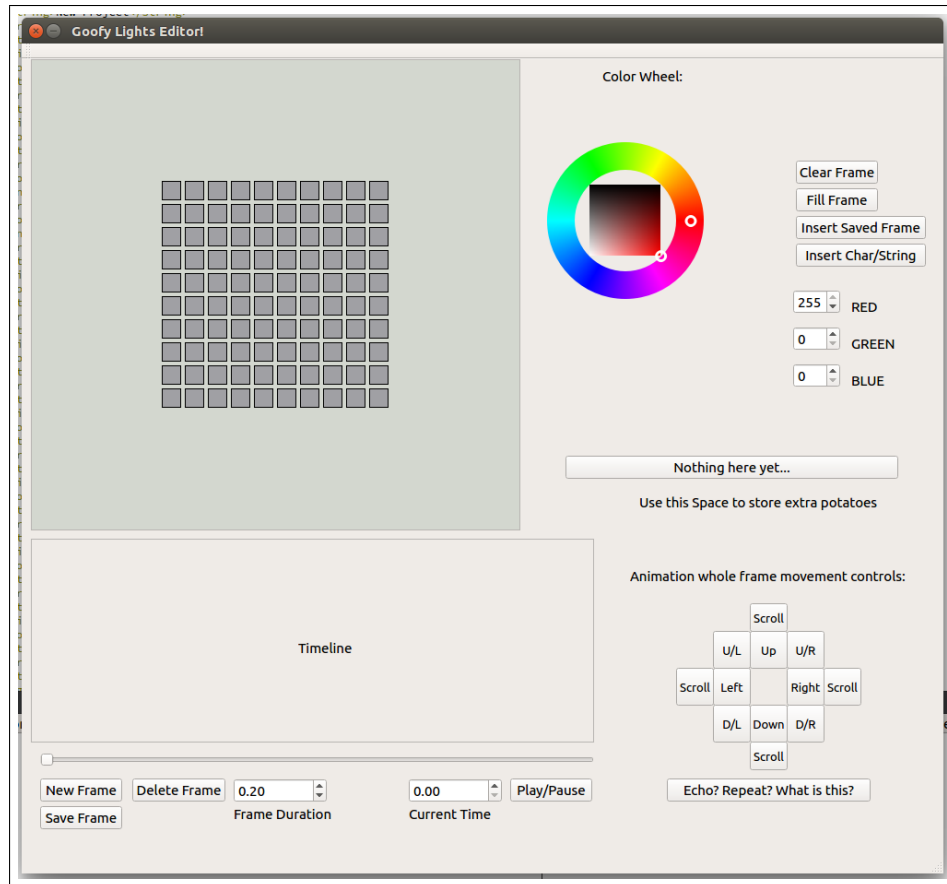
Figure 4: Grid size selection dialog added

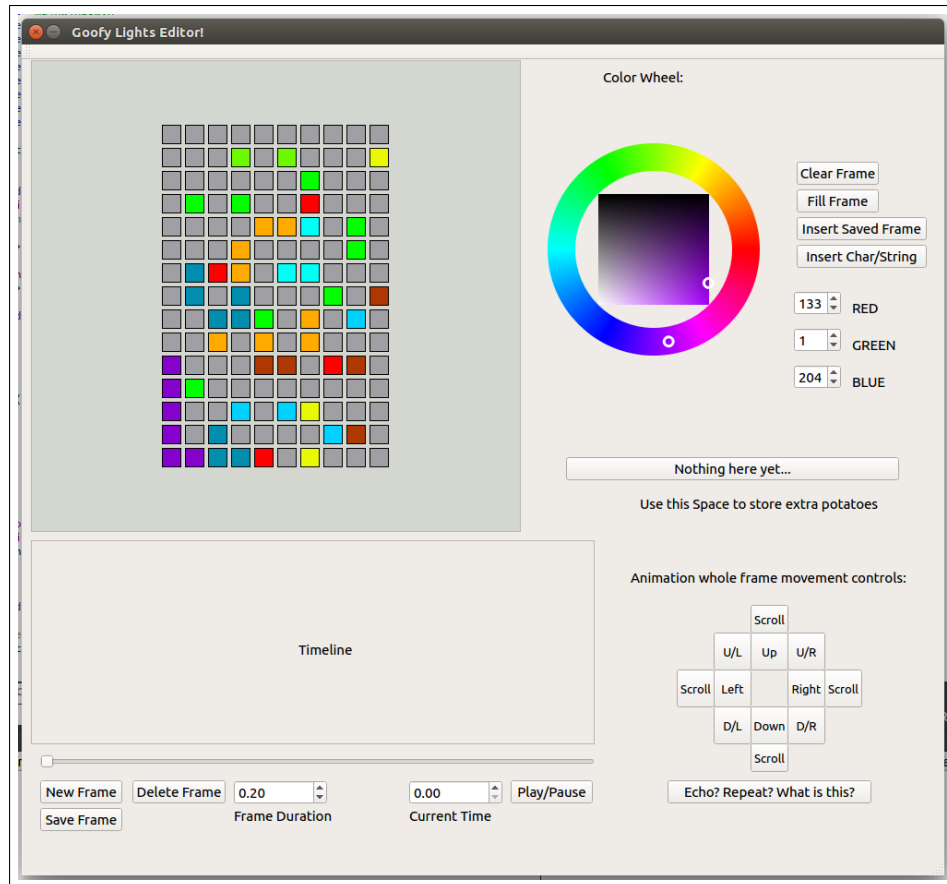Figure 5: Color wheel added in sprint 3

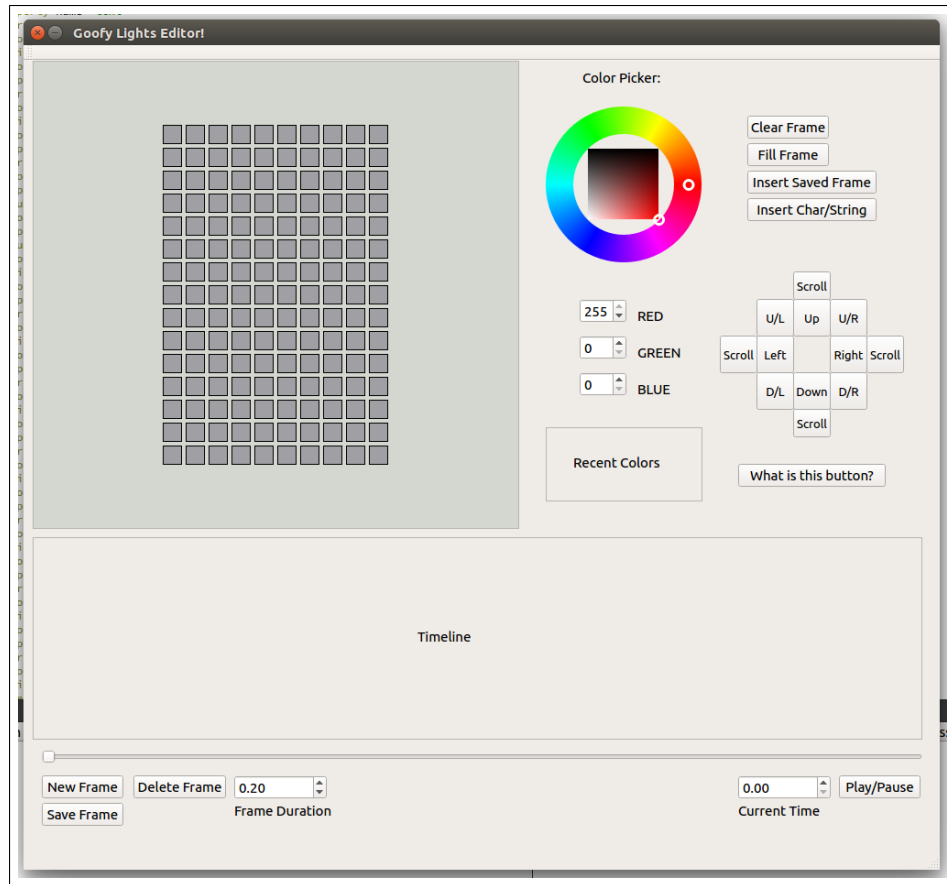Figure 6: Color wheel demo

Figure 7: Control buttons moved in sprint 3
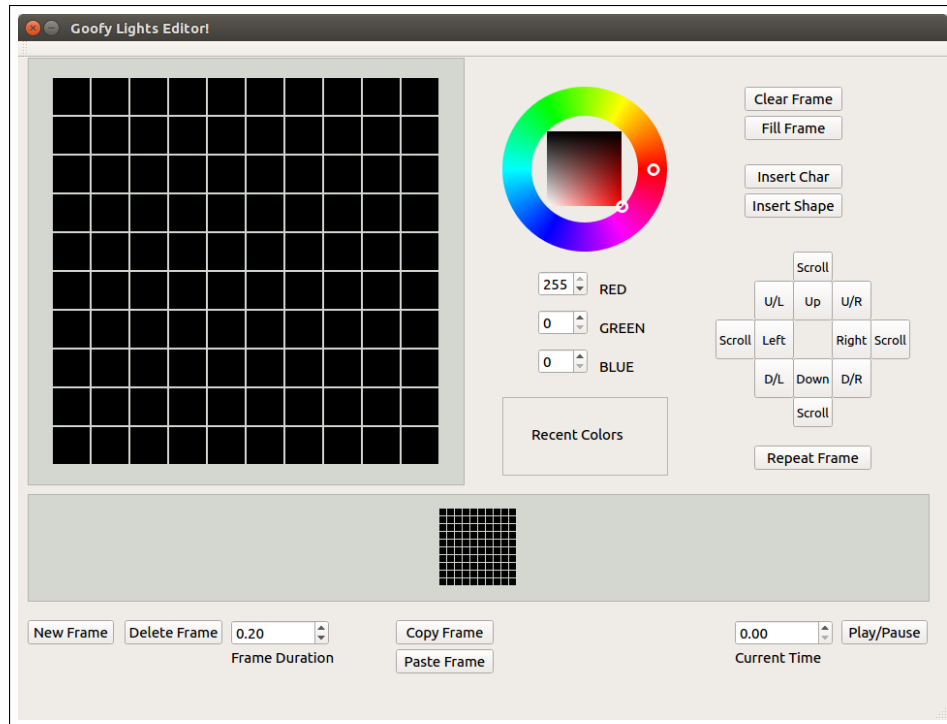
Figure 8: Timeline implemented. Default squares now black
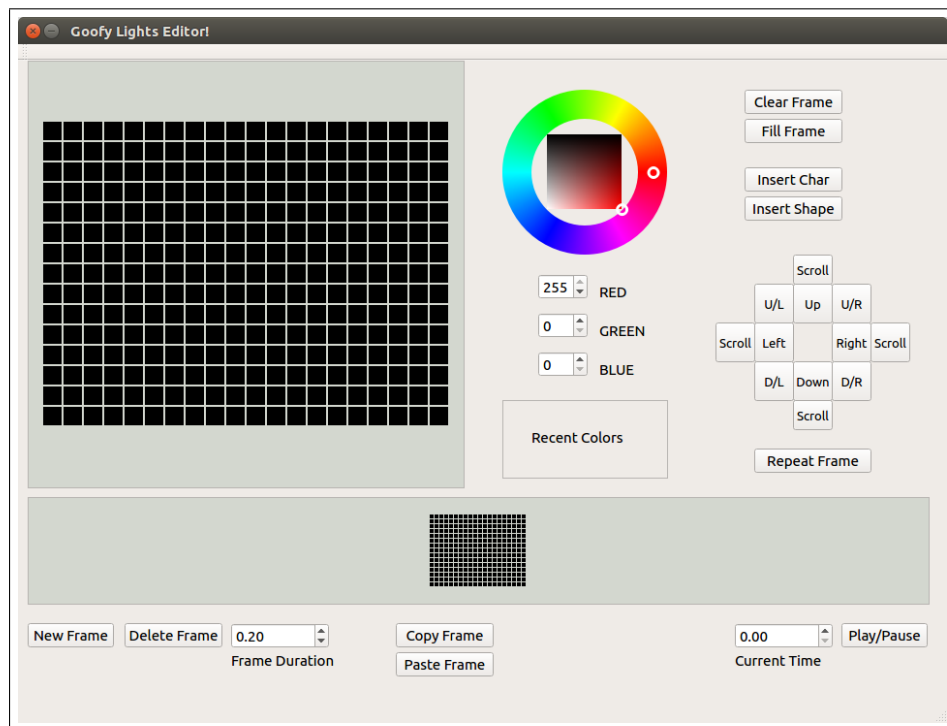
Figure 9: Grid size demo 15x20
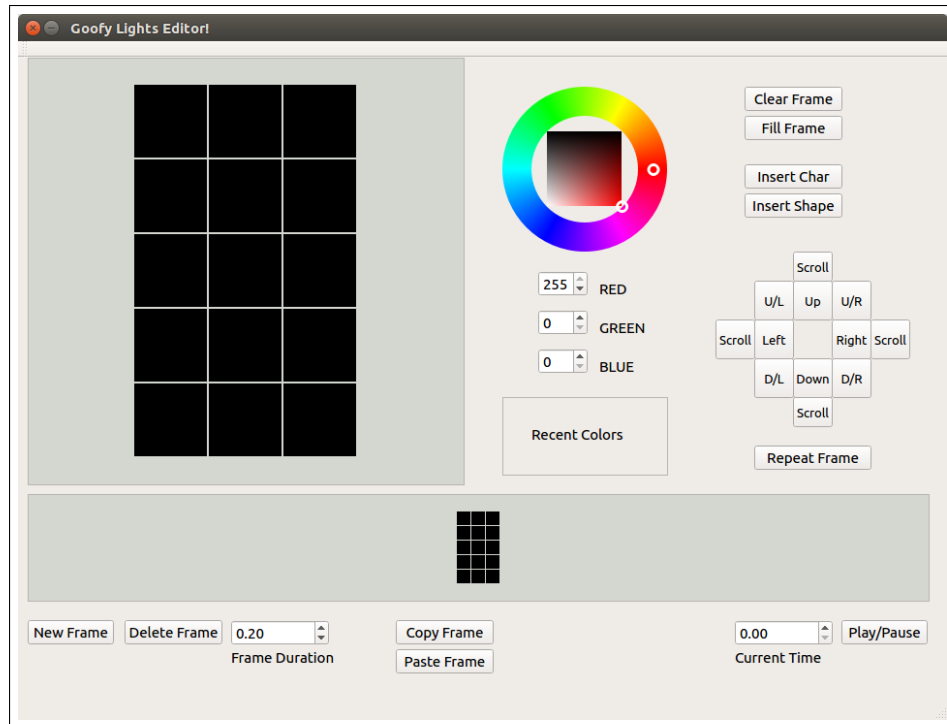
Figure 10: Grid size demo 5x3
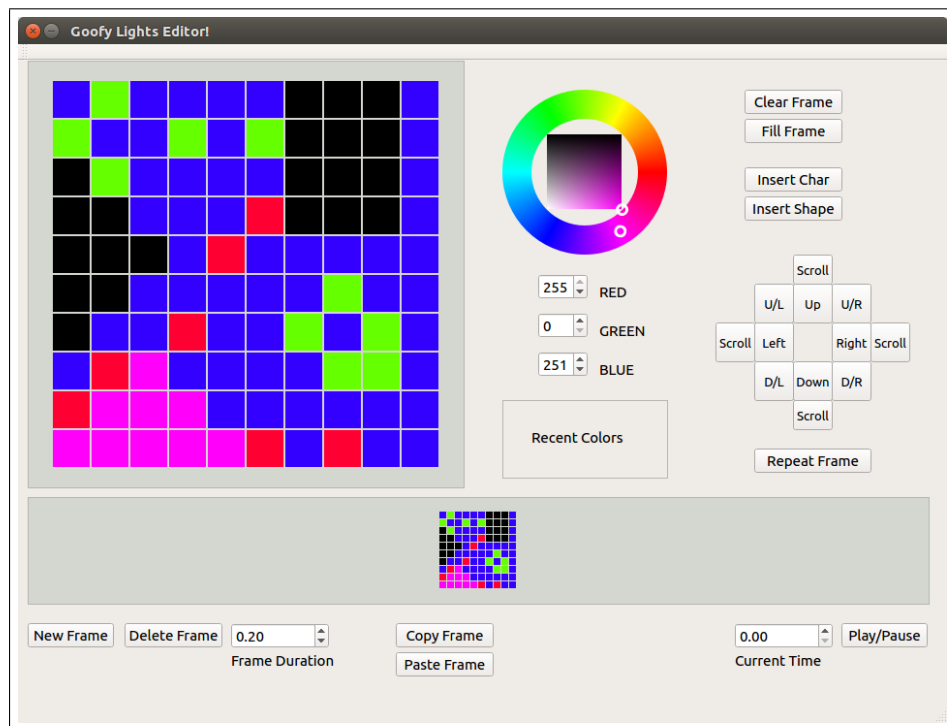
Figure 11: GUI usage demo at the end of sprint 3

3.1 Timeline

```
Rough timeline, revision #2
Starting March 30th (Thursday)

Week 1: (completed)
1) Rough out GUI
2) Create data structure
3) Start file manipulation functions
4) Make something to look at (pretty colors)

Week 2:
Sprint 2:
1) Add Row/Column data to Framedata
2) Delete attached RGB structures
3) Copy Framedata
4) Retrieve frame x
5) Fix RGB bug
6) Update frame x
//finish by TUESDAY???

Week 3:
Sprint 3:
SOLIDIFY GUI LAYOUT (buttons, sliders, sizes)
