Blockmation

Samuel Jones – srj12@aber.ac.uk

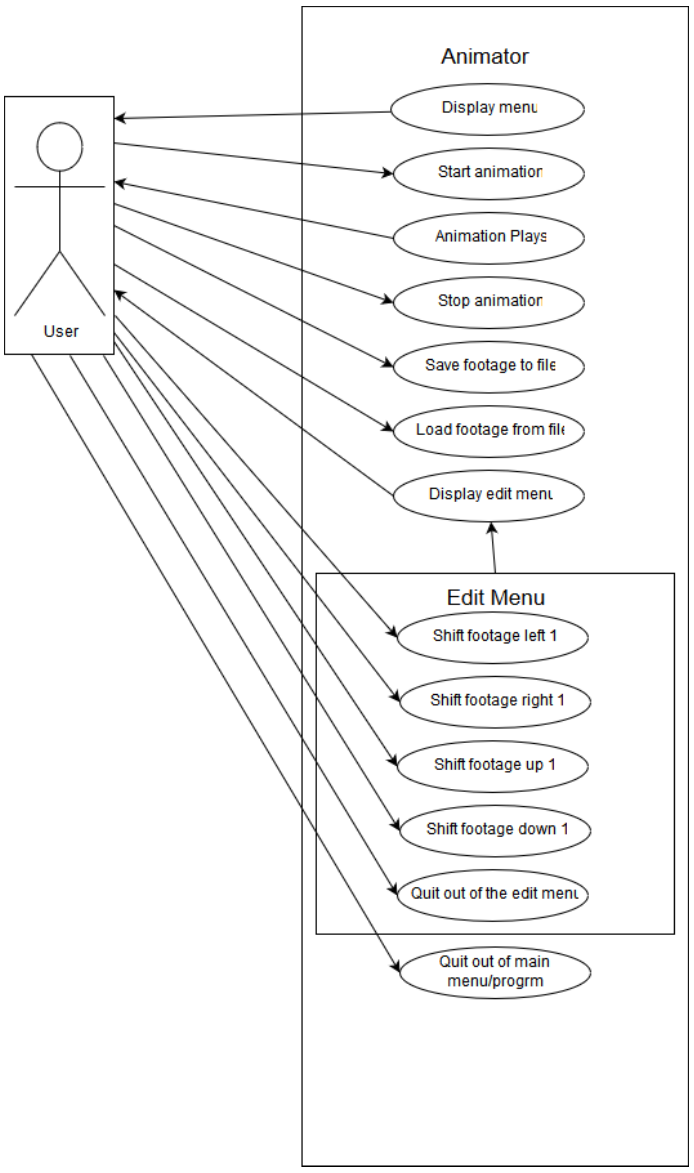
Date – 09th May 2017

Contents:

Introduction:

What is to follow is the report of how I completed the main assignment for CS12320. I completed the brief of the assignment will some difficulty in areas but a fully-fledged product has been the result. To start with this I created a use-case diagram that would serve as the base for the design section report.

Use-Case diagram:



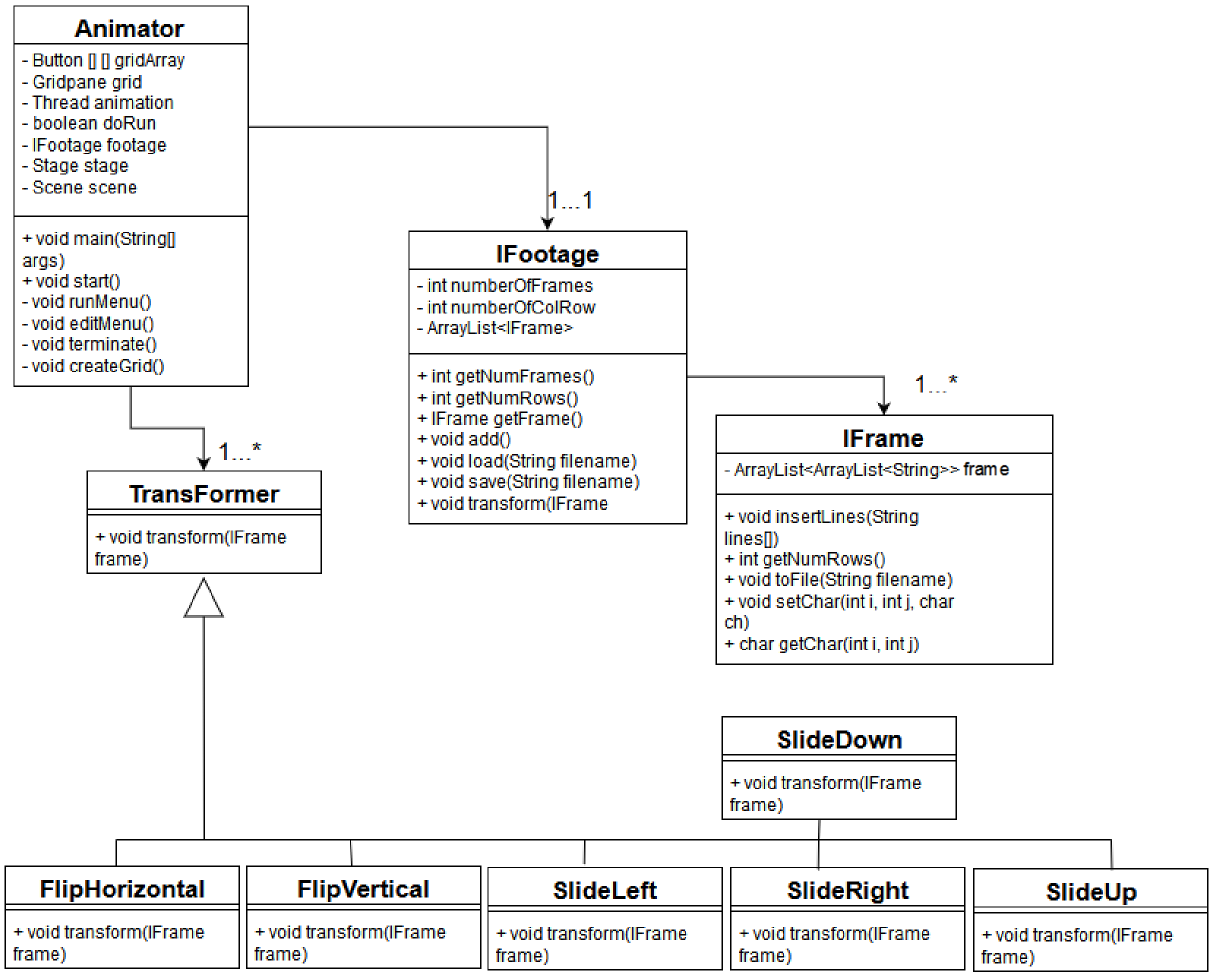
As you can see the use case diagram shows how the user should be able to interact with the menu at my first impression, and shows how they would also be able to go to the editing submenu, this however leaves out all detail on anything on the backend and thus is only so useful when in the design phase.

Design:

The earliest design decision I took was to split the transformations into a separate package so that it would be easy to distinguish the handling of the Frames and footage from the transformations, when feeding into the gui package. This made it easier during implementation.

When I started designing this project shortly after reading through the provided code I started work on a UML Class Diagram, which I find to be the best diagram for designing code and very useful to refer back to when creating the classes later down the line.

