**Assembly language and system programming**

Assignment \_\_\_\_final project\_\_\_\_

Student Name:\_\_\_\_\_\_\_\_\_\_\_楊家安\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student ID: \_\_\_\_\_\_\_\_\_\_\_0516021\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student email address:\_\_\_\_\_\_\_\_\_\_gxxygxxy@gmail.com\_\_\_\_\_\_\_\_\_\_\_

**[10%] Introduction [ at least 100 words]**

**WORD COUNT:\_\_\_\_\_\_\_188\_\_\_\_\_\_ [ Must be filled or zero score]**

This is final project. It’s programmed with OpenGL in C++ and some assembly procedures.

The final project could show the Student ID on the screen and move horizontally with bitmaps. It could also change the point size of the pixel of the image. It could also change the image to yellow gradually or to gray, also working in the game mode. There is also a game mode: Show the grid, and you could exchange two sub-images. You could also change the size of the grid.

Here are the key usage:

‘1’, ‘2’, ‘3’, ‘4’, ‘5’: change image point size.

‘i’: show or hide the student ID, the ID will move automatically.

‘a’: change the image to the initial image and change it to yellow gradually.

‘s’: switch the image between gray image and initial image.

‘g’: toggle the game mode state. If in game mode, show the grid.

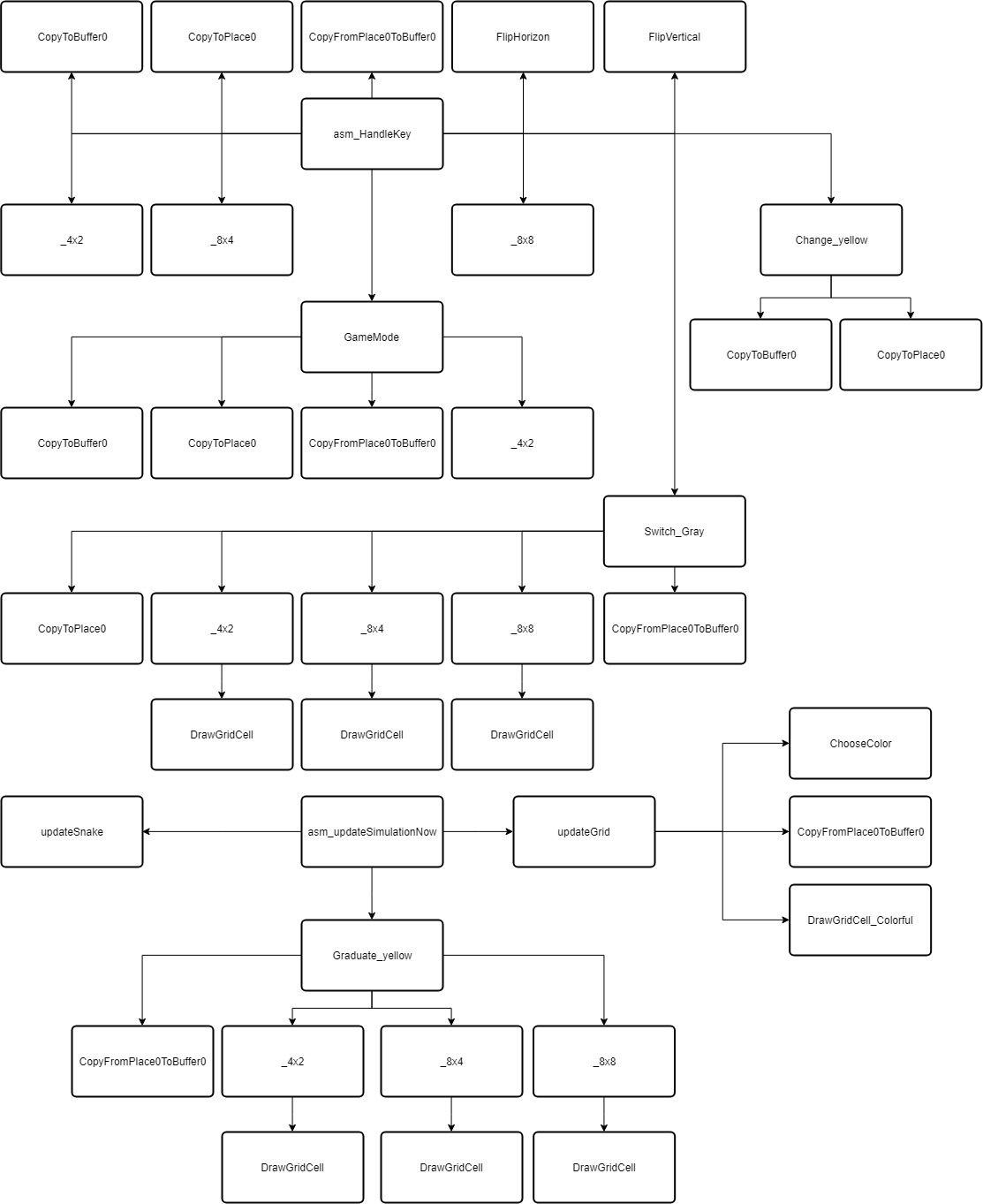
‘x’: flip the image horizontally.

