Michael P, Weihan L, Rueban R.

**ICS 3UI Graphics Project Journal**

**Graphics Version 1:**

The objective for the first version of our graphics project was to create a frame titled “Le thing” with a menu bar. The frame would contain a panel in its south region. In the panel there would be a button, which would manipulate a text field. The menu bar was to have one drop down menu titled “Options”. It would contain three items, an “Exit” button, a “Credits Button” button, and a “Help” button. The “Exit” button would close the window and end the program if clicked. The “Help” button would output “You figure it out” to the console if clicked. Finally, the “Credits” button would output “Bonjour” to the console if clicked. In the panel, there is supposed to be a button, which says “Insult me!”. There is also supposed to be a text field, originally reading “100”. This is eventually supposed to represent the player’s health. If clicked, the button alters the text field, subtracting 10 from whatever value is currently being displayed. When the health displayed in the text field reaches zero, the text field is changed to say “you died”.

We encountered few difficulties during this phase as none of the things we were attempting to do were particularly difficult and our program did not have very many files being used yet. The only difficulty we encountered was that we were unable to put the menu bar in a panel, which we learned was not possible. Michael obtained the code for the menu bar from a Youtube video: <http://www.youtube.com/watch?v=j6LVBEikup4>

menubar = new JMenuBar();

setJMenuBar(menubar);

Options = new JMenu ("Options");

menubar.add(Options);

miExit = new JMenuItem("Exit");

Options.add(miExit);

Exit exits = new Exit();

miExit.addActionListener(exits);

This is a snippet of code from our menu bar. The menu bar is set up in the main frame (FraMain). The options tab is then added to the menubar, and the Exit menu item is added to the Options tab. An ActionListener is then added to the exit tab, causing the window to close if it is clicked.

**Graphics Version 2:**

This second version of our program mirrors the first with a few changes. A centre panel is added. In this panel an image is loaded. The image is not scaled to the size of the panel. It is presented at a specified size and location in the panel.

This version of our program was slightly more complicated than the first as it incorporated a centre panel. One of the problems we encountered was getting the image to stop scaling to the size of the panel. This is a snippet of our code. To make it easier to read, we have removed commented code.

public PanCenter(){

setPreferredSize( new Dimension( 300, 304 ) ); //IMAGE IS SCALED TO FIT

setMaximumSize(new Dimension (300,304));

}

@Override

public void paint(Graphics g) {

super.paint(g);

Dimension size = new Dimension(img.getWidth(null), img.getHeight(null));

setSize(size);

g.drawImage(img, 0, 0, getWidth(), getHeight(), this); //MOVING DRAWIMAGE DOWN HERE MAKES SO IMAGE IS DRAWN WITH ABOVE DIMENSIONS

}

In this snippet, we set the size of our panel to be 300 x 304 no matter what size the frame is. If we just draw the image from there, it is scaled so that it will fit perfectly in the panel. Because our intent is to have the background move instead of the character, we want the background to stay its original size no matter what. This was accomplished by setting the dimension size (Dimension size = new Dimension…). We then draw the image at coordinates 0,0 at the original size.

**Graphics Version 3:**

The third version of our graphics project is dedicated to putting a second image (the hero) on top of our first one (the background). The first thing that we tried was to put the hero image on a separate panel from the background image, and layer the panels so that we could see both images. We also decided to increase the size of our centre panel so that we could see both images.

One of the main difficulties we had in this version of our program was layering the panels so that we could see one panel on top of the other without obscuring the one below it. In this case, despite the image having no background, the panel with the hero on it was obscuring the panel containing the image of the background. We also realized at this point that it would likely be difficult to facilitate hit detection between objects on different panels. We decided to put both images on one panel, layering the hero on top of the background. Another problem we encountered when creating the program was that when we loaded the two images in, they obscured the button and text field in the south panel. We are unsure why this problem did not occur when we loaded our first image in, however the below code illustrates how we fixed the issue:

setDoubleBuffered(true);

setOpaque(false);

These two lines are from the PanCenter constructor. The first is allowing for double buffering (one image is drawn on the screen while one is drawn in the processor cache). This is intended to make the movement of the objects smoother in future releases, but has no purpose now. The second line is to allow other objects like the south panel (the one containing the button and text field) to show through in front of the images.