UML Class Diagram:



The diagram shows my initial design for the project as you can see there are 6 subclasses under the superclass transformer, the idea being that “TransFormer” would be implemented as an abstract super class which would never actually be called. All of these subclasses would implement the transformer interface.

The IFootage and IFrame classes were intended to be named after their respective interfaces for ease of comparison during actual production, thus they picked up that exact name just with ‘Class’ added after the names.

The intention was that Animator required an IFootage object and there could only be one of either, however there could be variable amounts of IFrame objects per IFootage allowing for expansion of the footage by just adding extra frames. This is achieved by using an ArrayList to store the IFrames.

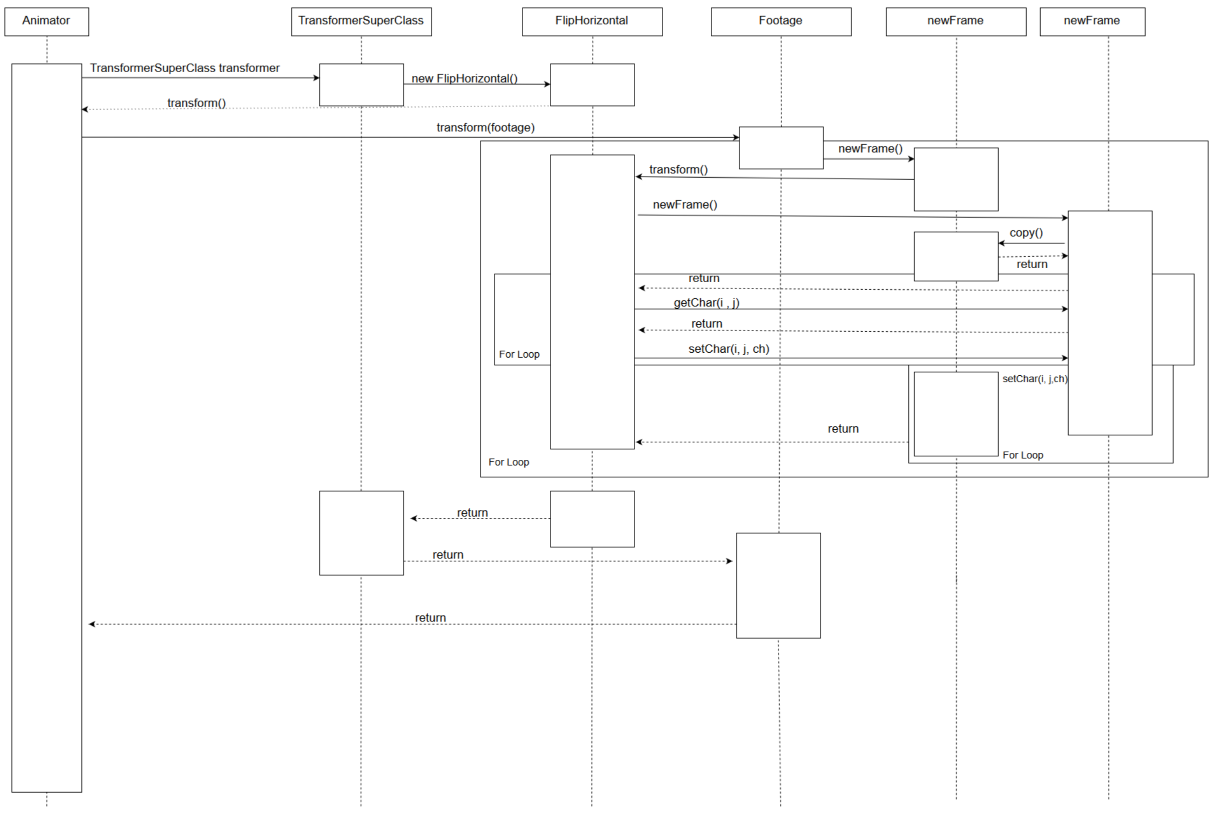
IFrame itself is a class which I found to be quite tricky to develop as a concept. It required a double nested ArrayList much like how the Button [][] Array is handled in the animator class. I didn't even know if it was possible to do this so after a bit of research I settled on the design seen. This allowed for variable sized 'rows' and 'columns' or rather variable amounts of strings in the variable list of ArrayLists containing said strings. Even though I needed to output the values from this ArrayList in most forms as a char instead of a string, it was simple to convert from a string to char by utilising charAt(0) thus allowing me to pull the value from the start of the string and since the string contained only one character, I could always pull this value as a char.

I decided early to implement a secondary menu function for the edit menu and just call that when executing the edit menu via the printMenu() function. This allowed me to work on the two independently and to keep printMenu() much shorter than it would have been otherwise and thus much easier to work on.

My decision to implement the transformer classes as subclasses came about by seeing that I would need to split up the transformations between multiple classes to make it easy in terms of declarations with the editMenu() function when actually selecting what transformations I wanted to use. Thus I thought I would be able to declare a general overarching object that could be turned into any of the subclass objects when it is selected and then the footage could be fed into it. This allows for easy implementation later.

Most other design decisions came about by trying to implement the interfaces that I was provided with, in some cases I would have utilised more functions such as a removeLines public method for the IFrame class would have been especially useful with the transformations for my initial designs.

The Sequence Diagram:

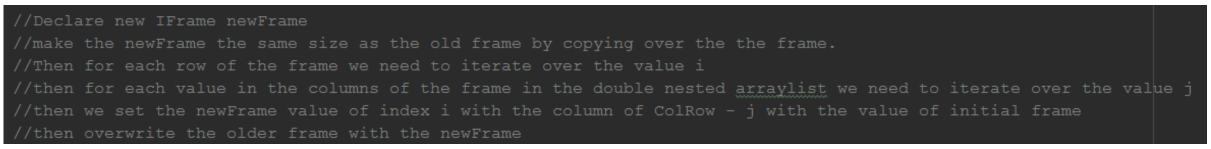


The Sequence diagram illustrates how the code deals with transforming the frame horizontally from one place to another but at the end of the day, it is done by copying it back to front from the original input, using a for loop into another frame and then it is copied back on top of the original frame. Thus we have two frames that are equal and the garbage collection picks up the secondary frame at some point from memory.

The Design of the transformations:

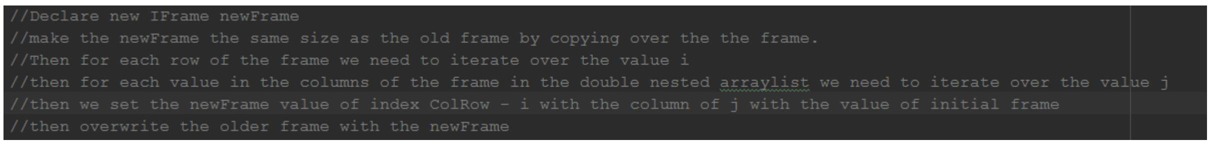
For the transformations, I used a form of pseudocode which is verging on purely English to design the transformations initially. I left the raw copies as comments above each of the transformation method implementations in the java files. I will screenshot and put them into the report.

