Mobile Application Programming: iOS

CS4962 Spring 2016
Vertical Shooter Project Final
Due: 11:59PM Monday, May 2nd

Abstract

This document describes the default final project option for CS4962. It is an iOS OpenGL Swift implementation of a vertical scrolling shooter in portrait orientation. The game consists of a player-controlled craft which finds itself encountering many enemies trying to destroy it. The player attempts to survive as long as they can as enemies present themselves. A menu presenting the name of the game, theme graphic, and options, introduces the game to the user. Options will include Start Game, Resume Game, High Scores. The game's theming is up to you, using whatever venue you choose, be it space ships, world war 2 fighter planes, or flying magic unicorns in the clouds.

In game, enemies appear from the edges of the screen following linear and non-linear paths around the screen until finally exiting the screen. The player must avoid the enemies, allowing them to exit the screen, or destroy them by shooting them with projectile weapons mounted on the craft. Some enemies will be destroyed with a single impact, while others will require several impacts to be destroyed. Rather than just popping out of existence, an appropriate destruction animation should be shown when an enemy is destroyed. The player receives points to their score when they destroy enemies, rather than just allowing them to exit the screen.

When an asteroid comes in contact with the player's craft, it takes damage. After a sufficient number of hits have been tallied, the player's craft is destroyed. A "Game Over" message is displayed to the player, and a tap on the screen takes them back to the menu where they may begin another game. If the user has enough points to have an entry in the High Scores menu, they should be taken to that menu instead, and given the option to enter their name, which is displayed next to their score.

Components

- **Main Menu:** This screen greets the user when they first open the application or return to the app after leaving it. The game is paused and saved when the user leaves, allowing them to return. The menu offers "Start Game" or "Resume Game" options as appropriate, as well as a "High Scores" option. This menu can be rendered in OpenGL or can use regular UIKit views in its composition. This screen should also feature a splash graphic, setting the theme for the application. It should combine harmoniously with the other graphics you use elsewhere in the application. You may create this graphic yourself or find an appropriate graphic online. Feel free to use copyrighted material in your submission, but do not submit copyrighted graphics to the App Store. Any graphics you use should be attributed in an Attributions.txt file somewhere in your submission zip containing a link to where you found the graphics.
- **High Scores Menu:** This screen contains a stylized list of the highest 5 (or more) scores achieved by players on this iOS device. It is accessible from the main menu. Also, when a game ends, the user should be taken to this screen after they are shown a "Game Over" message if they scored one of the high scores in that game. They should also be given the opportunity to enter their name in a popover UI element. Then the list should be modified to

contain their new high score. This menu can be rendered in OpenGL or can use regular UIKit views in its composition.