**Graphics Version 4:**

The fourth iteration of our program allowed us to move the background in all directions with the arrow keys, including diagonal (ie moving in the direction of more than one arrow key at a time). The concept of having something being moved by arrow keys was created in a scratch program called “Move\_Ball” by Weihan. The first version of the program only worked with one arrow key at a time, meaning that the image (in this case a blue ball). Michael edited the program to allow for diagonal movement in scratch program “Move Ball Diagonals” and put it into this release of the program.

Relative to some of the other releases, Version 4 was quite easy to create as Weihan already had the code for the key based movement somewhat complete. We did not run into any troubles in this version.

public void keyPressed(KeyEvent e) {

int code = e.getKeyCode();

if (code == KeyEvent.VK\_UP) {

if (velY == 0) {

velY += -2;

}

if (velY < 0) {

velY += 0;

}

if (velY > 0) {

velY += -4;

}

}

if (code == KeyEvent.VK\_DOWN) {

if (velY == 0) {

velY += 2;

}

if (velY < 0) {

velY += 4;

}

if (velY > 0) {

velY += 0;

}

}

if (code == KeyEvent.VK\_LEFT) {

if (velX == 0) {

velX -= 2;

}

if (velX > 0) {

velX -= 4;

}

if (velX < 0) {

velX -= 0;

}

}

if (code == KeyEvent.VK\_RIGHT) {

if (velX == 0) {

velX += 2;

}

if (velX < 0) {

velX += 4;

}

if (velX > 0) {

velX += 0;

}

}

}

public void keyReleased(KeyEvent e) {

int code = e.getKeyCode();

if (code == KeyEvent.VK\_UP) {

velY = 0;

}

if (code == KeyEvent.VK\_DOWN) {

velY = 0;

}

if (code == KeyEvent.VK\_LEFT) {

velX = 0;

}

if (code == KeyEvent.VK\_RIGHT) {

velX = 0;

}

}

This snippet of code is the edited picture movement code. At this point, the background is controlled by the arrow keys. The background moves in the direction that the arrow key is indicating (Pressing the left key moves the background left). When the key is released, the velocity in that direction is reset to 0.

**Graphics Version 5:**

For the fifth version of our game, we focused on implementing a ground. We create a bright green rectangle and hit test it. If the ground and the hero are in contact, the hero will not continue to descend through it. Additionally, the key bindings have been changed. To make the game feel as if the hero is moving, the background will now move in the opposite direction that the key pressed is directing it to. Controls are now as follows:

* D – Moves the background left (looks like the hero is moving right)
* A – Moves the background right (looks like the hero is moving left)
* SPACE – Moves the background down (looks like the character is moving up)
* DOWN ARROW – Moves background up (looks like the character is moving down)

This idea of hit testing the ground was done in Weihan’s scratch program titled “Ground\_Hit\_Test”. Weihan used a version of hit testing where he checked the Y coordinate of the ball in the program in relation to the Y coordinate of the top of the rectangle that represented ground. For the final version 5 release, Michael altered the design so that the program checks if one image is physically overlapping the other. This allows for things like jumping off of ledges. At this point in the game, hit testing is just a Boolean check for any collision between the two objects. The X coordinate is not yet involved. As a result, running into the side of the green box will teleport you to the top of the box.

Again, because Weihan was able to put the concept of the hit test into a scratch program, It was relatively easy for Michael to adjust and fix it so that it would work with Version 5 of the program. One difficulty that he experienced during the creation of the Ground in the program was that because Weihan’s program was using the ball as the moving entity while the final release used the background as the moving entity, all of the directions needed to be reversed for the code to work correctly. Additionally, the hit testing method needed to be altered for greater flexibility.

x += velX;

y += velY;

Shape Rect2 = new Rectangle2D.Double(199,283,200,200);

Shape Rect = new Rectangle2D.Double(x, y+440, 600, 900);

if (Rect.intersects((Rectangle2D) Rect2)){

i+=1;

System.out.println("Hit Number:" +i);

System.out.println(y);

y = 42;

}

This snippet of code is the hit test code used in this version of our program. A rectangle is made around both the hero and the ground (the green rectangle). If the 2 rectangles intersect, the Y coordinate is reset to 42. This will need to be changed in later versions if we want to add more levels to our game (for example, landing on a ledge will reset the Y coordinate to a different value than landing on the ground). Hopefully this will be handled with an array of boxes, and each one can be hit tested with the hero. Additionally, the boxes will have to be horizontally hit tested as well as vertically hit tested so that the hero does not run into the side of the terrain and be transported on top of it. This is not anticipated to be done in the next release.

