IBoundary.cs

Kod jest bardziej czytelny, jeżeli klasy są w osobnych plikach.

DrawGameObjects.DrawBall()

Nie potrzeba takich metod jak GetPosition(out int posX, out int posY) (i innych podobnych).

posX i posY można wyciągnąć bezpośrednio z obiektu ball (ball.Boundary.Min.X). Zbędne metody można usunąć i kod jest prostszy/czytelniejszy.

private double ChangeDegreeToRadians(double angle)

{

return Math.PI \* angle / 180.0;

}

Optymalizacja. Z wartości PI / 180.0 można zrobić const. Zaoszczędzisz sobie wykonywania za każdym razem tego samego dzielenia.

Logger.Instance.Write(

string.Format(

"Ball {0}: {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}",

reverse ? "reverse" : "",

Boundary.Min.X,

Boundary.Min.Y,

Boundary.Size.X,

Boundary.Size.Y,

OffsetX,

OffsetY,

DirectionX,

DirectionY,

Iteration,

Degree));

}

Do Vector2 możesz dodać override ToString który zwróci ładnie sformatowane wartości np.   
$”X: {X}; Y:{Y}”  
Wtedy w takich miejscach jak ten string.Format i innych gdzie logujesz wartości wektora, zamiast przekazywać osobno parametry Boundary.Min.X i Boundary.Min.Y możesz przekazać po prostu jeden parametr Boundary.Min i mieć formatowanie w jednym miejscu na wypadek gdybyś chciał je zmienić później.

public interface IBoundary

{

Rectangle Boundary { get; set; }

}

Nic w kodzie nie ustawia Boundary z zewnątrz, więc set nie jest potrzebny.

Usunąłeś użycia BorderEnum, LevelEnum i PositionEnum – te klasy można teraz usunąć.

BallEnum z jakiegoś powodu jest ciągle użyte w BallManager?

RandomGenerator.GenerateDegree()

if (type == DegreeType.Corner)

{

Degree += degreeMargin;

}

else if (type == DegreeType.Average)

{

Degree += degreeRangeMax;

}

else if (type == DegreeType.Centre)

{

Degree += degreeRangeMax \* 2;

Degree -= degreeMargin;

}

else

{

throw new ArgumentOutOfRangeException();

}

Do takiego sprawdzania DegreeType lepszy byłby switch (z ArgumentOutOfRangeException w sekcji default)

Jeżeli używasz ArgumentOutOfRangeException to zawsze lepiej wykorzystać konstruktor typu   
ArgumentOutOfRangeException(string paramName, object actualValue, string message);

Dzięki temu jak wyleci wyjątek od razu widać jaka wartość spowodowała problem.

Testy NUnit. Musiałem dodać NUnit Test Adapter z nugeta żeby VS wykrył testy.

Te testy są trochę mało czytelne.

Np. VerifyCalculateNewDegree… sprawdzają tylko czy metoda została wywołana. Nie jestem pewien czy jest z takiego testu jakaś wartość.

VerifyCalculateNewDegreeWithCentreType sprawdza, że metoda nie została wywołana, ale ani jego treść ani jego nazwa nie tłumaczy, że to dlatego że Centre to domyślna wartość.

VerifyBallMovement ma 10 wywołań tej samej funkcji co mogłoby być w pętli. Asserty sprawdzają, że pozycja jest 7, ale też nie wiadomo z czego to wynika.