Start filling out the functionality of buttons

Week 4:
Sprint 4:
Finish all functionality
Add extra features
Start final documentation

Week 5:
Final sprint??
System testing, find bugs, hopefully something works mostly
Update UML diagrams
Polish final documentation
```

---

4 UML Diagrams
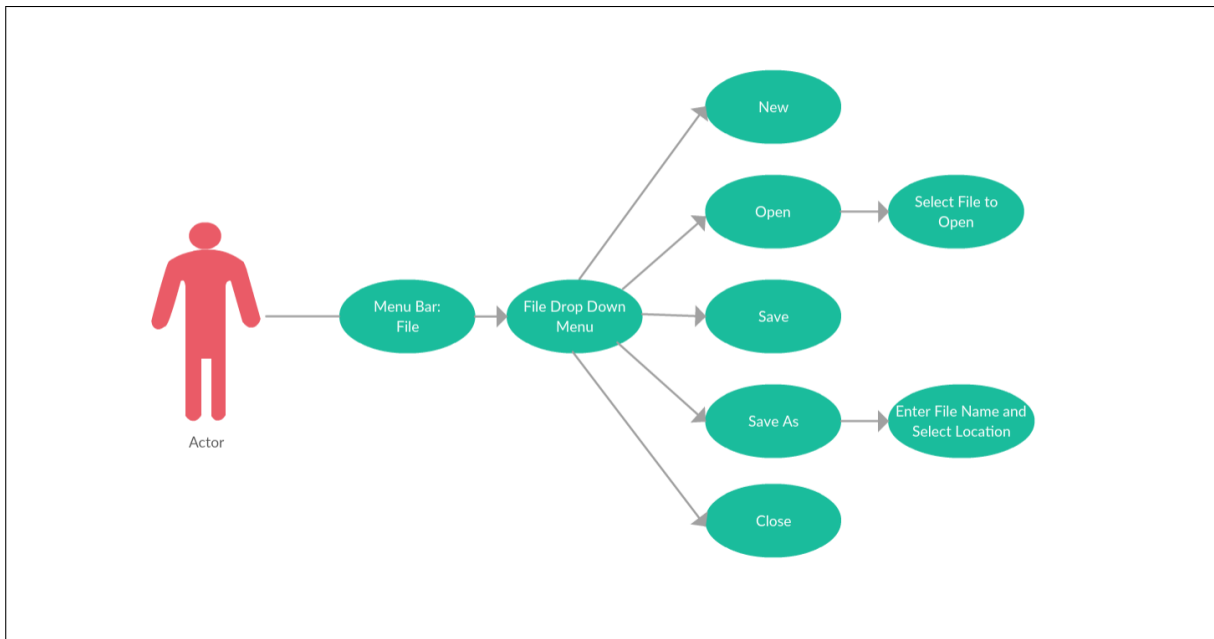
Figure 12: File manipulation
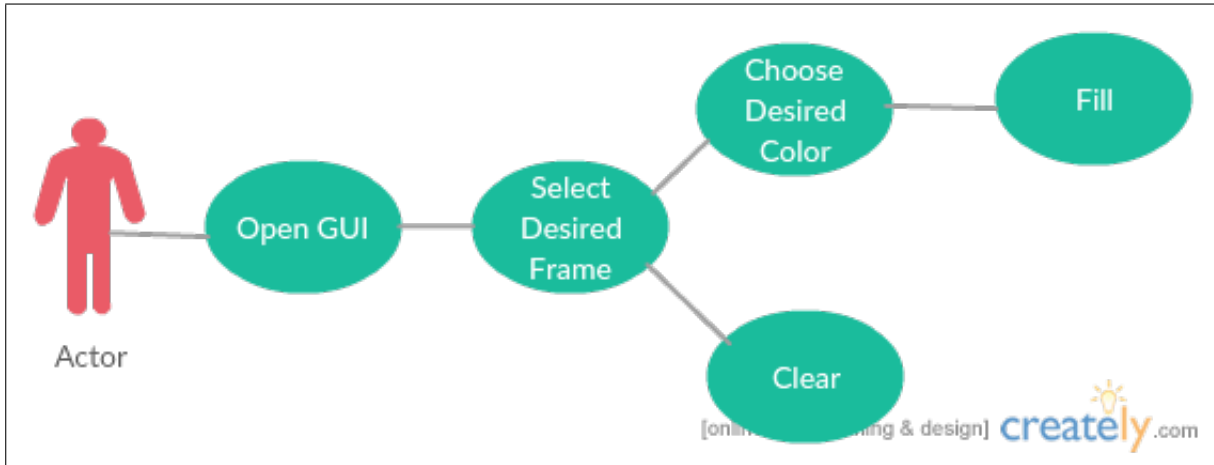
Figure 13: Fill or clear frame with current color
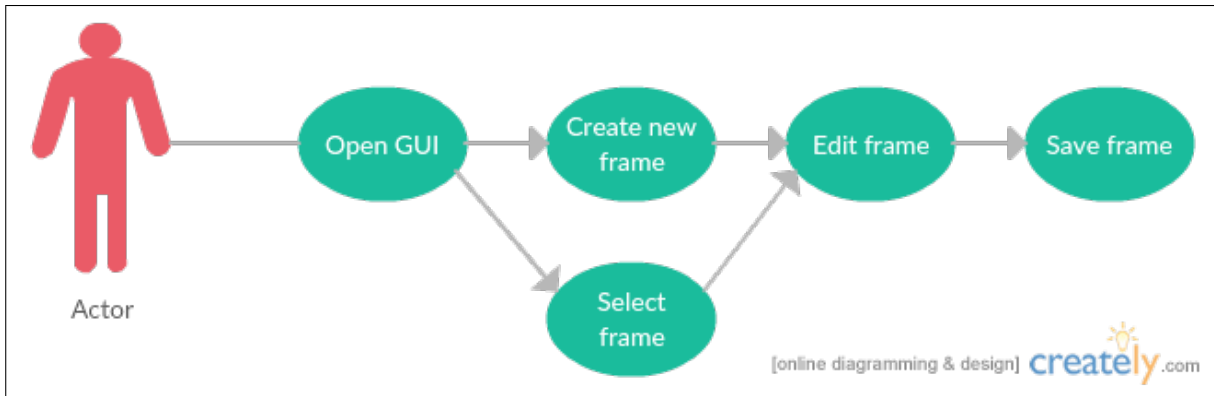
Figure 14: Save/copy current frame for re-use in animation
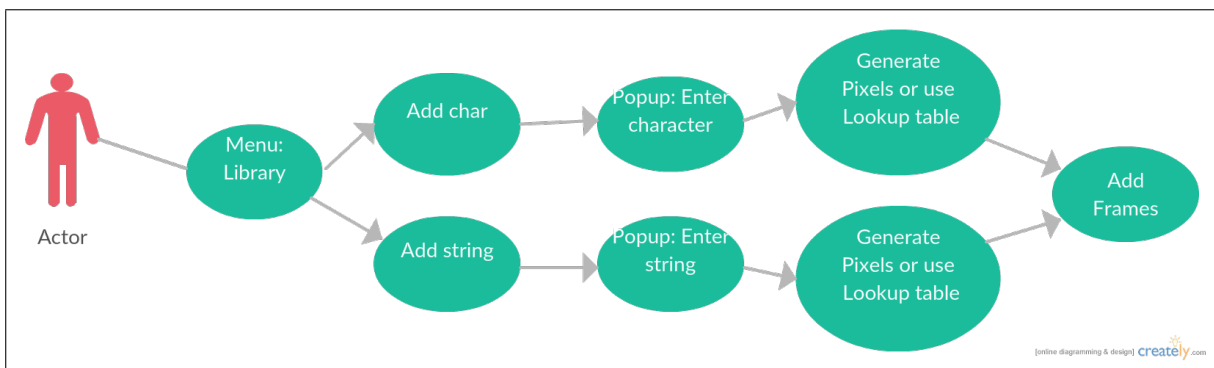
Figure 15: Add char/string from predefined set
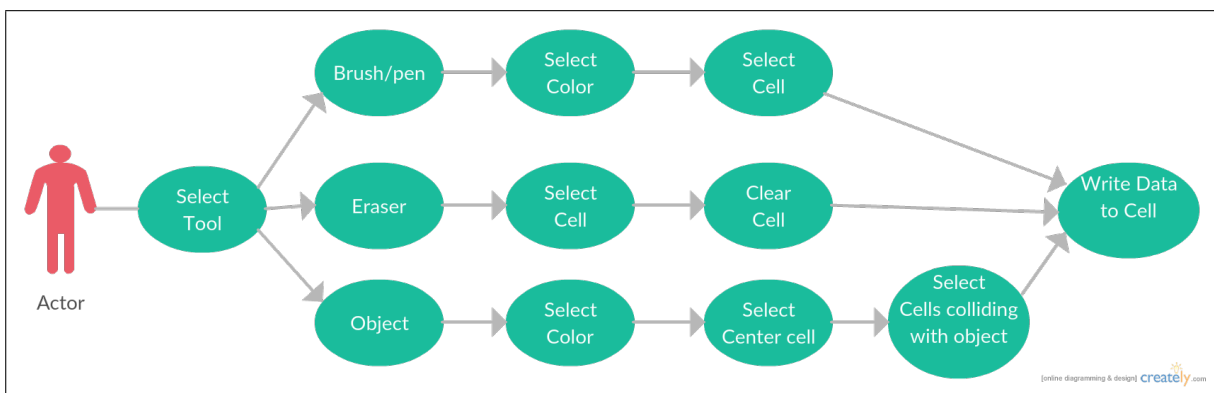
Figure 16: Add a pixel in any position in any color
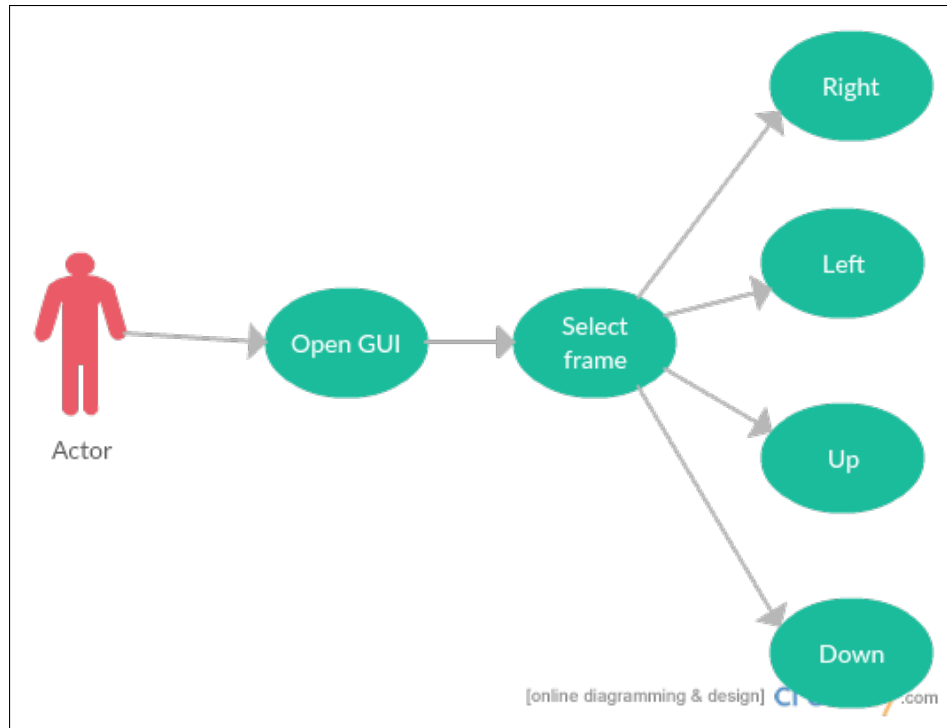
Figure 17: Move everything in frame 1 pixel in a direction
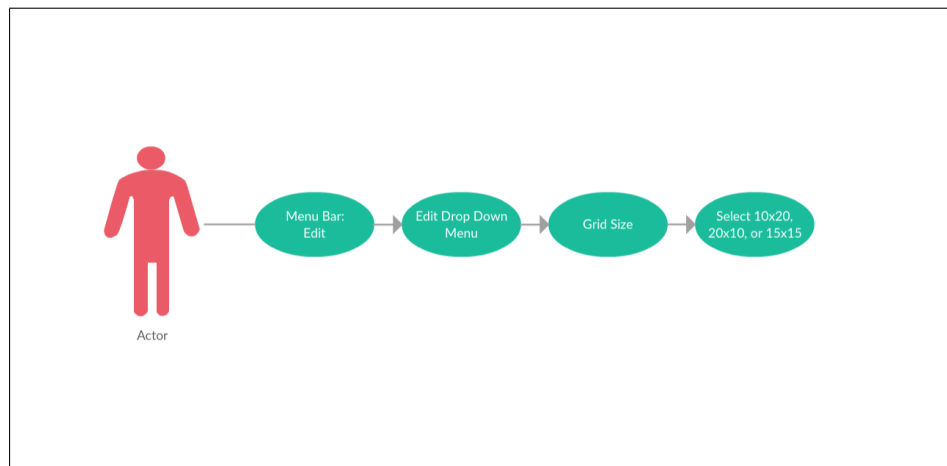
Figure 18: Choose grid size for the file
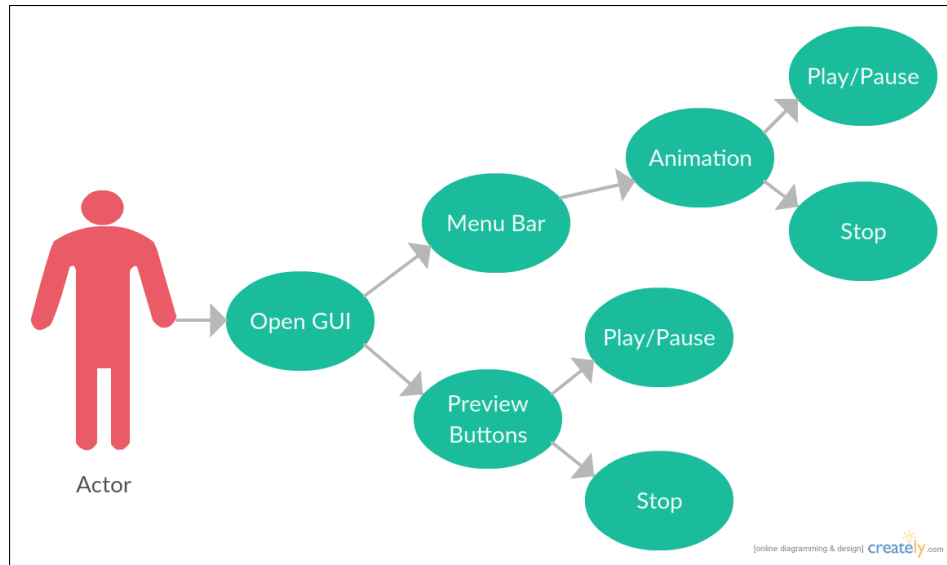
Figure 19: Preview play/pause/stop animation
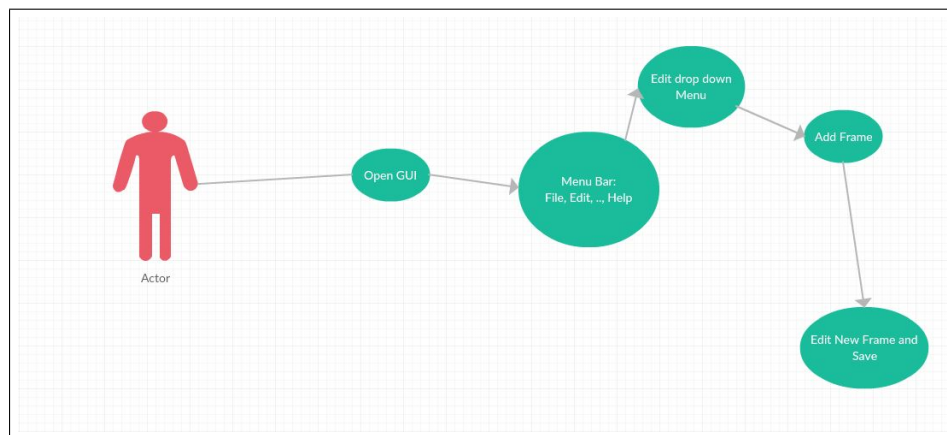
Figure 20: Create new frame
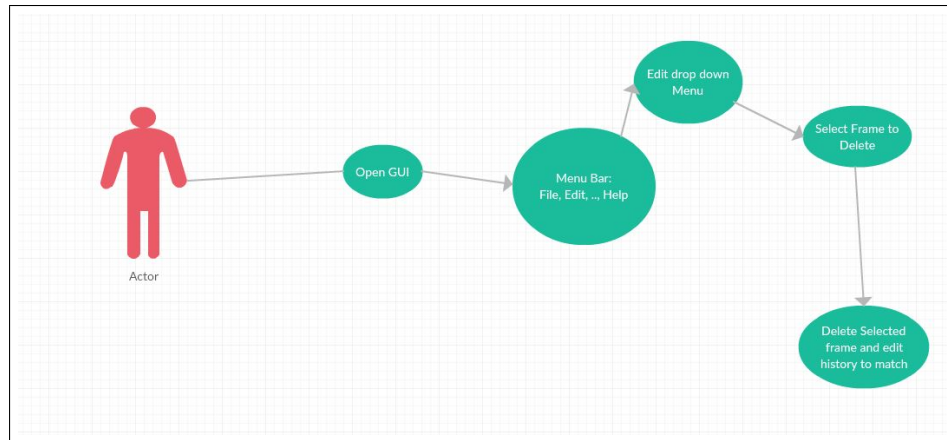
