```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

using Microsoft.Modeling;

namespace YahtzeeModel
{
    public enum ScoreType
    {
        One, Two, Three, Four, Five, Six,
        ThreeOfAKind, FourOfAKind, FullHouse, SmallStraight, LargeStraight, Chance,
        Yahtzee
    }

    /// <summary>
    /// The Yahtzee Rules Model
    /// </summary>
    static class YahtzeeRules
    {
        static int numRounds;

        static int numRolls;

        static int numHeld;//how many to hold
        static bool d1Held;
        static bool d2Held;
        static bool d3Held;
        static bool d4Held;
        static bool d5Held;

        static int d1Val; //values of dice
        static int d2Val;
        static int d3Val;
        static int d4Val;
        static int d5Val;

        static int upperScore;
        static bool upperBonusAwarded;
        static int lowerScore;

        static bool onesPlayed; //keep track of what has been played
        static bool twosPlayed;
        static bool threesPlayed;
        static bool foursPlayed;
        static bool fivesPlayed;
        static bool sixesPlayed;
        static bool threeOfAKindPlayed;
        static bool fourOfAKindPlayed;
        static bool fullHousePlayed;
        static bool smallStraightPlayed;
        static bool largeStraightPlayed;
        static bool chancePlayed;
        static bool yahtzeePlayed;

        [Rule]
        static void NewGame()
        {
            numRounds = 0;  // max 13

            numRolls = 0;   // max 3 per round

            numHeld = 0;    // max 4 per round, no unholding allowed
            d1Held = false;
            d2Held = false;
            d3Held = false;
            d4Held = false;
            d5Held = false;

            upperScore = 0;
            upperBonusAwarded = false;
            lowerScore = 0;

            onesPlayed = false;
            twosPlayed = false;
            threesPlayed = false;
            foursPlayed = false;
            fivesPlayed = false;
            sixesPlayed = false;
            threeOfAKindPlayed = false;
            fourOfAKindPlayed = false;
            fullHousePlayed = false;
            smallStraightPlayed = false;
            largeStraightPlayed = false;
            chancePlayed = false;
            yahtzeePlayed = false;
        }


        [Rule]
        static void RollAll(int d1, int d2, int d3, int d4, int d5)
        {
            Condition.IsTrue(numRolls < 3);
            Condition.IsTrue(numRounds < 13);

            if (numRolls == 0)
            {
                Condition.IsTrue(numHeld == 0);
            }
            else
            {
                Condition.IsTrue(!d1Held || d1 == d1Val);
                Condition.IsTrue(!d2Held || d2 == d2Val);
                Condition.IsTrue(!d3Held || d3 == d3Val);
                Condition.IsTrue(!d4Held || d4 == d4Val);
                Condition.IsTrue(!d5Held || d5 == d5Val);
            }

            /* store values from this roll */
            d1Val = d1;
            d2Val = d2;
            d3Val = d3;
            d4Val = d4;
            d5Val = d5;

            numRolls += 1;
        }

        [Rule]
        static void hold(int dnumber)
        {
            Condition.IsTrue(numHeld < 5);

            if (dnumber == 1)
            {
                Condition.IsTrue(d1Held == false);
                d1Held = true;
            }
            else if (dnumber == 2)
            {
                Condition.IsTrue(d2Held == false);
                d2Held = true;
            }
            else if (dnumber == 3)
            {
                Condition.IsTrue(d3Held == false);
                d3Held = true;
            }
            else if (dnumber == 4)
            {
                Condition.IsTrue(d4Held == false);
                d4Held = true;
            }
            else if (dnumber == 5)
            {
                Condition.IsTrue(d5Held == false);
                d5Held = true;
            }
            else
            {
                Condition.Fail();
            }

        }

        [Rule]
        static int Score(ScoreType scorePosition)
        {
            int upperScoreThisRound = 0;
            int lowerScoreThisRound = 0;

            /* TODO: check if position already played */
            if (scorePosition == ScoreType.One)
            {
                Condition.IsTrue(onesPlayed == false);
                onesPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Two)
            {
                Condition.IsTrue(twosPlayed == false);
                twosPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Three)
            {
                Condition.IsTrue(threesPlayed == false);
                threesPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Four)
            {
                Condition.IsTrue(foursPlayed == false);
                foursPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Five)
            {
                Condition.IsTrue(fivesPlayed == false);
                fivesPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Six)
            {
                Condition.IsTrue(sixesPlayed == false);
                sixesPlayed = true;
                upperScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.ThreeOfAKind)
            {
                Condition.IsTrue(threeOfAKindPlayed == false);
                threeOfAKindPlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.FourOfAKind)