FlipHorizontal:



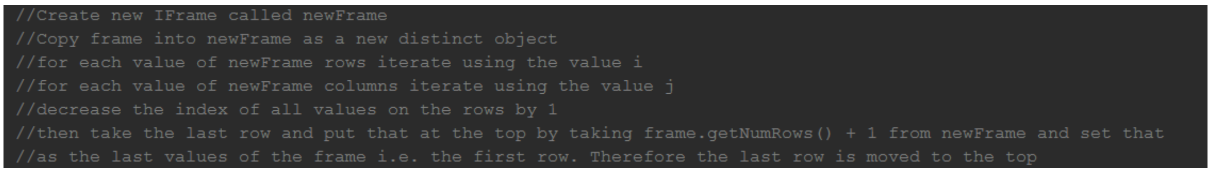
```
//Declare new IFrame newFrame
//make the newFrame the same size as the old frame by copying over the the frame.
//Then for each row of the frame we need to iterate over the value i
//then for each value in the columns of the frame in the double nested arraylist we need to iterate over the value j
//then we set the newFrame value of index i with the column of ColRow - j with the value of initial frame
//then overwrite the older frame with the newFrame
```

FlipVertical:



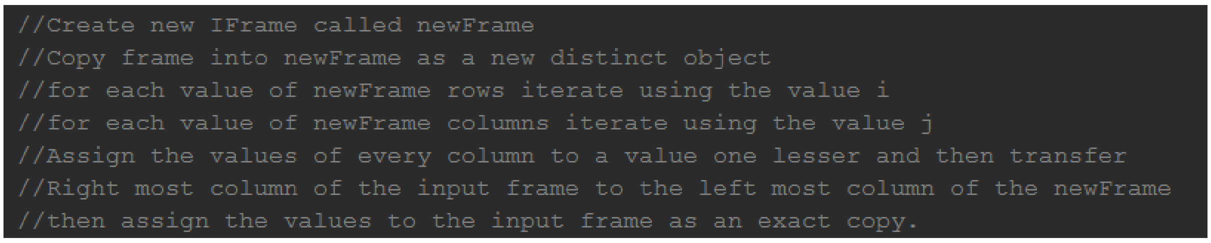
```
//Declare new IFrame newFrame
//make the newFrame the same size as the old frame by copying over the the frame.
//Then for each row of the frame we need to iterate over the value i
//then for each value in the columns of the frame in the double nested arraylist we need to iterate over the value j
//then we set the newFrame value of index ColRow - i with the column of j with the value of initial frame
//then overwrite the older frame with the newFrame
```

SlideDown:



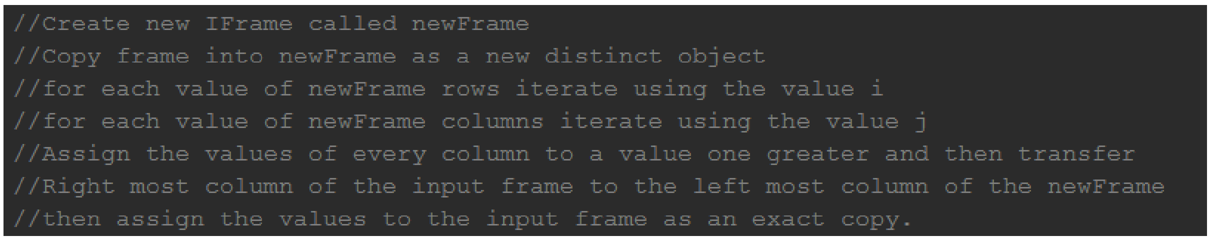
```
//Create new IFrame called newFrame
//Copy frame into newFrame as a new distinct object
//for each value of newFrame rows iterate using the value i
//for each value of newFrame columns iterate using the value j
//decrease the index of all values on the rows by 1
//then take the last row and put that at the top by taking frame.getNumRows() + 1 from newFrame and set that
//as the last values of the frame i.e. the first row. Therefore the last row is moved to the top
```

SlideLeft:



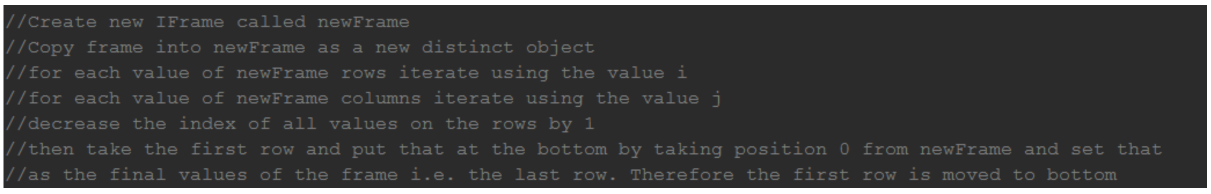
```
//Create new IFrame called newFrame
//Copy frame into newFrame as a new distinct object
//for each value of newFrame rows iterate using the value i
//for each value of newFrame columns iterate using the value j
//Assign the values of every column to a value one lesser and then transfer
//Right most column of the input frame to the left most column of the newFrame
//then assign the values to the input frame as an exact copy.
```

SlideRight:



```
//Create new IFrame called newFrame
//Copy frame into newFrame as a new distinct object
//for each value of newFrame rows iterate using the value i
//for each value of newFrame columns iterate using the value j
//Assign the values of every column to a value one greater and then transfer
//Right most column of the input frame to the left most column of the newFrame
//then assign the values to the input frame as an exact copy.
```

SlideUp:



```
//Create new IFrame called newFrame
//Copy frame into newFrame as a new distinct object
//for each value of newFrame rows iterate using the value i
//for each value of newFrame columns iterate using the value j
//decrease the index of all values on the rows by 1
//then take the first row and put that at the bottom by taking position 0 from newFrame and set that
//as the final values of the frame i.e. the last row. Therefore the first row is moved to bottom
```

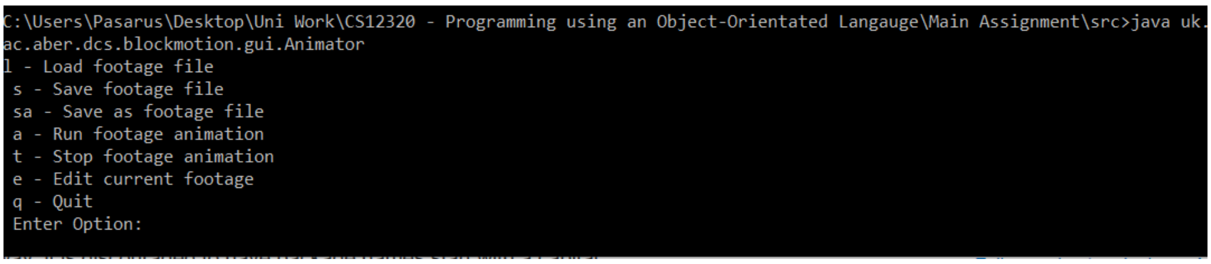
For the most part with the pseudocode you can probably tell that any vertical motion is largely the same and then any of the horizontal motion are relatively similar, whilst the same is true for the two flip transformations.

In the actual java files, you can see the transformations implemented right next to the pseudocode as some things changed from design to implantation.