‘y’: flip the image upside-down.

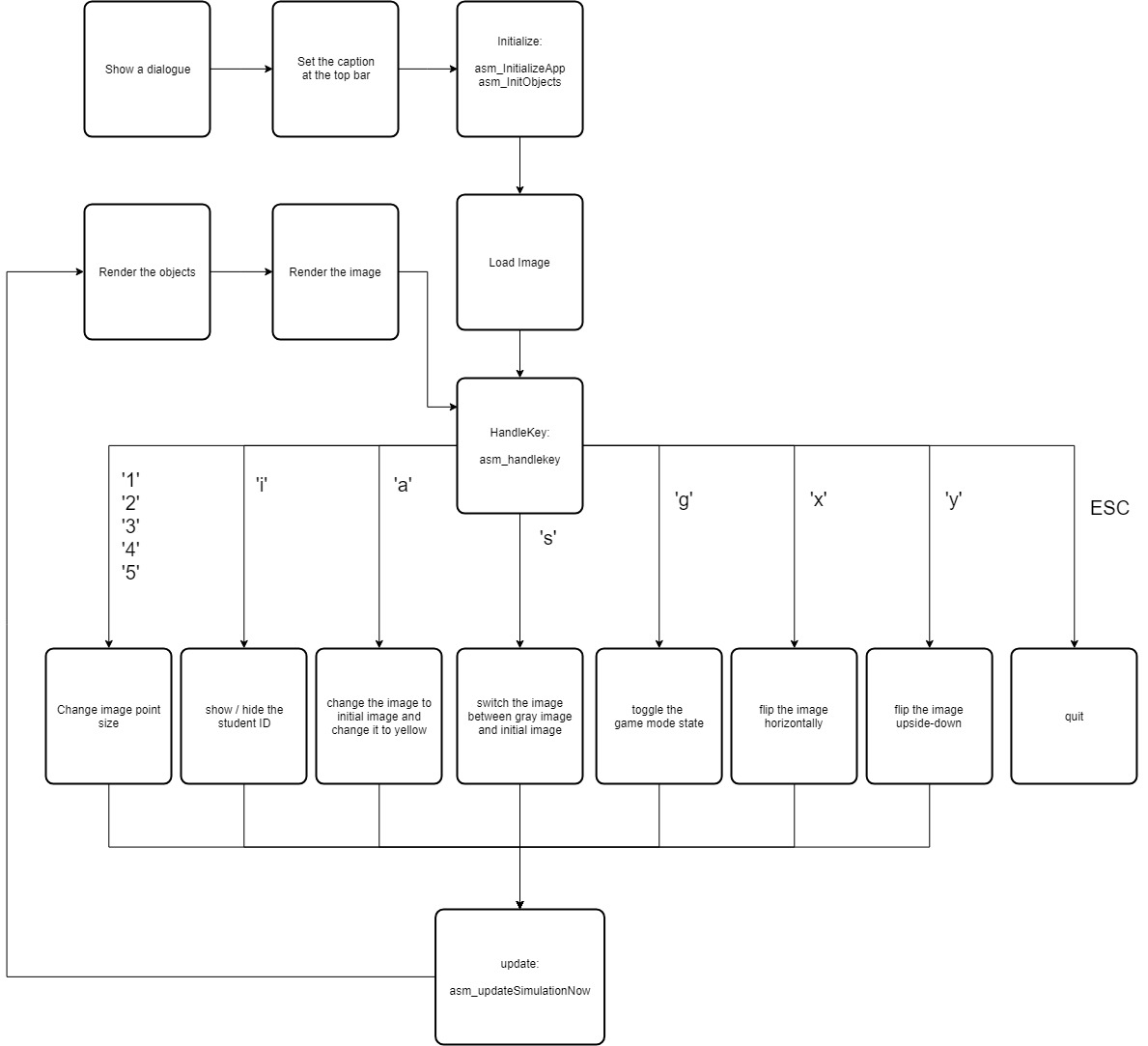
mouse: if in game mode, select and exchange the sub-images for two selected grid cells.