- **Game Screen:** This portion of the game will use the Sprite class and accompanying GLKViewController we wrote in class to form the game view. Sprites representing the player, enemies, and projectiles will be rendered using the Sprite class. Sprites can also be used to render the remaining damage the user's craft has, the player's score, and the "Game Over" message. If the user leaves the application, the current state of the game should be saved. When the user returns to the app, they should be taken to the main menu with the option to resume the game.
  - **Environment**: The player's craft is flying over a region of terrestrial ground, space, clouds, electric ether, or some other appropriate environment. That environment scrolls down the screen at a constant rate, giving the impression that the user's craft is flying forward.
  - **Enemies**: Other enemies will fly over the environment in linear and non-linear paths. They should be able to follow loops, wavy patterns, or change direction abruptly to make the game more difficult. They can even attempt to fly in the user's craft's direction. Some enemies should fire projectiles at the user's craft. These projectiles cannot be destroyed by the user and must be avoided. If you'd like enemies to be attached to the environment (e.g. flak turrets on the ground), setup the movement of that enemy such that it moves at the same rate as the scrolling background so it looks like it is attached there. You may use any mechanism you'd like to script these movements. An ordered list of points that the enemy follows with offsets from the enemy's appearance time is probably the most straightforward.
  - **Levels**: You should script which enemies appear and when. This information constitutes a "level" in the game. The background environment should be different in each level, giving an easy way for the user to tell that they have advanced levels. Once the user has survived all of the scripted enemies for a level, they should be taken to the next level. Use some animation to indicate that is has occurred (onscreen message, fading out and in, gradual change in background, etc). The game should contain at least 3 such levels.
  - **Control:** The player's craft is controlled using tappable regions on-screen. The touches in these regions can be captured by defining touchesBegan, touchesMoved, and touchesEnded methods on the GLKViewController, which receives the touches if the GLKView does not. The touch locations will need to be converted from the Points coordinate space to the coordinate space you use for your game. Once you've done this, you can put button-shaped translucent sprites in the game screen to show where these regions are. When the user is touching the regions these sprites cover, change the appearance of the sprites in some way so the user knows the action is being activated. The controls need are: strafe left, strafe right, accelerate forward, decelerate backward, and fire. Tapping or holding fire fires a projectile from the user's weapon, continuing to fire at a pre-determined rate if held down. Strafe left and right cause the user's craft to move to the left or right in the x direction at a pre-determined rate, and continue moving at that rate until the user stops pressing in that region. Accelerate and decelerate work in the same way, moving in the y direction. Since the background is moving, in the y-direction, having movement controls that work the same as strafe give the impression that the user's craft is speeding up or slowing down, then resuming its previous constant speed, even though it is in fact not moving at all if no controls are pressed.
  - **Animations:** The sprites composing the game screen should animate in some way, in addition to moving to show their positions. These animations can be achieved in one of two ways: using sprite sheets that contain different frames for animations used in the screen, or using rotation and scale manipulations on the sprites. The methods to achieve

these effects were described during lectures 22 to 24. In either case, animations for at least the following events should be provided:

  - The user changing their craft's direction should show on the craft sprite, rotating slightly along the craft's y-axis to show a strafe, and rotating along the x-axis to show acceleration or deceleration. Changes in images can be used instead, showing a small rotation for a strafe and engine glow, fire, or other effect for accelerate / decelerate.
  - Enemies should animate in some way as well, rotating in the direction they are flying, have portions of their craft glow, exhibit motion from pilots, or other appropriate animation according to your application's theme.
  - The user's craft being destroyed should show an explosion, debris floating, or other appropriate effect showing the craft was destroyed before the game over message is presented. It should also have an animation when it is struck but not destroyed, allowing the user to see that this has occurred.

- **Game Loop:** You may use any architecture to organize your application, such as Model View Adapter or view-aware model objects. Whatever organization you choose will need to include a game loop that is executed at least once per drawn frame. Calculate the amount of time that has elapsed since the last execution (using NSDate), then use that elapsed time to calculate how far objects should move based on their movement path. Then check for collisions between the enemies and the user's craft, and collisions between the enemy bullets and the user's craft, and collisions between the user's bullets and enemies. You can do oriented rectangle-rectangle collisions you prefer a more accurate collision model, but circle-circle collisions are easier. Two objects collide if the distance between their center points is less than the sum of their radii. Other events should also be checked for, like when enemies should be created, when enemies fire their weapons, and when a level has been completed.
- **Display Loop:** A loop separate from the game loop should be executed between game loop executions as often as computation time permits, up to every 1/60th of a second. This separation can take on two threads of execution and locking mechanisms (preferred), or can be triggered by two NSTimer objects scheduled to generate events. Be aware if you use timers that if you schedule execution every 1/60th of a second, the timer will not necessarily execute every 1/60th of a second. You should still use NSDate to determine how much time has passed between game loop updates. The display loop should use the current state of the game between game loop executions to draw every element that is currently on-screen, the background, the user's damage amount remaining, and current score. Use an image containing number digits and several sprites to display the score.

Handin

You should hand in your final project package as a zip archive **INCLUDING ANY IMAGE ASSETS YOU USE** to the CADE Lab handin system on the command line:

handin cs4962 projectF your_project_zip_file.zip