Jarren Long

CIS 256 7:30

11/29/2011

Final Project

# Abstract

The purpose of this assignment was to create a horse racing application that would display three animated horses racing across the screen towards the finish line. I find horse races to be “mind-numbingly” (as I say) boring, so I spiced it up a bit. This application was themed using characters from the *Sonic The Hedgehog* series by Sega. While the underlying code follows the same structure as a horse race, the characters, music, and overall look-and-feel were taken straight out of a Sega Genesis (Dear Sony, please don’t sue, it was for educational purposes, I swear on your bottom line) to bring back the button-mashing days of old.

Note that additional features were added, including a main menu, three different game modes (AutoPilot, where all three players are controlled by the computer, and 1-2 Player modes, which allow players to “mash” the ‘A’ and ‘L’ keys to move their racer across the track), custom controls and business objects, and a basic gambling system, which allows the player(s) to bet on each racer in increments of 10 dollars. The winner of the race will be paid out based on the supplied formula of (1+(highest player odds – winner’s odds)) \* winner’s bet.

While there is plenty of room for improvement, this is a complete game demonstrating all aspects of application development using the C# language with the .NET libraries.

# Playing Sonic Racing

The controls for this game were designed to be as simple as possible, while bringing back some of the “old school” ways of the late 20th century. Use the Up/Down arrows to navigate through the main menu options, and the [Enter] key to select. When the user clicks the Start button on the main game form, keys [A] and [L] are used for controlling players 1 and 2 respectively (if running in 1 or 2-Player modes). These need to be pressed AS FAST AS POSSIBLE to make your player move faster, so mash those button! When the race finishes, the user can click the Reset button to move the players back to the starting area, or can press the [F1] key to return to the main menu to change the play mode, view the about box, or quit the application.

# Pseudocode Breakdown

## Application Entry Point

* Start app, launch splash screen
* When splash screen closes, launch game form

## Splash Screen

* Create window with transparent background
* Use timer to control window redraws
* Every timer tick, draw the splash image with a different alpha value
* Fade image into view
* Play "Sega" sound
* Fade image out if view
* When image is invisible, close splash screen

## Game Form

* On first run, configure the players using default values and launch the main menu window
* When main menu closes, get selected option id
* If selected option was Quit, close the application
* If AutoPilot was selected, configure all players to run automatically
* If Player 1 was selected, configure player 1 to use key A
* If Player 2 was selected, configure player 1 to use key A and player 2 to use key L
* Configure player 3 to run automatically
* Configure all player's OnFinish() callback function
* Assign pall 3 player's data objects to the player's respective race control
* Create a hidden button with a Key Up event handler and make button always have focus
* Show the window
* If the player clicks the Place Bet button, display the Betting Window
* --On successful return: Grab the wagers from the betting form and configure each player's bet data in their objects
* If the player clicks the Reset button, reset each player's racing control to the starting position
* If the player clicks the Go button, play the starting sound and Start() each player's racing control, then mark the game as running
* If a key is pressed, and it matches one of the player's assigned keys, move the player forward
* If F1 is pressed, hide the game window and show the main menu
* When the OnFinish() is called, figure out if the player was the first player to finish
* -If they were, play the "winning" sound and calculate their winnings based on all player's odds
* -If they were not, subtract their bet from their money
* If the player went broke, display a message box telling them they are out of the race
* Add the player's win/loss record and track time to their record list
* If all three players have finished the race, mark the game as not running

## Main Menu

* Draw background image and option labels
* Listen for key down events
* When arrow up/down keys are pressed, highlight the next option label
* When enter key is pressed, play the "select" sound
* Mark which option was selected using an ID number
* If Quit is selected, close the application
* If About is selected, launch the about box
* If Autopilot, player 1, or player 2 is selected, launch the Game Form
* Close the menu form

## Betting Form

* When opened, grab the player data from the game form race controls
* Set the values for the name, odds, and wager objects
* -User inputs values
* If OK is clicked, make sure all wagers are a multiple of 10
* If they are, close the window with a success return value
* If they are not, tell the user to fix the values
* If cancel is clicked, close the window with a cancel return value

