Michael Camara

Honor Code Pledge: This work is mine unless otherwise cited

CMPSC 382

Due Date: 12/8/15

Final Project Report: LaserHorse

LaserHorse is an Android application whereby the user controls a horse that shoots lasers from its eyes. Although the concept is certainly abstract, and perhaps slightly ludicrous, it serves as a platform for demonstrating a variety of visual computing techniques in a dynamic setting. The design of this application attempts to capture the user's attention, provide a continually entertaining experience, and ensure that all of the features of the game are easy to understand. By making appropriate color choices and presenting data clearly, LaserHorse hopes to deliver on these goals for its many potential users.

The premise and controls for the game are relatively simple. The user controls a small horse that can be moved by touching and dragging a finger on the screen of an Android device. As long as the user continues to touch the screen, the horse under the user's finger will continue to shoot red, rectangular lasers from left to right in regular intervals. Additionally, filled circles of different colors appear randomly from the right side of the screen and gradually move to the left. The basic goal is for the user to shoot lasers at these circles to destroy them, which increments a visible score counter. However, if one of these circles collides with the left edge of the screen then the user will lose points; and if a circle collides with the horse then it will lose one "life." The game ends when the horse runs out of lives, at which point the user's final score is displayed prominently and the game can be restarted if desired. Although additional elements are incorporated in the game, these highlight the fundamental components.

In order to create this game, the Eclipse IDE was utilized with the official Android SDK plugin. This allowed for the creation of an Android project, complete with many of the backend files necessary for an Android program to run as intended. Although the Android API is vast and complex in places, the core functionality of running an Android program can be attributed to two important classes: Activity and View. An Activity is similar to a JFrame window from the Java AWT library: it provides a parent container that can house a number of different components inside of it. The Activity is responsible for much of the higher level details of the program, controlling various layout and control flow options, partially through linked XML files that are highly configurable. A View is then similar to a JPanel in Java, providing more focus and control over how individual elements are displayed. These two types of classes were created in LaserHorse, named MainActivity and CustomView, respectively. MainActivity helps launch the application when it is opened, and then CustomView handles the interactive gameplay. Although significantly more Android classes could have been used to create a comprehensive game framework, these two were selected to provide enough abstraction from the complexities of Android development such that more work could be focused on the coding itself.

CustomView contains two methods that affect the control flow of the program and handle user input: onDraw() and onTouchEvent(). The latter method is triggered whenever the user touches the screen in some way, although the exact type of touch gesture can be further isolated if desired. This method is mainly responsible for recording the X and Y positions on the screen where the user touches, as well as controlling a Boolean value that indicates whether the user is continually touching the screen. The onDraw() method then uses this information to begin a long sequence of events involving multiple different classes, including: Horse, Laser, Orb, Powerup, CollisionDetector, and GameUI.

The Horse class is arguably one of the most pivotal parts of the program, as it essentially acts as the user's avatar in the game. Since the Swing and AWT Java libraries are not included in the Android API, several other classes and methods had to be utilized to draw a horse image. The two most fundamental graphical classes are Paint and Canvas. Paint handles many of the characteristics of the user's virtual "brush," such as color, style, text size, and more. Canvas then uses the information stored in Paint to actually draw an image to screen. This class has many built in methods for creating simple objects like rectangles and circles; but it additionally allows one to draw another type of object called a Path. Paths can be used to create shapes, similar to how custom shapes are created in Processing: vertices with X and Y positions are specified during creation of a Path, and the Canvas then draws the path by connecting those vertices together. Many of these Path objects are created in the Horse class during initialization, and then they are stylized with a Paint object and finally drawn to the screen with a Canvas object. The CustomView class then uses scaling and translation to resize and move the horse to where the user is touching.

While the user touches the screen, Laser objects are continually created from the horse's eyes and projected to the right of the screen. These Laser objects are bright, highly saturated red rectangles, which the user tries to direct at likewise bright and highly saturated Orb objects. These high saturation values were chosen to highlight the active, interactive parts of the program and provide sufficient contrast with the background. The Orb objects are further assigned a random hue during initialization, the values of which are spread apart such that there can be sufficient distinction between them. These orbs move from right to left, and when a Laser object makes contact with it (as determined by the CollisionDetector class), it is removed and a large, unsaturated circular area of the same hue is created on the Background object behind it. As the

user plays the game and "pops" many of these orbs, the background becomes filled with these unsaturated, pastel splashes, creating unique combinations of colors each time. Even though the newly created background colors share the same hue as some of the orbs, the difference in their saturation values provides ample contrast to allow distinction between the interactive and non-interactive elements. Additionally, the background colors move to the left at a slower speed than the orbs, creating a motion parallax effect that further aids in discerning foreground and background elements.

In order to add more interactivity and objectives to the game, a Powerup class was created that improves the performance of the horse. These powerups are shaped like stars and have a brightly saturated green color. When the horse collides with a powerup, the frequency of its lasers decreases, such that it can shoot more lasers per interval of time. Powerups are essential for the user to get a high score in the game, and thus the player should be able to detect them very quickly. For this reason, knowledge of pre-attentive processes was utilized during implementation of this class. First, as previously mentioned, the Powerup object has a unique star shape, clearly different from the circular Orb or rectangular Laser objects. Additionally, while the orbs and lasers move across the horizontal plane, the powerups move along the vertical plane. A rotational transform is further applied to continually spin the star shape, such that this combinations of distinct movement should immediately grab the user's attention and give them sufficient time to intercept the powerup before it moves off screen.

The GameUI class controls how certain data is displayed to the user. Specifically, it displays the number of "lives" the horses has left, the number of powerups or "power level" of the horse, and the user's current score. The horse begins with five lives and loses one whenever it contacts an orb, and the player loses the game when there are no lives remaining. In order to

display all of this information, the GameUI first creates a white rectangle on a thin upper portion of the screen, extending the entire width of the screen. This is done in order to enhance the luminance contrast between the background and any text that is displayed on top of it, as all of the text has dark lettering. However, the rectangle is further given a slightly transparent alpha level, such that gameplay elements can still be partially observed below it.

Next, the data for lives is placed in the top left part of the screen, power in the top middle, and score in the top right. These positions, particularly for lives and score, have been standardized in many videogames, and thus will help the user immediately recognize their meaning. While the score is then displayed as a simple numeric counter, the displays for lives and power use a combination of "X" and "_" characters: the "X" symbolizes a life or a powerup, while a "_" symbolizes a lost life or missing powerup. Although numeric counters could be used for this data, these kinds of glyphs may help the user understand their contextual information better without needing to stop and read them. When the user observes many "_" and few "X" in the life bar, they know they only have few chances remaining; while observing many remaining "_" in the power bar shows that the user can still collect more powerups before reaching the maximum power level.

By using theories and techniques from visual computing, each class aims to improve upon the user experience by emphasizing relevant information without detracting from the fun of the game. Additional features may still be added to LaserHorse, such as additional types of powerups or different characteristics to distinguish hues of orbs. However, the current version of LaserHorse is fully functional, and with some additional polish can hopefully be released to the public.