2D games using Silverlight - Collision detection implementation

This article shows how to implement collision detection in a Microsoft Silverlight game application. This is the second article in a series which will show how to create a complete working game (a clone of the classic Arkanoid game).

Introduction

The first article in this series 2D games using Silverlight - implementing the game loop introduced the possibility of writing 2D games using Silverlight, an approach that has benefits when targeting all of the platforms: Windows Phone 7, Windows Phone 8 and Windows 8 and Internet browsers. The original article covered the creation of the game skeleton for a simple Arkanoid clone called *JailBreaker*, and included the games loop and the game controllable loop.

JailBreaker (Arkanoid clone) running on a PC

This article extends the game skeleton by adding collision detection.

Collision detection implementation strategy

There are numerous strategies and algorithms for collision detection: broad detection phase, narrow detection phase, how to use dot production to estimate distance, etc. However it does not make sense to re-invent the wheel: for more complicated games there are open source physics engines, and for simple games like this there are often platform APIs that can be used to help with collision detection. For this example we use the collision detection functionality built into the platform, along with a good understanding of the simple collision detection required for this game.

Considering the game environment:

- There are few components with complex non-predictable movement - essentially just the club and ball. All other elements either stand still or fall in predictable manner - they do not collide. As a result we only need to consider a single object, the ball, for determining whether a collision has occurred.
- We only need to consider collisions in three directions along the active object movement: forward, forward diagonal up, forward diagonal down. This limitation allows us to minimize number of tests required

Collision detection in the platform:

- There is a helper class VisualTreeHelper in the platform that allows hit test on the component tree (all visible components in Silverlight application form a component tree hierarchy)
- There is a support for object animation based on StoryBoard that is suitable for steady moving objects
- the game movement is based on screen refresh rate, which is not constant across devices. To make object speed the same on different devices/conditions we need to measure the refresh rate before calculation the object next position
- During object movement, the object perception will be conducted on the next the object position because we are going to make the calculations in a hook to the screen refresh loop.

New module in the project PhysicalBody

The new class PhysicalBody(PhysicalBody.cs, inside Jailbreaker.zip) is attached to a UI element statically created in XAML. It handles the element:

1. position update
2. object collision assessment
3. checking the class container extends to keep the element position inside the container

(C) Copyright Nokia 2014. All rights reserved.

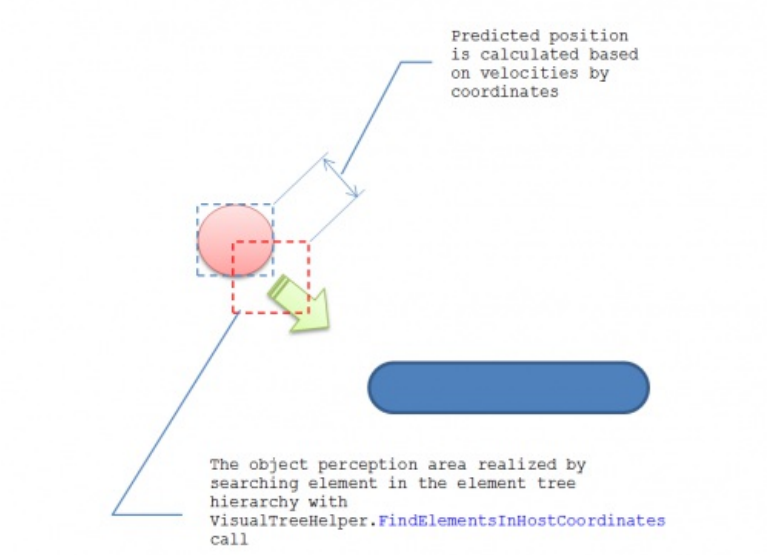
Also it accepts `gameOver` and `hitScores()` method delegates from the class container

Position update

The position update is a simple increment of the current position based on the current body speed. Note that the body speed defines the body direction according to the coordinate system: positive value is from left to right by X axis, and from top to bottom in Y axis and negative value is in the opposite direction.

Object collision assessment

This is the crucial part of the article. The collision detection is implemented in one function and follows the strategy we have declared above in the preceding sections.



Let us examine the code:

```
private void assessCollision()
        {
            // get bounding box and offset the box in movement direction on one step
            Canvas canvas = (Canvas)body.Parent;
 // (1) for converting screen-to-control coordinates
            var transform = canvas.TransformToVisual(Application.Current.RootVisual); //
for converting screen-to-control coordinates

            var origin = transform.Transform(new Point(Canvas.GetLeft(body) + velocityX,
Canvas.GetTop(body) + velocityY));

            var boundingBox = new Rect(origin, new Size(body.ActualWidth,
body.ActualHeight));

// (2) check whether there are element exist on the body's way
//       in three directions : forward, forward-diagonal-up and forward-diagonal-down

// (3) use VisualTreeHelper helper class to walk through elements hierarchy tree

// (4) filter out all objects except Shapes

            if (isCollided(VisualTreeHelper.FindElementsInHostCoordinates(new
Point(boundingBox.Right, boundingBox.Bottom), (Canvas)body.Parent)) ||
                        isCollided(VisualTreeHelper.FindElementsInHostCoordinates(new
Point(boundingBox.Right, boundingBox.Y), (Canvas)body.Parent)) ||

isCollided(VisualTreeHelper.FindElementsInHostCoordinates(new Point(boundingBox.X,
boundingBox.Bottom), (Canvas)body.Parent)))
```


(C) Copyright Nokia 2014. All rights reserved.

```
            {
                bounceBackY();
            }
        }
```

for more details please check "PhysicalBody.cs" inside Jailbreaker.zip

Summary

Given approach shows how easy to implement collision detection in Silverlight game project if use only the platform means. Working example is available in source code that can be build and run on Windows Phone 7.X device. Also since Windows platform does not allow side-loading installation binaries, just for reference the same code has been deployed to a web site and can run embedded into a browser.

(C) Copyright Nokia 2014. All rights reserved.