Figure 21: Delete a frame
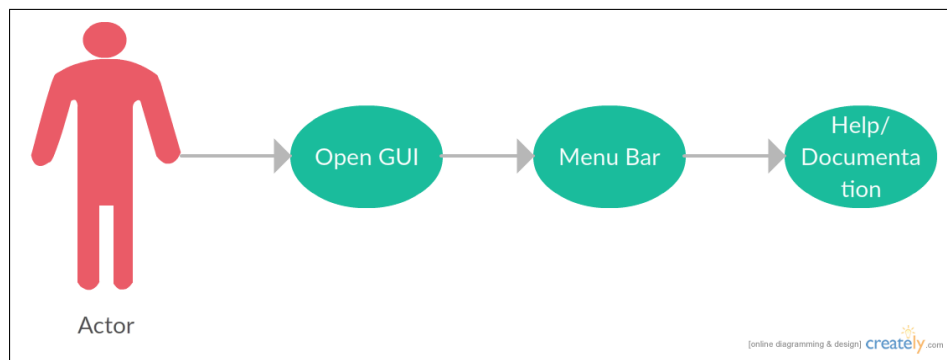
Figure 22: Open help/documentation text
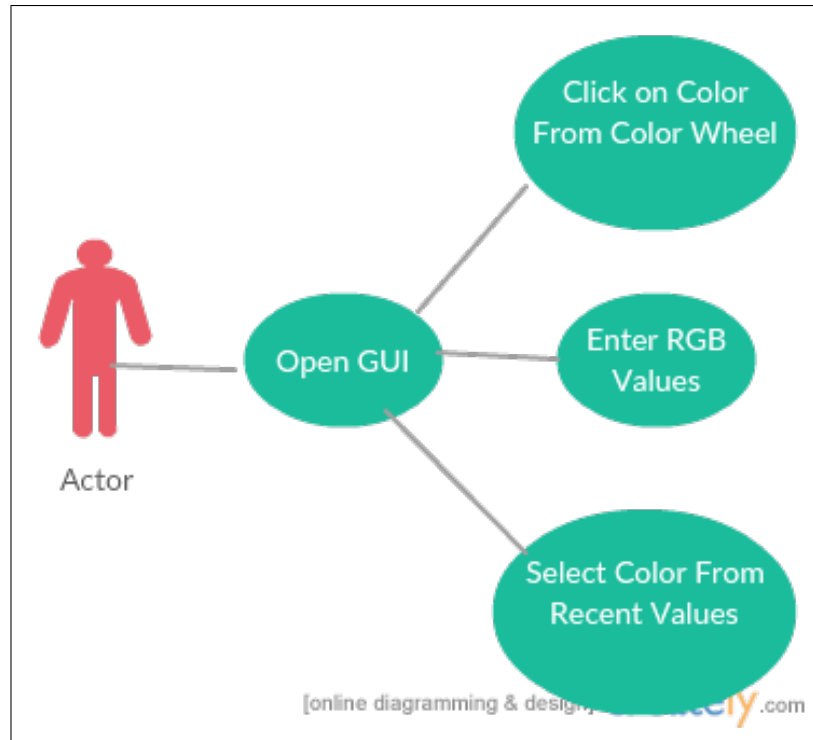
Figure 23: Chose a color on the colorwheel
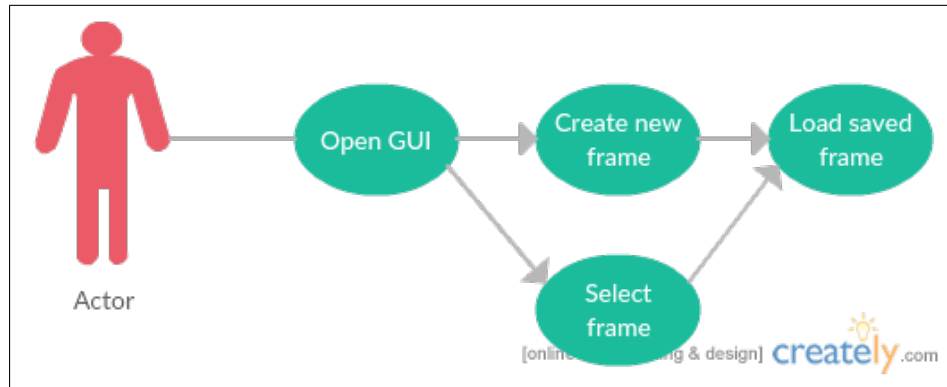
Figure 24: Insert a saved frame
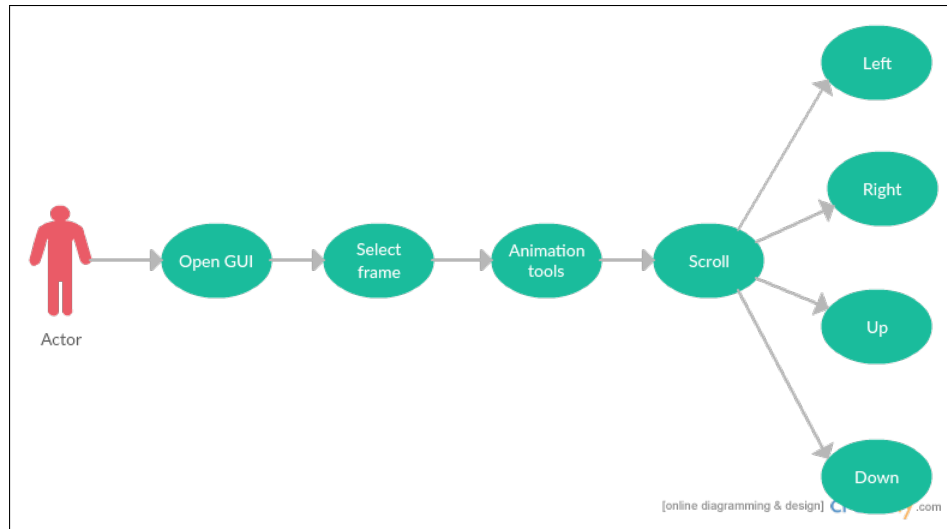
Figure 25: Scroll everything across entire frame

5 Files

5.1 main.cpp file

```cpp
#include <iostream>
#include "mainwindow.h"
#include <QApplication>

#include "framestructure.h"
#include "FrameList.h"
#include "FrameManipulation.h"
#include "FileOperations.h"
#include <globals.h>
#include <sizedialog.h>

int G_COL = 0; //initialize globals -P
int G_ROW = 0;
double G_SCALE = 0;

int G_RED = 255;
int G_GREEN = 0; //fixed -P
int G_BLUE = 0;
int G_RED_RIGHT = 0;
int G_GREEN_RIGHT = 0;
int G_BLUE_RIGHT = 0;

int G_FRAMECOUNT = 0; //hah -P


long FrameIDCount = 0;

// creates the dynamic RGB array
t_RGB** create_RGB(int r, int c);

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    //LET'S GET IT STARTED -P
    SizeDialog dialog1;
