Design Overview for Escapetown

Name: Sanya Baweja
Student ID: 104215361

Summary of Program

My idea for the custom design project was a simple game using ruby and gosu libraries. In this game, we control a character with the arrow keys to dodge falling obstacles and gather points. The objective is to survive for as long as possible without hitting any obstacles. The game features a simple graphical interface created using the Gosu library in Ruby. Points are earned by collecting them while avoiding obstacles, and the game concludes when the player collides with an obstacle. The final score is shown at the end of the game.

Required Data Types

Describe each of the records and enumerations you will create using the following table (one per record).

Table 1: <<Player>> details

Field Name	Type	Notes
x: Represents the player's horizontal position.	Integer or float	Represents the player character and stores its position (x).
y: Represents the player's vertical position.	Integer or float	Represents the player character and stores its position(y).
width: Represents the width of the player.	Integer or float	Represents the player character and stores its width.
height: Represents the height of the player.	Integer or float	Represents the player character and stores its height.

Table 2: <<Obstacle>> details

Field Name	Type	Notes
x: Represents the player's horizontal position.	Integer or float	Represents the player character and stores its position (x).
y: Represents the player's vertical position.	Integer or float	Represents the player character and stores its position(y).

width: Represents the width of the player.	Integer or float	Represents the player character and stores its width.
height: Represents the height of the player.	Integer or float	Represents the player character and stores its height.

Table 2: <<Keys>> details

Value	Notes
Left	Represents the value of the left arrow key and is assigned the constant value Gosu::KB_LEFT.
Right	Represents the value of the right arrow key and is assigned the constant value Gosu::KB_RIGHT.
Escape	Represents the value of the Escape key and is assigned the constant value Gosu::KB_ESCAPE.

Overview of Program Structure

I first made sure to download and install Gosu on my computer before using the require 'gosu' command to import the necessary Gosu library. As a result, I had access to all the essential equipment.
I then went on to create a few constants that would specify the size of the game window and the player's movement speed. I created the ideal environment for a satisfying gaming experience by doing this.
I created a useful module named "Keys" to make managing keyboard input simpler.
This gave keyboard keys unique codes, acting as though they were speaking a language. It made it simple for me to determine whether the player used the left, right, or escape keys.
To describe the player character, I created a record 'Struct'—or, as you may put it, a blueprint. Important player details like height and position were kept in this database. It was similar to creating the player's foundation within the game.
Then, I shall define an Obstacle record using Struct. It has fields (x, y, width, height, speed) to store the obstacle's position, size, and speed.
I shall use the updated Keys module to check if the left or right arrow keys are being pressed, and then call the corresponding movement methods of the player.
Then I'll define a method in charge of processing button presses. The pressed button will close the game window if its ID matches the Keys::ESCAPE constant (Escape key). I made sure that if the player decided to press the escape key, the game window would gracefully close, providing an escape from the virtual world back to reality

I'll create a class called Player to stand in for the player character in the game. This class served as a container for the player's attributes and behavior. It was like giving the player a unique identity, complete with their own movements and actions.
Similarly, I shall define a class Obstacle to represent the falling obstacles in the game. This class encapsulates the obstacle's position, size, and behavior.
Finally, I'll construct a GameWindow class object and call the show method to launch the game. This technique launches the game loop, which renders and updates the game until the window is closed. Using the newly produced records, the Player and Obstacle objects are instantiated, and the game loop and rendering functionality are kept.
This gives the player, obstacle, and keyboard keys a clearer and more organised representation in the game.