**Graphics Version 6:**

For the sixth version of our program, we worked on implementing gravity. This was first also accomplished by Weihan in the scratch program titled: “Ground\_Hit\_Test”. This method of creating gravity was edited by Michael. It was still flawed. Rueban created the final method for gravity in scratch program “Gravity\_Test”. It was changed slightly and implemented by Michael. In this release, when the space bar is pressed, the background is moved in a parabolic trajectory, making it appear as if the character is jumping.

Because the initial logic for gravity was already created in a scratch program, it was relatively easy to implement. One issue we encountered was that we were having difficulty getting the hero to jump continuously when the SPACE button was held down. Another issue we encountered with this version of the program was the trampoline effect, where the hero would jump higher and higher if the SPACE button was held down.

if (nGrav<5){

nGrav+=1;

}

velY-=nGrav;

x += velX; //FIXED RUEBAN'S CODE, TAKEN FROM: http://www.2oi.sgrondin.ca/ss15/motion.html

y += velY;

if (y == 42){ //CURRENTLY HARDCODE, IF WE WANT MULTIPLE LEVELS, WE NEED TO FIX THIS

nGrav = 0; //STILL FULL OF BUGS... NEED TO FIX

velY = 0;

}

This is part of the code we used to implement gravity. Our movement code works by creating a velocity which we add to the X or Y value. Gravity acts as a constant velocity downwards. It is subtracted from the Y velocity before the position of the images is adjusted. We put 4 pixels per clock cycle as our maximum speed for falling to avoid the jumping animation looking choppy (without this, the pixel gap between the times we drew the background could get larger indefinitely). In the next release we hope to iron out the bugs mentioned above and generally polish the game, hopefully making the large green block used as ground transparent, and adding a new background. We also hope to make our program better annotated and more readable.

**Graphics Version 7:**

Due to time limitations, this version of our program does not differ very much from the 6th version. In the previous version, we were having difficulty getting the hero to jump continuously when the SPACE button was held down. Michael and Rueban realized that these problems were because the hero did not ever reach the ground value. Rueban then fixed the problem by changing the ground value to <42 instead of ==42. Additionally, Weihan created a new background for our game. We imported this, and synchronized the ground level in the picture with the green rectangle. We then altered the size of the rectangle to make it much thinner so that there was less of a chance that the hero would run into the side of the rectangle and be transported to the top it. The width of the rectangle was also increased so that it would stretch the width of the picture. The rectangle was then set to be invisible to make the game look better. One but we still have in our code was that when the character jumped off of the ground level, it fell very slowly. In the next release, we hope to fix this bug and add another key feature, which will eventually become the character’s weapon.

**Graphics Final Version:**

Weihan cleaned up the code for the final program, getting rid of commented out code and adding spaces/annotations to make it easier to read. He also added the new key feature. For this version of the program, pressing “E” will make the character swing his sword. Weihan also created a new graphic for the hero. Michael fixed the bug by removing the ground value. The ground in the program is now governed only by hitboxes. Rueban wrote the final graphics report while Michael wrote the project journal.

While the game is stable as far as we have tested it, there are a few features that we need to change. One of them is that if the “Insult me!” button is clicked, the game becomes unresponsive until the user presses TAB once. This is because of this piece of code:

setFocusable(true);

This piece of our program allows the program to focus specifically on one object at a time. It allows us to focus on the background and move it with the keys. Without this piece of code, we are unable to move the background at all. However, clicking the button changes the focus of the program to the button, and key presses to move the background are ignored. When TAB is pressed, the focus of the program is changed and the background becomes responsive again. This feature will not be an issue in the future as health will eventually be governed from the main panel through hit testing. There will be no button.

**Conclusion:**

During the time we worked on the project, we created a frame with a title, and two panels with in the frame. We created a button that controlled a label, and we imported 2 images layered on top of each other. We allowed the background to be controlled by keyboard buttons, and implemented a ground so that the background could only travel so far in one direction. We also implemented gravity.

The three of us hope to work on this project during the summer. We hope to add more platforms, projectiles launched from the hero’s weapon, animated movement, and hopefully enemies. We also hope to add in the concept of health.