    dialog1.setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);
        //You will NOT exit this window your way -P
    dialog1.setModal(true); //YOU SHALL NOT CLICK OUTSIDE OF THIS WINDOW -P
    dialog1.exec(); //execute pls -P

    //read in a file here probably -P

    MainWindow w;
    w.show();

    // linked list test
    std::cout << "FrameList testing" << std::endl;
```

```cpp
t_RGB ** rgb_data;

t_FrameData FrameData; // Local frame data
FrameList frames(G_ROW, G_COL); // linked list for frame data. r c for print function

//Frame 0
// Generate new rgb_data arrary
rgb_data = create_RGB(G_ROW, G_COL);

// fill rgb_data for Frame 0
unsigned short color = 0; // arbritrary data
for (int i = 0; i < G_ROW; i++){
    for (int j = 0; j < G_COL; j++){
        rgb_data[i][j].R = color++;
        rgb_data[i][j].G = color++;
        rgb_data[i][j].B = color++;
    }
}
FrameData.ID = FrameIDCount++;
FrameData.durration = 123;
FrameData.data = rgb_data;
frames.AddTail(FrameData); // add this frameData to linked list
rgb_data = NULL; // disconnect this pointer from rgb_data


// next frame Frame 1
// Generate new rgb_data arrary
rgb_data = create_RGB(G_ROW, G_COL);

 // fill rgb for Frame 1
color = 100; // arbritrary increment to make data different
for (int i = 0; i < G_ROW; i++){
    for (int j = 0; j < G_COL; j++){
        rgb_data[i][j].R = color++;
        rgb_data[i][j].G = color++;
        rgb_data[i][j].B = color++;
    }
}
FrameData.ID = FrameIDCount++;
FrameData.durration = 212;
FrameData.data = rgb_data;
frames.AddTail(FrameData);
rgb_data = NULL; // disconnect this pointer from rgb_data


// next frame Frame 2
rgb_data = create_RGB(G_ROW, G_COL);
FrameData.ID = FrameIDCount++;
FrameData.durration = 214;
FrameData.data = rgb_data;
fillFrame2(FrameData, 21, 32, 45);
frames.AddTail(FrameData);

// copyFrame Test Frame 3
```

```cpp
    t_FrameData newFrame;
    rgb_data = create_RGB(G_ROW, G_COL);
    newFrame.ID = FrameIDCount++;
    newFrame.durration = 217;
    newFrame.data = rgb_data;
    copyFrame(newFrame, FrameData);
    frames.AddTail(newFrame);

    // std::cout << "Now printing frames" << std::endl;
    // frames.PrintNode();
    FrameList frameList;
    FileOperations::LoadFromFile("autofill.proj", &frameList);
    frameList.PrintNode();
    FileOperations::SaveToFile(frames, "autofill.proj");

    frames.DeleteList();
    std::cout << "Program end" << std::endl;
    return a.exec();
}
```

5.2 FrameList.h file

```cpp
/* Tim Sonnen      Lab #5
 * 9/24/2015
 *
 * FrameList.h
 * Class that holds a linked list
 */

#ifndef LINK_H
#define LINK_H

#include <iostream>
#include "framestructure.h"
class FrameList{
private:
    struct Node{
        t_FrameData FrameData;
        struct Node *next;
        struct Node *prev;
    };
    typedef struct Node* NodePtr;

    NodePtr head;
    NodePtr tail;
    int row, col;
    int count;

public:
    // Constructor
    FrameList(int r, int c){
        head = NULL;
        row = r;
        col = c;
        count = 0;
    }

    // Constructor for empty FrameList
    FrameList(){
        head = NULL;
        row = 0;
        col = 0;
        count = 0;
    }

    // Destructor
    ~FrameList(){
      FrameList::DeleteList();
    }
    // Add a node onto the end of the linked list.