Testing:

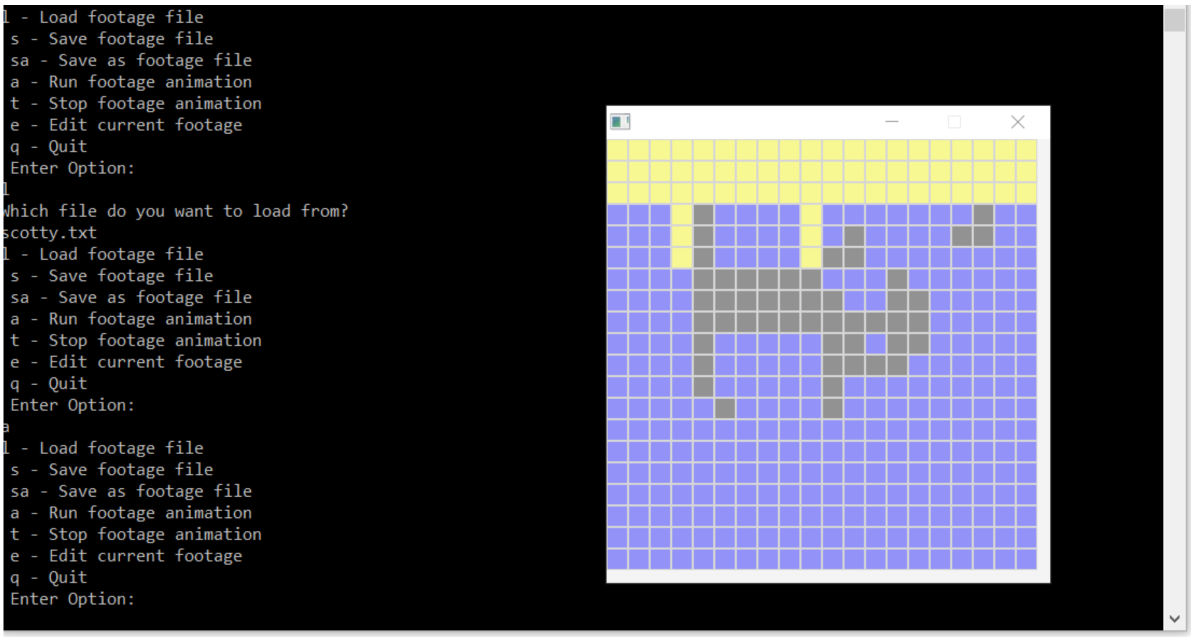
As with all programming there is the possibility of errors or inaccuracies and thus we need to test things. This section shows how I tested things and how it all worked at the time of testing.

The program running:



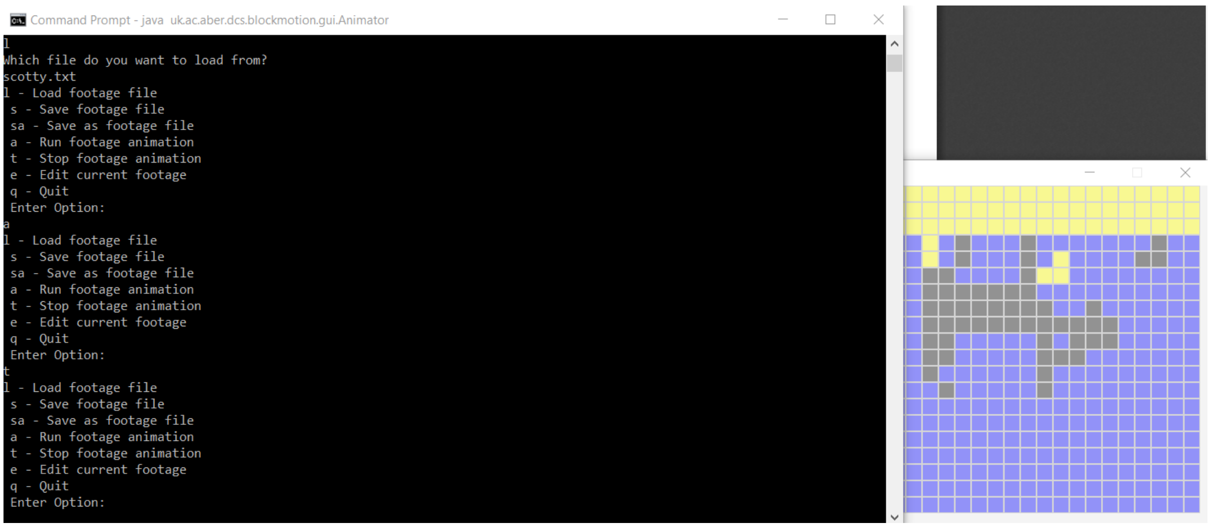
```
C:\Users\Pasarus\Desktop\Uni Work\CS12320 - Programming using an Object-Orientated Langauge\Main Assignment\src>java uk.
ac.aber.dcs.blockmotion.gui.Animator
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Load and run functioning:



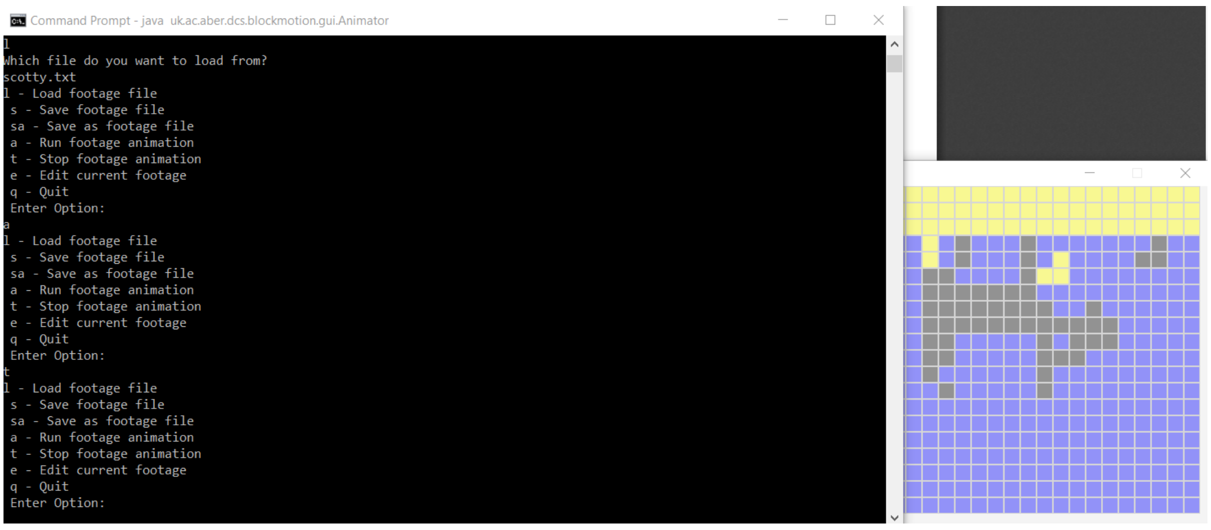
```
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
l
Which file do you want to load from?
scotty.txt
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
a
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Stop animation:



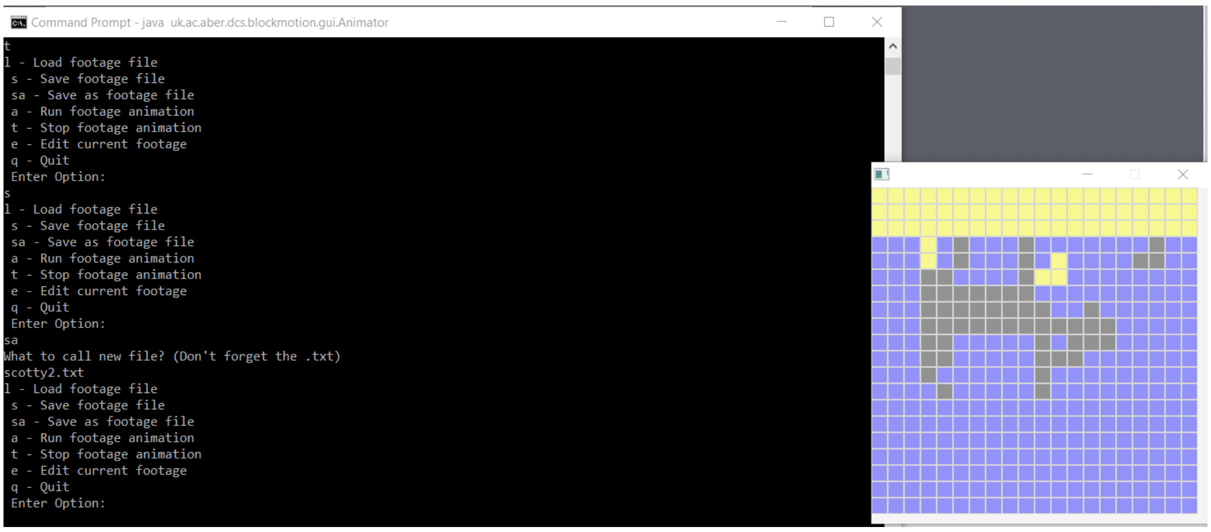
```
Command Prompt - java  uk.ac.aber.dcs.blockmotion.gui.Animator
l
Which file do you want to load from?
scotty.txt
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
a
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
t
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Saving:



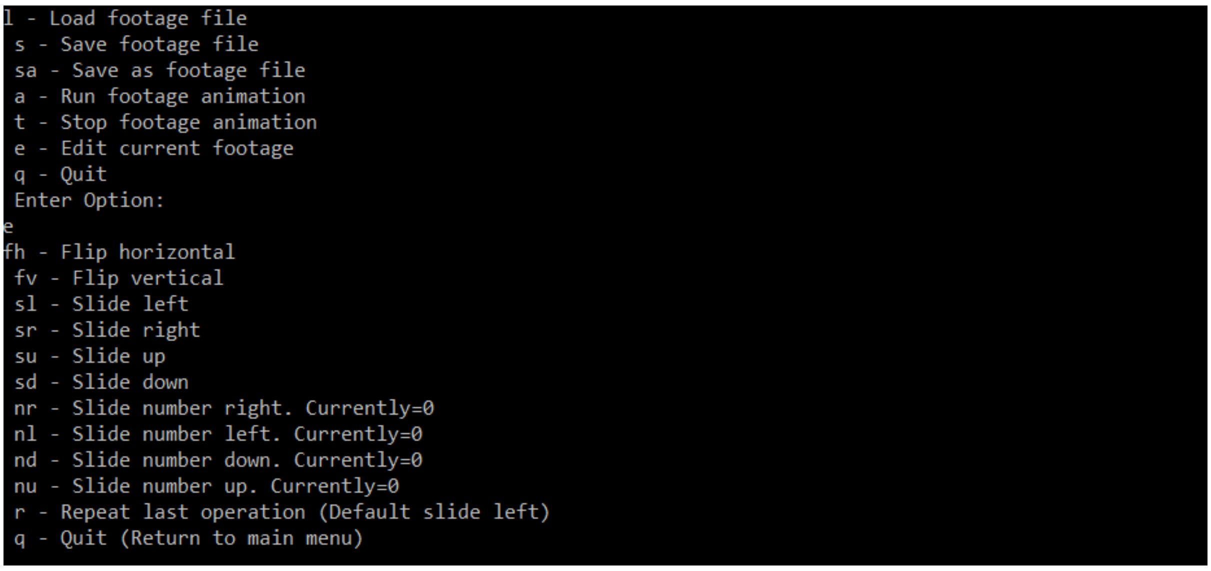
```
Command Prompt - java  uk.ac.aber.dcs.blockmotion.gui.Animator
l
Which file do you want to load from?
scotty.txt
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
a
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
t
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Save as:



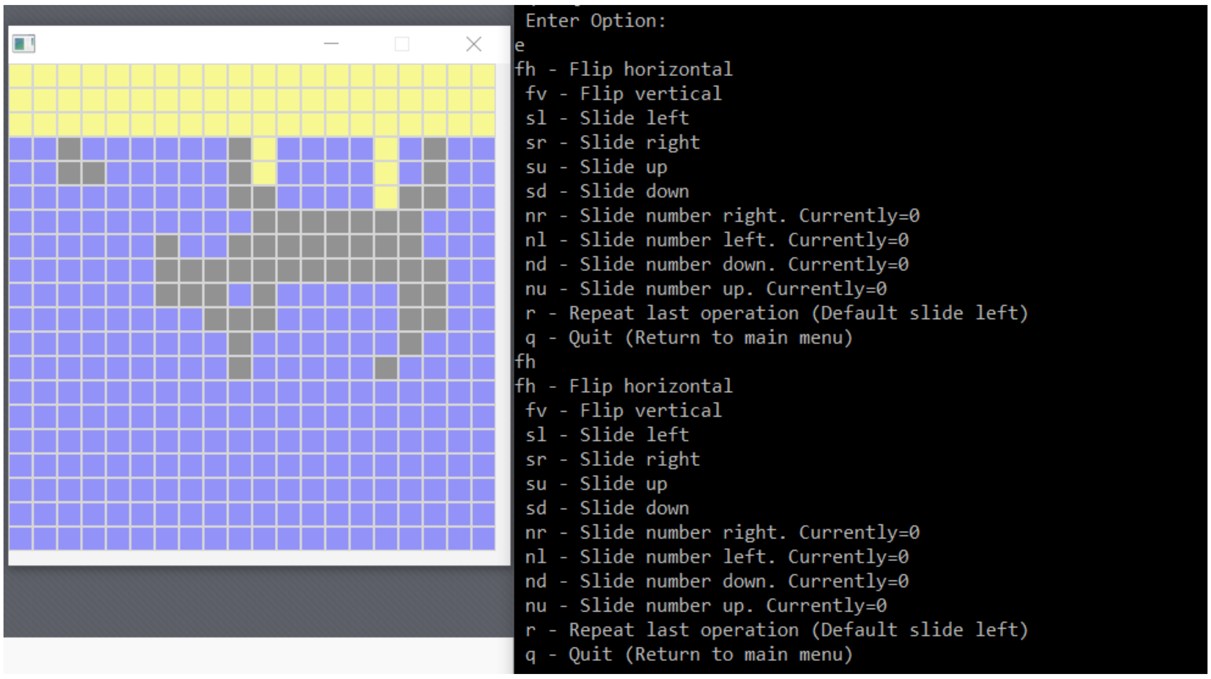
```
Command Prompt - java  uk.ac.aber.dcs.blockmotion.gui.Animator
t
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
s
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
sa
What to call new file? (Don't forget the .txt)
scotty2.txt
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Edit menu:



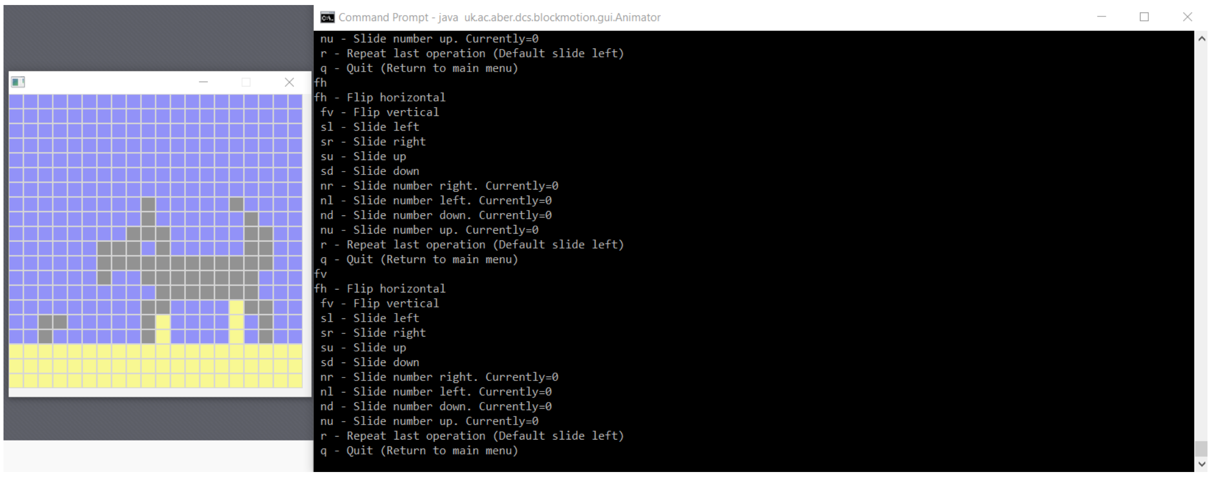
```
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
e
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Flip Horizontal:



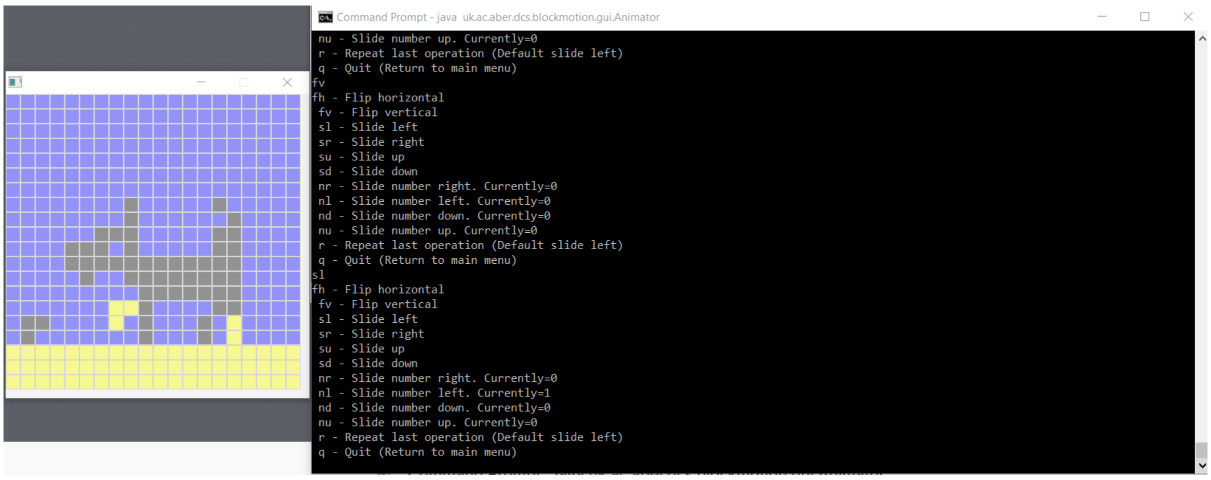
```
Enter Option:
e
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
fh
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Flip Vertical:



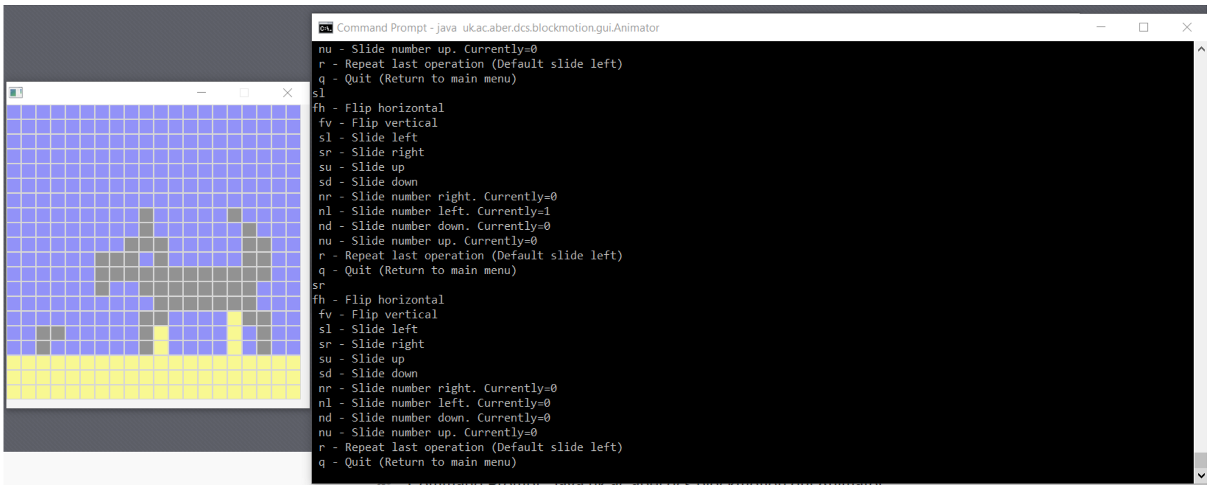
```
Command Prompt - java  uk.ac.aber.dcs.blockmotion.gui.Animator
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
fh
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
fv
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Slide left:



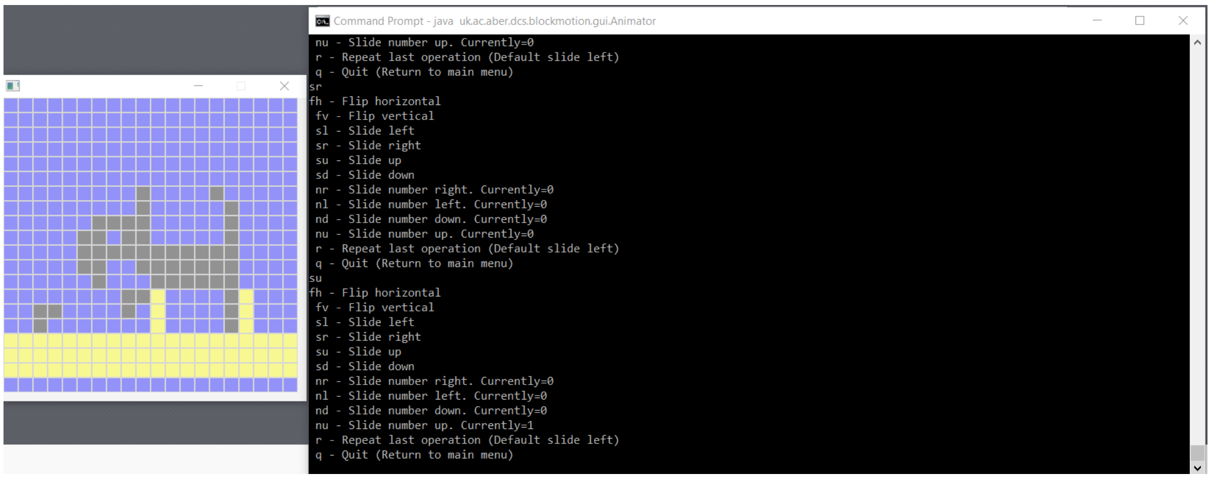
```
Command Prompt - java  uk.ac.aber.dcs.blockmotion.gui.Animator
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
fv
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sl
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=1
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Slide Right:


Command Prompt - java uk.ac.aber.dcs.blockmotion.gui.Animator

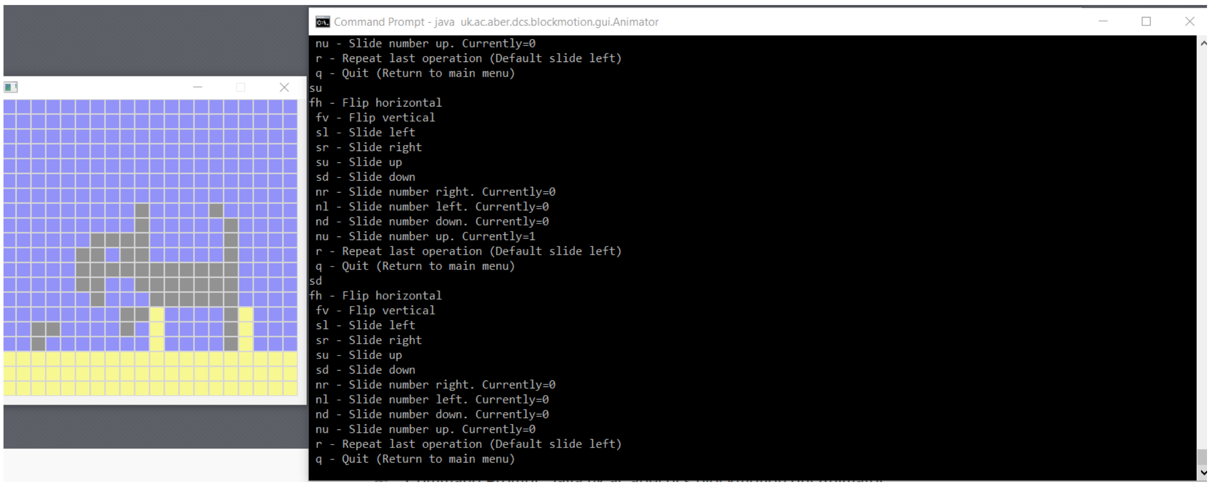
```
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sl
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=1
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sr
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Slide Up:


Command Prompt - java uk.ac.aber.dcs.blockmotion.gui.Animator

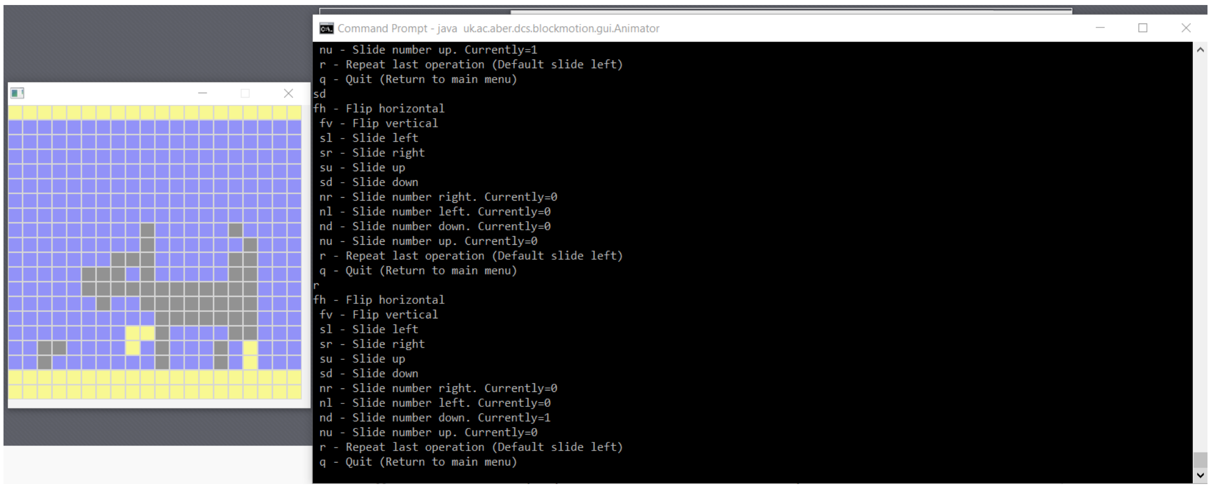
```
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sr
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
su
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=1
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

Slide Down:


Command Prompt - java uk.ac.aber.dcs.blockmotion.gui.Animator

```
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
su
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=1
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sd
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

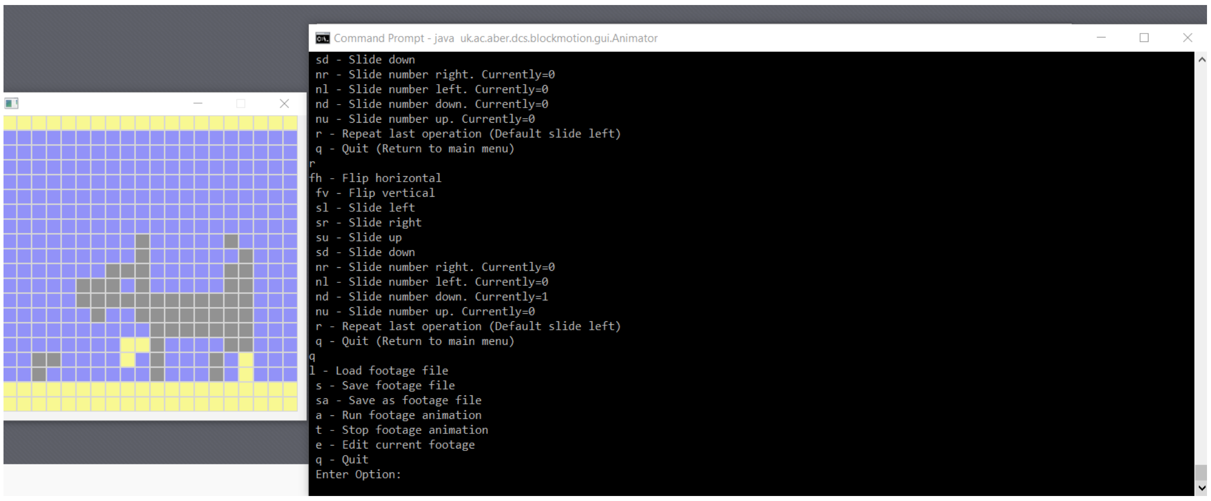
Repeat last operation:



Command Prompt - java uk.ac.aber.dcs.blockmotion.gui.Animator

```
 nu - Slide number up. Currently=1
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
sd
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
r
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=1
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
```

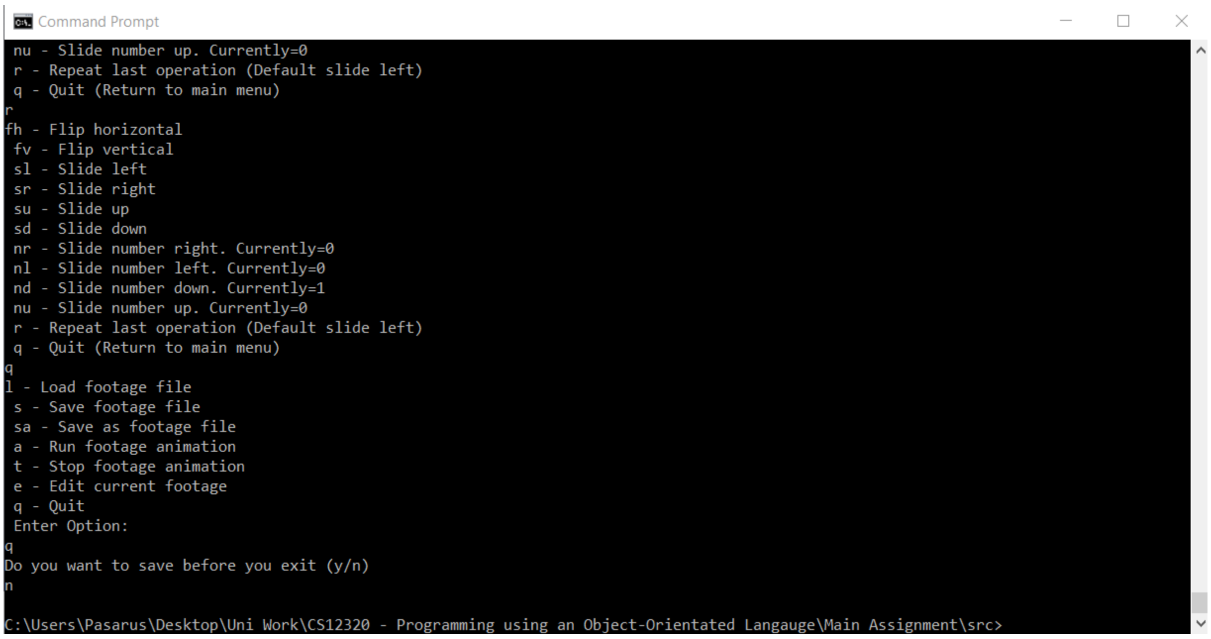
Quitting back to main menu:



Command Prompt - java uk.ac.aber.dcs.blockmotion.gui.Animator

```
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=0
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
r
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=1
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
q
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
```

Quitting the program:



```
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
r
fh - Flip horizontal
 fv - Flip vertical
 sl - Slide left
 sr - Slide right
 su - Slide up
 sd - Slide down
 nr - Slide number right. Currently=0
 nl - Slide number left. Currently=0
 nd - Slide number down. Currently=1
 nu - Slide number up. Currently=0
 r - Repeat last operation (Default slide left)
 q - Quit (Return to main menu)
q
l - Load footage file
 s - Save footage file
 sa - Save as footage file
 a - Run footage animation
 t - Stop footage animation
 e - Edit current footage
 q - Quit
 Enter Option:
q
Do you want to save before you exit (y/n)
n

C:\Users\Pasarus\Desktop\Uni Work\CS12320 - Programming using an Object-Orientated Langauge\Main Assignment\src>
```

As you can see the program has been tested to work fully in a command prompt environment however due to running it in a command prompt environment and compiling it raw from the java. The class files that are relevant are now seen in the source code part of the IntelliJ project, this has no effect on performance and functionality but it does however leave a little to be desired in terms of tidiness.

Not only was I testing with the command prompt at the end however, the debugger has been an invaluable tool when finding errors and exactly what caused them and being able to step into a function that is being called, just so I know what is being called next is one of the biggest helps.

Evaluation:

I went about completing this assignment by thinking of the main classes and producing the use case diagram which was hard having forgotten what a use case diagram looked like. Then I read through the provided code and produced a UML Class Diagram and started to produce my code and get it all hooked in together.

Starting by producing the menu and edit menu sections was a good start because I could see where other parts of the code would be used. It allowed for further thought on how to implement the transformations specifically as I could see how they would be handled and displayed in the menus. Then I moved to implementing the IFrame interface as the IFrameClass in which I had a few problems with some method implementation, for example the copy () method I originally assumed I could just return the object with "return this;" however during the production of the transformations I found this to be incorrect. When it didn't function properly I revisited and produced the code that you see in the java file now. Not quite reading the interface implementation early on was a downfall of my work. I spent hours debugging to find that I had simply not read the proper implementation of a method, with the copy method for as a prime example.

However, any issues I had with the implementation of the IFrame interface, I found that the chosen way of utilising ArrayLists to save the frame allowed me to produce footage/frames of varying sizes. Then again with the implementation of footage the usage of ArrayLists only worked favourably as I could have the footage however long I wanted.

Next I produced the IFootage class which allowed me to start putting frames together to feed into the animator. I had very few issues with implementing IFootage and no issues with saving to a file. However, I had an error with loading where I was attempting to load into a frame one smaller than required when it was declared which I fixed by just changing the size of the newly created object during the creation of the original blank frame. Other than my error with the load method I found IFootage relatively easy to implement.

The transformations had a few errors but due to careful designing originally, I had fewer than expected errors with two transformations working fully without errors straight after producing them. However, with SlideLeft and SlideRight I found it hard to move the far left/right most 'pixels' or column to the other side and originally I increased the size of the ArrayList which then caused issues as I was just increasing the size of the frame. I fixed this by just not editing the first column that I would have iterated over and copying from the input frame onto the copied frame the furthermost column, left/right respectively. Other than that, the transformations were quite pleasant to implement.

The menu() and editMenu() methods had some interesting errors that I had to resolve for example in the editMenu() I found that the repeat last operation was harder to implement than last though because I still had to have the counters tick for each one of the repeats which required me to check the class of the transformer object. This however doesn't work with 'instanceof' as a check because it would be checking to subclasses with the same superclass which meant I had to use 'class.instanceof(transform)' instead.

One problem I had that I could not fix was with the createGrid() method which meant I could not create grids of varying size as every time I called a createGrid method with any loaded footage that was not size 20 it would chuck out an error that I couldn’t figure out how to solve and this is the only thing that doesn’t work.

The only bit of the program I could attribute to “flair” might be the addition of a default in the repeatable part the edit menu, so that that function does something if nothing has occurred before.

I learnt how to properly implement and design classes from interfaces as well as how to check between different classes for a conditional statement, I am sure that I missed a few things because I was new to Java at the start of this module but if I mentioned every small bit I learnt along the way I would go over the word limit, that means this assignment was very worthwhile to complete.

What grade would I give myself? Having produced this I obviously see it as less perfect than it possibly could be. Having put very little effort into flair besides one minor addition, I would have to discount all flair marks but other than that I think that the code produced is of a decent quality and it would not be hard pushed to give a mark of 70% and up.