## About Form

* Display application name, version, and description information
* When OK is clicked, close the window

## Race Control

* When created, start the animation timer
* Display the control with a picture on the left side
* Every time the animation timer ticks, get the next frame to display from the player's data object
* When the control is Start()ed:
* Mark the control's starting time
* Set the initial image to display
* If the control is set to autorun, start the move timer
* Every time the move timer ticks, move the image to the right
* If the image reaches the right side of the control, Stop() the control
* If the control is Reset(), move the picture to the far left of the control, reset the animation frame, and update the picture's tooltip
* If the control is Stop()ped, stop the move timer (if running as autopilot) and call the OnFinish() callback (if assigned)
* If OnFinish() returns true, show the "winning" animation sequence, else show the "losing" sequence

## Race Object (Data Structure)

* This object will hold all of the player metrics, including their name, how much money they have, their bet, their track record, and the sprite sheet they will use.
* If a sprite frame is requested, calculate which image block to return from the sprite map based on the current x and y offsets (assuming 128x128 sprites)

# Application Task, Object, Event (TOE) Chart

This is a breakdown of the complex objects used throughout the application. Primitive data types and temporary variables were left out for simplicity, as was the breakdown of the RaceObject, which is just a large class used for data storage (performs very limited computation).

|  |  |  |
| --- | --- | --- |
| Task | Object | Event |
| **SplashForm** |  |  |
| Animate (fade) splash image | m\_fadeTimer | fadeTimer\_Tick() |
| **BettingForm** |  |  |
| Show player 1 name | m\_player1NameLabel |  |
| Show player 1 odds | m\_player1OddsLabel |  |
| Show player 1 wager | m\_player1WagerTextBox |  |
| Show player 2 name | m\_player2NameLabel |  |
| Show player 2 odds | m\_player2OddsLabel |  |
| Show player 2 wager | m\_player2WagerTextBox |  |
| Show player 3 name | m\_player3NameLabel |  |
| Show player 3 odds | m\_player3OddsLabel |  |
| Show player 3 wager | m\_player3WagerTextBox |  |
| Check wager values | m\_okButton | okButton\_Click() |
| Close betting form | m\_cancelButton | cancelButton\_Click() |
| **MenuForm** |  |  |
| Cycle through the menu options using the arrow keys, [Enter] selects option |  | MenuForm\_KeyDown() |
| Show the AutoPilot option | m\_autoLabel |  |
| Show the 1 Player option | m\_p1Label |  |
| Show the 2 Player option | m\_p2Label |  |
| Show the about box (.NET builtin form, no modifications made) | m\_aboutLabel | (About box launched from KeyDown event handler) |
| Quit the application | m\_quitLabel | (App closed from GameForm) |
| **GameForm** |  |  |
| Used for playing game sounds | m\_soundPlayer |  |
| Used for displaying the main menu | m\_menu |  |
| Used for the creation of the initial RaceObjects passed to the RaceControls, disposed of after the GameForm starts the first time | m\_p1,m\_p2,m\_p3 |  |
| Hide the game form if it is displayed before the main menu (fixes a timing issue on newer computers) |  | RaceForm\_Shown() |
| Display player 1’s race status | m\_p1RaceControl |  |
| Display player 2’s race status | m\_p2RaceControl |  |
| Display player 3’s race status | m\_p3RaceControl |  |
| Start the race controls | m\_goButton | goButton\_Click() |
| Reset the race controls | m\_resetButton | resetButton\_Click() |
| Place bets on each player | m\_betButton | betButton\_Click() |
| Distribute key up events to their respective players based on the player’s assigned move key | m\_keyForwardButton | keyForwardButton\_KeyUp() |
| If the button loses focus, we will reset it’s focus (so it ALWAYS has focus) |  | keyForwardButton\_Leave() |
| **RaceControl** |  |  |
| Show the current frame for the player | m\_pictureBox |  |
| Control how fast the player can move across the screen | m\_moveTimer | timer1\_Tick() |
| Control the animation speed of the player | m\_animationTimer | animationTimer\_Tick() |
| Show a tooltip on the player’s picture with their name, odds, and money | m\_toolTip |  |
| Track what time the player started racing at for the current race | m\_startTime |  |
| Generates a random number to figure out how far to move the player | m\_random |  |
| Holds all of the player’s data | m\_player |  |
| **RaceObject** | (This is a LARGE data structure) | (No event handlers) |
| Retrieve the next frame in the animation sequence |  | Frame (Property) |