ESC: show the student information box and press Enter in the command line to quit.

**[10%] System Chart** [at least 10 components]



[**10%] Flow Chart [**at least 10 components**]**



**[10%] System Architecture** [**at least 100 words]**

**WORD COUNT:\_\_\_\_\_\_\_220\_\_\_\_\_\_ [ Must be filled or zero score]**

There are some procedures that already exists in the origin file:

asm\_InitializeApp: call SetTextColor, GotoXY, WriteChar, Delay, Crlf, ReadInt, MsgBox

setCursor : call GotoXY

asm\_ClearScreen: call SetTextColor, Clrscr

asm\_ShowTitle: call SetTextColor, WriteString

asm\_InitObjects: call ShowDigit\_X, ShowDigit\_Y

asm\_computeCircularPosX: no call

asm\_GetNumParticles: no call

asm\_GetParticleMaxSpeed: no call

asm\_GetParticleSize: no call

asm\_handleMousePassiveEvent: call WriteDec

asm\_handleMouseEvent: call WriteDec, Switch\_Cell

asm\_HandleKey: call CopyToBuffer0, CopyToPlace0, CopyFromPlace0ToBuffer0, \_4x2, \_8x4, \_8x8, Change\_yellow, Switch\_Gray, GameMode, FlipHorizon, FlipVertical, asm\_EndingMessage

asm\_EndingMessage: call MsgBox

asm\_SetWindowDimension: no call

asm\_GetNumOfObjects: no call

asm\_GetObjectType: no call

asm\_GetObjPosX: no call

asm\_GetObjPosY: no call

asm\_GetObjectColor: no call

asm\_ComputeRotationAngle: no call

asm\_ComputePositionX: no call

asm\_ComputePositionY: no call

ASM\_setText: call SetTextColor

asm\_ComputeParticlePosX: no call

asm\_ComputeParticlePosY: no call

asm\_updateSimulationNow: call updateSnake, Graduate\_yellow, updateGrid

asm\_SetImageInfo: no call

asm\_GetImagePixelSize: no call

asm\_GetImageStatus: no call

asm\_getImagePercentage: no call

asm\_GetImageColour: no call

asm\_getStudentInfoString: no call

asm\_GetImageDimension: no call

asm\_GetImagePos: no call

<The following procedures are what I create>

Switch\_Gray: call CopyToPlace0, \_4x2, \_8x4, \_8x8, CopyFromPlace0ToBuffer0

Change\_yellow: call CopyToBuffer0, CopyToPlace0

ShowDigit\_X: no call

ShowDigit\_Y: no call

updateSnake: no call

updateGrid: call ChooseColor, CopyFromPlace0ToBuffer0, DrawGridCell\_Colorful

\_4x2: call DrawGridCell

\_8x4: call DrawGridCell

\_8x8: call DrawGridCell

DrawGridCell: no call

DrawGridCell\_Colorful: no call

ChooseColor: no call

GameMode: call CopyToBuffer0, CopyToPlace0, CopyFromPlace0ToBuffer0, \_4x2

CopyToBuffer0: no call

CopyToPlace0: no call

CopyFromPlace0ToBuffer0: no call

CopyFromBuffer0ToPlace0: no call

Switch\_Cell: call CopyFromPlace0ToBuffer0

Graduate\_yellow: call CopyFromPlace0ToBuffer0, \_4x2, \_8x4, \_8x8

FlipHorizon: call CopyFromBuffer0ToPlace0

FlipVertical: call CopyFromBuffer0ToPlace0

**[30%] The approach [ at least 300 words]**

**WORD COUNT:\_\_\_\_\_\_\_528\_\_\_\_\_\_ [ Must be filled or zero score]**

<To read the image and show it at the background, modify the asm\_SetImageInfo and asm\_GetImageColour, asm\_GetImageDimension>

asm\_SetImageInfo:

The image total size is w\*h\*3(for R, G, B of each pixel). I use movsb to move imagePtr(esi) to mImagePtr0(edi). And move w, h, 3, 3 to mImageWidth, mImageHeight, bytesPerPixel, mBytesPerPixel respectively.

asm\_GetImageColour:

To return the RGB of the pixel, I lodsb the mImagePtr0 at the position (mImageWidth\*(mImageHeight-iy)+ix)\*3 for R,

(mImageWidth\*(mImageHeight-iy)+ix)\*3 + 1 for G

(mImageWidth\*(mImageHeight-iy)+ix)\*3 + 2 for B.

asm\_GetImageDimension:

Move mImageWidth, mImageHeight to iw, ih respectively.