            {
                Condition.IsTrue(fourOfAKindPlayed == false);
                fourOfAKindPlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.FullHouse)
            {
                Condition.IsTrue(fullHousePlayed == false);
                fullHousePlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.SmallStraight)
            {
                Condition.IsTrue(smallStraightPlayed == false);
                smallStraightPlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.LargeStraight)
            {
                Condition.IsTrue(largeStraightPlayed == false);
                largeStraightPlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Chance)
            {
                Condition.IsTrue(chancePlayed == false);
                chancePlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }
            else if (scorePosition == ScoreType.Yahtzee)
            {
                Condition.IsTrue(yahtzeePlayed == false);
                yahtzeePlayed = true;
                lowerScoreThisRound += calcScore(scorePosition);
            }

            /* update score */
            upperScore += upperScoreThisRound;
            lowerScore += lowerScoreThisRound;

            if (upperScore >= 63 && !upperBonusAwarded)
            {
                upperScore += 35;
                upperBonusAwarded = true;
            }

            /* TODO: validate new score against something? */


            /* reset for next round */
            numRolls = 0;
            numRounds += 1;

            numHeld = 0;
            d1Held = false;
            d2Held = false;
            d3Held = false;
            d4Held = false;
            d5Held = false;

            /* return score */
            return upperScore + lowerScore;
        }

        static int calcScore(ScoreType scorePosition)
        {
            int num1s = countDie(1);
            int num2s = countDie(2);
            int num3s = countDie(3);
            int num4s = countDie(4);
            int num5s = countDie(5);
            int num6s = countDie(6);

            if (scorePosition == ScoreType.One)
            {
                return num1s;
            }
            else if (scorePosition == ScoreType.Two)
            {
                return num2s * 2;
            }
            else if (scorePosition == ScoreType.Three)
            {
                return num3s * 3;
            }
            else if (scorePosition == ScoreType.Four)
            {
                return num4s * 4;
            }
            else if (scorePosition == ScoreType.Five)
            {
                return num5s * 5;
            }
            else if (scorePosition == ScoreType.Six)
            {
                return num6s * 6;
            }
            else if (scorePosition == ScoreType.ThreeOfAKind)
            {
                if (num1s >= 3 || num2s >= 3 || num3s >= 3 || num4s >= 3 || num5s >= 3 || num6s >= 3)
                {
                    return (num1s * 1) + (num2s * 2) + (num3s * 3) + (num4s * 4) + (num5s * 5) + (num6s *  ↙
    6);
                }
            }
            else if (scorePosition == ScoreType.FourOfAKind)
            {
                if (num1s >= 4 || num2s >= 4 || num3s >= 4 || num4s >= 4 || num5s >= 4 || num6s >= 4)
                {
                    return (num1s * 1) + (num2s * 2) + (num3s * 3) + (num4s * 4) + (num5s * 5) + (num6s *  ↙
    6);
                }
            }
            else if (scorePosition == ScoreType.FullHouse)
            {
                if ((num1s == 3 || num2s == 3 || num3s == 3 || num4s == 3 || num5s == 3 || num6s == 3)
                    && (num1s == 2 || num2s == 2 || num3s == 2 || num4s == 2 || num5s == 2 || num6s == 2))
                {
                    return 25;
                }
            }
            else if (scorePosition == ScoreType.SmallStraight)
            {
                if ((num1s >= 1 && num2s >= 1 && num3s >= 1 && num4s >= 1)
                    || (num2s >= 1 && num3s >= 1 && num4s >= 1 && num5s >= 1)
                    || (num3s >= 1 && num4s >= 1 && num5s >= 1 && num6s >= 1))
                {
                    return 30;
                }
            }
            else if (scorePosition == ScoreType.LargeStraight)
            {
                if ((num1s == 1 && num2s == 1 && num3s == 1 && num4s == 1 && num5s == 1)
                    || (num2s == 1 && num3s == 1 && num4s == 1 && num5s == 1 && num6s == 1))
                {
                    return 40;
                }
            }
            else if (scorePosition == ScoreType.Chance)
            {
                return (num1s * 1) + (num2s * 2) + (num3s * 3) + (num4s * 4) + (num5s * 5) + (num6s * 6);
            }
            else if (scorePosition == ScoreType.Yahtzee)
            {
                if (num1s == 5 || num2s == 5 || num3s == 5 || num4s == 5 || num5s == 5 || num6s == 5)
                {
                    return 50;
                }
            }

            /* scratch */
            return 0;
        }

        static int countDie(int faceVal)
        {
            int count = 0;

            if (d1Val == faceVal)
                count++;
            if (d2Val == faceVal)
                count++;
            if (d3Val == faceVal)
                count++;
            if (d4Val == faceVal)
                count++;
            if (d5Val == faceVal)
                count++;

            return count;
        }

    }
}
```