    void AddTail(t_FrameData x);

    // Function will call DeleteNode for every item in the Linked List, and delete
```

```cpp
    // the head node until all items in the linked list have been deleted.
    void DeleteList();

    // Delete the first node in the list.
    void DeleteNode();

    // Add node at position x
    // 0 will be the first node. 1 would be the 2nd node in the list.
    void AddNode_Middle(t_FrameData x, int pos);

    // Delete node at position x in the list
    // If pos == 0 this refers to the head node, and
    // If pos == count - 1 this refers to the tail node.
    void DeleteNode_Middle(int pos);

    // retrieve node at given position x.
    // If pos == 0 this refers to the head node and,
    // If pos == count - 1 this refers to the tail node.
    t_FrameData * RetrieveNode_Middle(int pos);

    // Return the first node found in the list
    t_FrameData FirstNode();

    // Advance one node in the list
    int AdvanceList();

    // Get the row count
    int GetRowCount();

    // Get the column count
    int GetColCount();

    // Output the values in the nodes, one integer per line.
    void PrintNode();

    //Returns a value if the list is empty
    int IsEmpty();

    // Return a count of the number of nodes in the list.
  int Size();

    // Updates frame data
    void UpdateNode(t_FrameData d, int position);
};
#endif
```

5.3   FrameList.cpp file

```cpp
/* Tim Sonnen      Lab #4
 * 9/25/2015
 *
 * FrameList.cpp
 *
 */

#include <iostream>

#include "FrameList.h"
#include "framestructure.h"

using namespace std;

/* Add an item to the end of the list*/
void FrameList::AddTail( t_FrameData n ){
    NodePtr p;

    //Allocate the node
    p = new Node;
    p->FrameData = n;
    p->next = NULL;

    //Check if the list is empty
    if(head == NULL ){
        head = p;
        tail = p;
        p->prev = NULL;
    }
    else{
        // next/prev pointers
        tail -> next = p;
        p -> prev = tail;

        // update tail pointer
        tail = p;
    }
    this->count++;
}

/*Function to delete all the entries in the Linked List upon program termination */
void FrameList::DeleteList(){
    while (head != NULL)
    {
        DeleteNode();
    }
}

/*Deletes the first node in the list*/
void FrameList::DeleteNode(){
    NodePtr p = head;
```

```cpp
    if( p == NULL){
        /*Nothing. Error case.*/
        return;
    }
    else{
        head = p->next;
        p->next = NULL;
        if (head != NULL)
            head->prev = NULL; //this is causing a seg fault when I close -P
        // Delete Attached RGB structure here
        delete p;
    }
    this->count--;
}

void FrameList::AddNode_Middle(t_FrameData x, int pos){
    // indexing scheme to start at 0 to n where n == items in linked list
    // Node 0 is the head of the list while node (count - 1) is the tail
    int tempCount = 0;

    NodePtr current = head;
    NodePtr insert = new Node;
    insert -> FrameData = x;
    insert -> next = NULL;
    insert -> prev = NULL;

    if (head == NULL)
    {
        head = insert;
    }

    else if (pos == 0)
    {
        insert -> next = head;
        insert -> prev = NULL;
        head -> prev = insert;
        head = insert;
        this->count++;
        return;
    }
    else
    {
        while (tempCount != pos-1 && current != NULL)
        {
            current = current -> next;
            tempCount++;
        }
        insert -> next = current -> next;
        // Adjustment of previous pointer for addition of node x
        if (current -> next != NULL)
        {
            NodePtr p = current -> next;
            p -> prev = insert;
```

```cpp
        }
        // else if current -> next == null then insert == null above

        // Adjustment of prev pointer for node added at position x
        current -> next = insert;
        insert -> prev = current;

        this->count++;
        return;
    }
}

void FrameList::DeleteNode_Middle(int pos){

    if (head == NULL){
        // Error list is empty, do nothing and return.
        return;
    }

    NodePtr current = head;

    if (pos == 0){
        // Delete the head node.
        head = current -> next;
        if (head != NULL)
            head -> prev = NULL;
        // Delete Attached RGB structure here
        delete(current);
        this->count--;
        return;
    }

    for (int i = 0; current != NULL && i < pos - 1; i++){
        current = current -> next;
    }

    if (current == NULL || current -> next == NULL){
        // the position given is greater than total number of nodes in the list.
        return;
    }

    // if this point has been reached and the function has not returned, current -> next
    //    holds
    // the node to be deleted from the list.
    NodePtr p = current -> next -> next;
    // Adjustment of previous pointers.
    p -> prev = current;
    // Delete Attached RGB Structure here
    delete (current -> next);
    current -> next = p;
    this->count--;
    return;
}
```

```cpp
// Added function to search the lined list for node at position x
// same indexing scheme as before, passing 0 to this function refers to the head
// passing count - 1 to this function refers to the tail.
t_FrameData * FrameList::RetrieveNode_Middle(int pos){
   int tempCount = 0;
    t_FrameData *rtnVal = NULL;
   NodePtr temp = head;

   if (pos == 0)
   {
      // return pointer to the head node's FrameData.
      if (head == NULL)
      {
         // no pointer to return Error.
                return NULL;
      }
      else
      {
            *rtnVal = head -> FrameData;
            return rtnVal;
      }
   }
   else
    {
         while (tempCount != pos && temp != NULL)
      {
         tempCount++;
         temp = temp -> next;
      }

      if (temp != NULL)
      {
         // Just checking to make sure no bounds have been crossed.