<To Show the student ID with bitmap and move>

Variables: digitMaxSpeed (moving speed), digit\_Left(left-most position of student ID), digit\_Right(right-most position of student ID, objnumbers(number of shown spheres)

Flags: flag\_showdigit (To see whether the ID should be shown)

Procedures: ShowDigit\_X, ShowDigit\_Y, updateSnake

ShowDigit\_X is to judge whether the x-coordinate of the sphere should be saved in objPosX. If bitmap is 1, save to objPosX. Also compute the number of spheres shown and save in objnumbers.

ShowDigit\_Y is to judge whether the x-coordinate of the sphere should be saved in objPosY. If bitmap is 1, save to objPosY.

updateSnake is put in asm\_updateSimulationNow, to move all spheres by increasing/decreasing the numbers saved in objPosX. If touch the bound, change the moving direction.

<Change the point size of the pixel>

Change mImagePixelPointSize

<Change to yellow>

Procedures: Change\_yellow, Graduate\_yellow

First copy the initial image (in mImagePtr0) to Buffer0 and Place0.

The method to changing a pixel to yellow:

If(R > G){ R--; G++;}

If(B > 0) B--;

Graduate\_yellow is put in asm\_updateSimulationNow. Every time it do the method above on every pixel once in Place0, and copy to Buffer0. Later, draw the grid if in need.

<Change to gray>

Procedures: Switch\_Gray

The method to changing a pixel to gray:

tmp = (R+G+B)/3;

R = tmp; G = tmp; B = tmp;

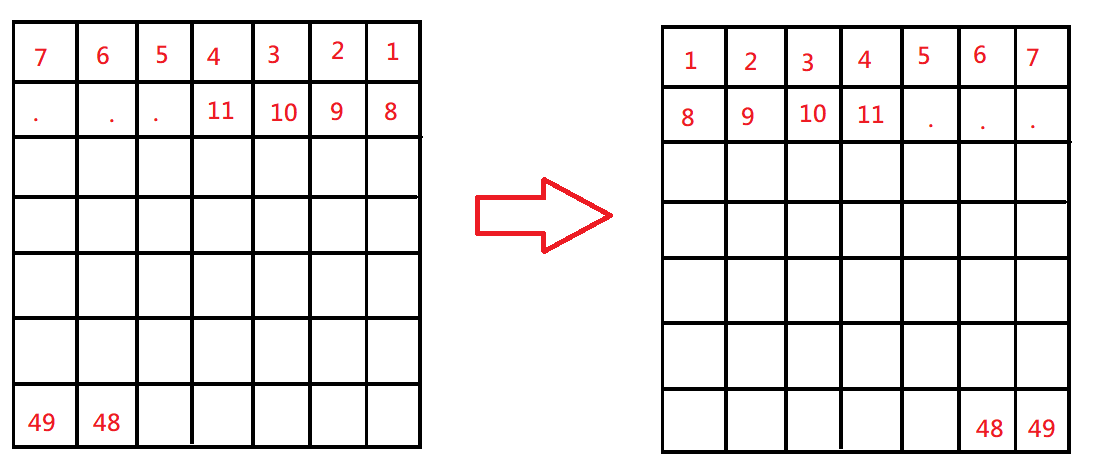
Doing the method above on every pixel in Place0 and draw the grid if in need. At the end, copy to Buffer0.

<Flip the image horizontally>

Procedures: FlipHorizon

Every number represents a pixel (3 bytes). The number also means the order.