# Final Project Grading Sheet

When you submit your work, include the following form pasted into the message window of Angel. In the left most column, identify the feature you coded. I will not give points if you fail to identify your work or that work is not in your TOE/pseudocode. If you add a custom feature I pre-approved, include a paragraph below the form to describe what I should look for.

**Student Name: Jarren Long**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Description | Points | Awarded/Lost |
|  | Basic Features | **80** |  |
|  | TOE/Pseudocode |  |  |
|  | Basic Look-n-Feel |  |  |
|  | Basic Operation |  |  |
|  |  |  |  |
|  | Optional Features | **Up to 40** |  |
| Yes | Added look-n-feel (#1) | **10** |  |
| Yes | Add Audio (#2) | **10** |  |
| Yes | Gambling Feature (#3) | **10** |  |
| Yes | Additional gambling | **10** |  |
| No ☹ | Printed output | **10** |  |
| Yes | Business Object | **10** |  |
| Yes | Windows Control | **15** |  |
| Yes | Pre-approved feature | **1-10** |  |

# Preapproved Feature Descriptions

Rather than using animated GIFs, I created three sprite sheets that include all of the frames to be animated. These frames are extracted from the maps on-the-fly, which are then drawn back to back to create the animated pictures. This shaved over 1Mb off the final size of the project, and allowed me to animate all of the images manually, which I can control via code.

I also added a main menu which can be used to select three different game modes, “AutoPilot,” “1 Player,” and “2 Player”. AutoPilot mode allows all three players to be controlled by the PC, while 1 and 2 player modes allow the user(s) to control their characters using the ‘A’ and ‘L’ keys, respectively. This makes for a more interactive application that can be played by two people. The main menu also has options to display the about box and to quit the application.

The GameForm and RaceControl both use multiple advanced coding techniques that were not covered in our class. For example, the RaceControl is passed a deep copy (full copy, not just a reference or shallow copy of just the primitive types) of a RaceObject when the game is first started, and then internally manages the state of the object. After receiving the object, the temporary object is disposed of, so it no longer takes up memory in the main game form (as far as the user is concerned). Another technique used between the GameForm and the RaceControls is the use of a callback delegate, which is fired when the RaceControls detect that their player has crossed the finish line. The callback takes two parameters, a reference to the RaceControl’s internal RaceObject, and a TimeSpan that represents the amount of time it took that player to move from the start to the finish line. The RaceObject reference allows the delegate to manipulate fields inside of the RaceObject without having to worry about passing the entire object as a parameter (just a reference, AKA a pointer to the object), which speeds execution.

Another notable feature is the use of a hidden button on the GameForm. This button is designed to always have focus (even if we tab away from it), and accept KeyUp events. When an event is received, the handler looks to see if it matches one of the keys assigned to the RaceControls, and if so, calls the UpdatePosition() method associated with that control. This allows us to watch for multiple key events and control multiple RaceControls, without having to worry about which control has the user’s focus. The handler also listens for the [F1] key, which will hide the main game form and launch the main menu when the race is not running.

Finally, a setup project was added to the solution to allow for the installation of the application to the local hard drive. The setup project also adds shortcut icons to the user’s desktop and start menu that point to the application after it is installed.

I designed this code with the intent of adding a few additional features (time permitting, it was not), mainly printing out player metrics, such as wins/losses and track times, so there is code in there that is currently unused (see the internal TrackRecord class and associated list). These extra bits can be safely ignored.