SIT771 Object Oriented Development

Credit Task 3.3: Moving the Player

Overview

This is the second of a series of tasks in which you will develop a small program. These tasks are designed to help you explore the concepts being covered, and to practice your programming skills.

The material in Course 2, Week 1 will help you with this task.

In this task you will extend the Player class to allow the user to move the player around on the screen.

Submission Details

Submit the following files to OnTrack.

- The Program and Player code (*Program.cs* and *Player.cs*)
- A screenshot of your program running

You want to focus on the use control flow, as well as reinforcing your understanding of classes, methods, and objects.

Instructions

Get started by opening up the project and getting everything ready to start working on making the required additions.

1. Return to your **Robot Dodge** game project. Open it in Visual Studio Code, and have a Terminal open with that folder as the current working directory.

2. Open your **Program.cs** and **Player.cs** files.

Adding a Main Loop

The first change that we need to make is to add a loop to process events in `Main`.

Use the code from the videos and the code analysis task as examples. You want the game to loop while the window has not been closed. Within the loop you will need to:

1. Tell `SplashKit` to `ProcessEvents`
2. Clear the game window
3. Ask the Player to draw themselves
4. Refresh the screen

Test this by making sure that the window stays around until you close it.

Asking the Player to Handle Input

Now that we have a basic event loop in place, we can start to get the Player to respond to events that happen. To do this we need to revisit our design and add some responsibilities (methods and properties) to the `Player` class. These can then be used by the program. The new design is shown below.

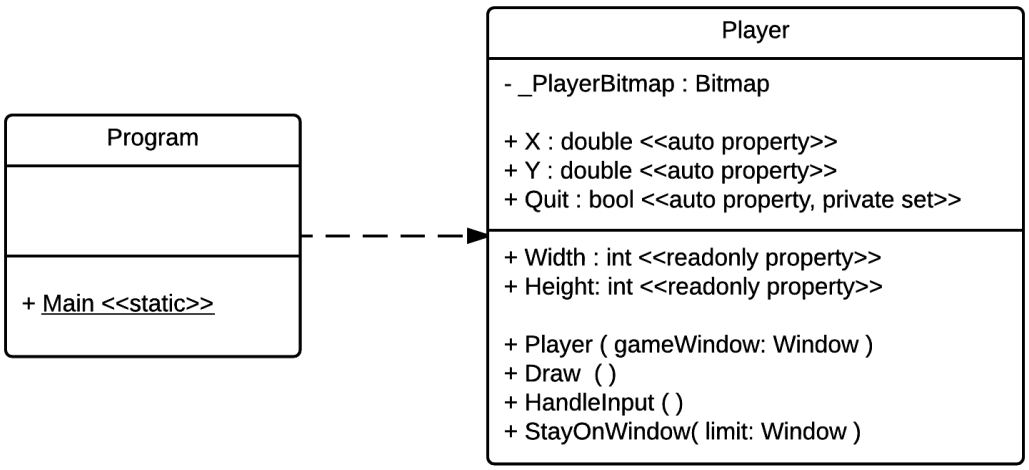

Figure: Design for iteration 2 of Robot Dodge

Changes include the following additions:

- A **Quit** property (use an auto property, with a private set)

  This will indicate if the player wants to quit.

  Ensure that it is set to `false` when the player is created.
- The **HandleInput** method

  This will check if any of the arrow keys are held down, and will move the player by changing its X and Y location in response to these events. Introduce a SPEED constant, set to 5, and add/subtract this to/from X and Y when the keys are held down.

  It will also check if the Escape key was typed, and if it was typed will set Quit to true.
- A **StayOnWindow** method

  We want to keep the Player on the screen. This method will be passed a Window and the Player will need to make sure that it remains within the Window bounds (plus a small gap of 10 pixels).

  In this method declare a constant GAP to help with this:

  ```
  const int GAP = 10;
  ```

  Then you can use this to check if the player is out of bounds. For example, if X is less than GAP, then change X to be equal to GAP. Similarly, you need to check if the rigth side of the player is larger than the Width of the Window minus GAP, and if it is move the player so that their right side is at this position.

This will require some thinking, remember to take into consideration the width of the screen, the player's width, and the player location. Similar logic can then be used for the Y and the height of the Window.

Inside your `Program` you can then make use of these in `Main`.

- Have the game end when the player's `Quit` property is true, or the window is closed.

  Using a while here means you need to express this as what it needs to be to run, not to quit. So it will be something like "while the player has not quit, and they have not closed the window".

- After processing events, ask the Player to `HandleInput` then tell them to `StayOnWindow` and pass in the game window.

Code up these changes, and test that your program works. I would suggest that you implement this one piece at a time. Get something like the new Quit property working, then run the program, make sure it works, correct any issues, then get the player moving, then make them stay on the window. Trying to get this all working at once is likely to make your life really difficult, the more code you write before testing the more issues you will need to deal with.

Once you are done, backup your work, take a screenshot and submit this to OnTrack for feedback. Feel free to play around with additional extensions if you want, like adding a boost or having the player wrap around the screen rather than being limited to stay within the screen bounds.

Mostly, we will want to check your use of control flow, appropriate indentation, and demonstrations of the use of different logical expressions. Feel free to play around a little with this game as you go.