            *rtnVal = temp -> FrameData;
            return rtnVal;
      }
      else
      {
            return NULL;
      }
   }
}

/*Returns the first node in the list */
t_FrameData FrameList::FirstNode(){
    return head->FrameData;
}

/* Advance one node through the list */
int FrameList::AdvanceList(){
   this->head = this->head->next;

   /* If we are out of the list return 0, else return 1 */
   if (this->head == NULL)
```

```cpp
            return 0;
        else
            return 1;
}


/* Get the row count */
int FrameList::GetRowCount(){
    return this->row;
}


/* Get the column count */
int FrameList::GetColCount(){
    return this->col;
}


/*Returns if the list is empty or not*/
int FrameList::IsEmpty(){
    if(head == NULL){
        return 1;
    }
    else{
        return 0;
    }
}


void FrameList::PrintNode(){
    NodePtr p = head;
    /* sample output
        ID: 0 Dur: 123
        0,1,2 : 3,4,5 : 6,7,8 :
        9,10,11 : 12,13,14 : 15,16,17 :
        18,19,20 : 21,22,23 : 24,25,26 :
        27,28,29 : 30,31,32 : 33,34,35 :
        36,37,38 : 39,40,41 : 42,43,44 :
    */

    while(p != NULL){
        cout << "ID: " << p->FrameData.ID << "\tDur: " << p->FrameData.durration << endl;
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                cout << p->FrameData.data[i][j].R << "," << p->FrameData.data[i][j].G
                    << "," << p->FrameData.data[i][j].B << " : ";
            }
            cout << endl;
        }
        cout << endl;
        p = p->next;
    }
}


int FrameList::Size(){
    return this->count;
}
```

```cpp
void FrameList::UpdateNode(t_FrameData d, int position)
{
    int pcount = 0;
    NodePtr current = head;
    NodePtr temp = new Node;
    NodePtr old;
    temp -> FrameData = d;
    temp -> next = NULL;
    if (position == 0)
        {
        if (head == NULL)
            head = temp;
        else if (head -> next == NULL)
            {
            head = temp;
            delete current;
            }
         else
        {
            old = current;
            current = current -> next;
            temp -> next = current;
        delete old;
            }
        }
    else
        {
        current = current -> next;
        pcount++;
        while (pcount != position-1)
            {
            current = current -> next;
            pcount++;
            }
        temp -> next = current -> next -> next;
        old = current -> next;
        current -> next = temp;
        delete old;
        }
    return;
}
```

5.4 FrameManipulation.h file

```c
#ifndef FRAMEMANIPULATION_H
#define FRAMEMANIPULATION_H

#include "framestructure.h"

// Directions
#define D_UP      1
#define D_DWN     2
#define D_LEFT    3
#define D_RIGHT   4
#define D_UP_L    5
#define D_UP_R    6
#define D_DWN_L   7
#define D_DWN_R   8

// Return codes
#define SUCSSESFUL 0
#define ERROR     1


// Prototypes

// creates a RGB Array and returns a pointer to it.
t_RGB** create_RGB(int r, int c);

// Takes origional frame and returns a new copy of it
int copyFrame(t_FrameData &copyFrame, t_FrameData origFrame);

// translates from by a given direction
int translateFrame(t_FrameData d, int direction);

// Fills given frame with color
int fillFrame(t_FrameData &d, t_RGB rgb_fill);
int fillFrame2(t_FrameData &d, unsigned short r, unsigned short g, unsigned short b);

#endif // FRAMEMANIPULATION_H
```

5.5 FrameManipulation.cpp file

```cpp
#include "FrameManipulation.h"
#include <iostream>
#include "globals.h"

// Function creates a 2d memory element of the RGB struct then passes back the pointer
    to it.
t_RGB** create_RGB(int r, int c)
{
    t_RGB** arr = new t_RGB*[r];
    for(int i = 0; i < r; ++i)
        arr[i] = new t_RGB[c];
    return arr;
}



// creates a RGB Array and returns a pointer to it.
t_RGB** create_RGB(int r, int c);



// Takes origional frame and returnes a new copy of it
// Frame must be declared and rgb_data array must already be allocated
int copyFrame(t_FrameData &copyFrame, t_FrameData origFrame)
{
    // Error checking
    if (origFrame.data == NULL)
        return ERROR;

    int i, j; // loop control

    //Fill rgb_data with data from FrameData
    for (i = 0; i < G_ROW; i++) {
        for (j = 0; j < G_COL; j++) {
            copyFrame.data[i][j] = origFrame.data[i][j];
        }
    }

    return SUCSSESFUL;
}

// translates from by a given direction
int translateFrame(t_FrameData d, int direction)
{
    int i = 0; //counters
    int j = 0;
    t_RGB *temp; //Temp variable for row that is pushed out of the frame
    t_RGB emptyRGB;
    emptyRGB.B = 0;
    emptyRGB.G = 0;
    emptyRGB.R = 0;
```

```cpp
//Condition for up, up left, and up right. Uses recursion for left and right
    translation
if(direction == D_UP || direction == D_UP_L || direction == D_UP_R){
    temp = d.data[0];
    for(i = 0; i < G_ROW-1; i++){
        d.data[i] = d.data[i+1];
    }
    d.data[i] = temp;
    for(int j = 0; j < G_COL; j++){
        d.data[i][j] = emptyRGB;

    }

    if(direction == D_UP_L)
        translateFrame(d, D_LEFT);
    if(direction == D_UP_R)
        translateFrame(d, D_RIGHT);

}
//Condition for down, down left, and down right. Uses recursion for left and right
    translation
if(direction == D_DWN || direction == D_DWN_L || direction == D_DWN_R){
    temp = d.data[G_ROW-1];
    for(i = G_ROW-1; i > 0; i--){
        d.data[i] = d.data[i-1];
    }
    d.data[i] = temp;
    for(j = 0; j < G_COL; j++){
        d.data[i][j] = emptyRGB;
    }
    if(direction == D_DWN_L)
        translateFrame(d, D_LEFT);
    if(direction == D_DWN_R)
        translateFrame(d, D_RIGHT);
}
//Condition for left transition
if(direction == D_LEFT){
    for(i = 0; i < G_ROW; i++){
        for (j = 0; j < G_COL-1; j++){
            d.data[i][j] = d.data[i][j+1];
        }
    }
    for(i = 0; i < G_ROW; i++){
        d.data[i][j] = emptyRGB;
    }
 }
//Condition for right transition
if(direction == D_RIGHT){
    for(i = 0; i < G_ROW; i++){
        for (j = G_COL-1; j > 0; j--){
            d.data[i][j] = d.data[i][j-1];
        }
    }
    for(i = 0; i < G_ROW; i++){
        d.data[i][j] = emptyRGB;
```

```
        }
    }

    return SUCSSESFUL;
}

// Fills given frame with color
int fillFrame(t_FrameData &d, t_RGB rgb_fill)
{
    int i, j; // loop control

    // Error checking
    if (d.data == NULL)
        return ERROR;

    //Fill data
    for (i = 0; i < G_ROW; i++) {
        for (j = 0; j < G_COL; j++) {
            d.data[i][j] = rgb_fill;
        }
    }

    return SUCSSESFUL;
}

int fillFrame2(t_FrameData &d, unsigned short r, unsigned short g, unsigned short b)
{
    t_RGB rgb;
    rgb.R = r;
    rgb.G = g;
    rgb.B = b;

    // error check if over bounds for type short
    if (r > 255 || g > 255 || b > 255)
        return ERROR;

    return fillFrame(d, rgb);
}
```

Other files here