Illustration:

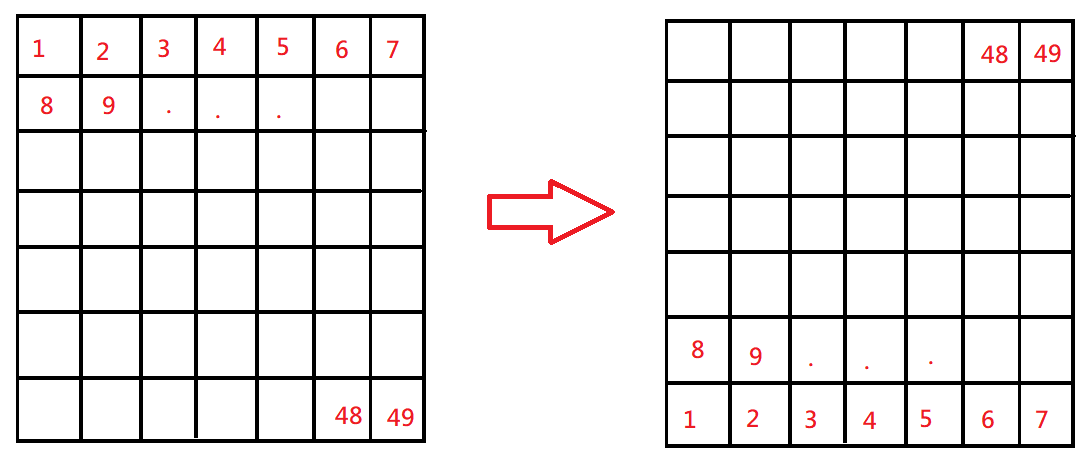


<Flip the image upside-down>

Procedures: FlipVertical

Every number represents a pixel (3 bytes). The number also means the order.

Illustration:



<Show the grid>

Important Variables: grid\_x (byte size of a cell in x-coordinate), grid\_y (byte size of a cell in y-coordinate)

Procedures: \_4x2, \_8x4, \_8x8, DrawGridCell, DrawGridCell\_Colorful

DrawGridCell will draw only one cell, edi is the upper-left address of the cell. The cell size depends on grid\_x, grid\_y.

DrawGridCell\_colorful is the colorful grid version. It is called in updateSimulationNow and every time the color is different (a cycle: red->orange->yellow->green). It will show the colorful grid cell depending on the cursor position. If the left button has been clicked once, also show that grid cell in colorful.

\_4x2, \_8x4, \_8x8: draw the whole grid on the image in different size. Method: call DrawGridCell in a loop for many times in different positions.

<Exchange two sub-images>

Procedures: Switch\_Cell

Record how many times the left button has been clicked(0 or 1 or 2), and the position cell of first time clicks and second clicks.

Use esi edi and loop to swap the two grid cell.

**[20%] Discussion/Experiments [ at least 200 words]**

**WORD COUNT:\_\_\_\_\_\_\_296\_\_\_\_\_\_ [ Must be filled or zero score]**

Many bugs happen when I doing the final project.

Sometimes the programs crash. Sometimes it shows the wrong image. Sometimes the function will not work after I press many keys.

The first mistake happens when I save the pixel RGB to the array. The array is based on 1-byte. I move the 4-byte to it by accident, just like this: mov [edi], eax. As a consequence, some pixels on image becomes black. It takes me a little time to discover the mistake at first.

The second mistake happens when I change the image to gray. The formula is: (R+G+B)/3 and move to R, G, B. Because I know it will overflow when adding, I extend it. However, I use cbw. I didn’t notice that the R, G, B is unsigned and the cbw will make mistakes. Finally, I use movzx to solve the problem. However, there is a similar problem I didn’t discover. I use jg instead of ja. jg is for signed and I should use ja for unsigned. The bug will show a very strange image.

The third mistake happens when I change the image to gray or yellow in the game mode or entering the game mode. The grid will not be shown when I change the color in game mode, because I only show the grid originally when the cursor goes to another grid cell. The bug happens due to the whole structure, and it takes many times to fix the bug.

There are still other small bugs when doing the projects, but solved in little time. At first the bug is easily found and fixed, however, as time goes by, the bug is more and more difficult to find because there are too many codes. Finally, all of them are fixed.

**[10%] Conclusion [ at least 100 words]**

**WORD COUNT:\_\_\_\_\_\_\_134\_\_\_\_\_\_ [ Must be filled or zero score]**

I learn a lot in the project.

There are many codes in the program, and I notice the modularization of the program is very important in huge project. There are a few reasons. One is, to make the code clear because sometimes a part of codes are used lots of times, so we can modularize it to make the code clear. Another is, it makes the debug easier because we just need to modify the module. The other is, we can easily change the structure by modularization.

I spend much time thinking the whole structure, because if the structure is incorrect, it may be hard to modify. Sometimes it have to modify by deleting most of the codes. Also, if the structure is incorrect, the more I program, the more bugs there will happen.