```
// This is a Spec Explorer coordination script (Cord version 1.0).
// Here is where you define configurations and machines describing the
// exploration to be performed.

using YahtzeeModel;


/// Contains actions of the model, bounds, and switches.
config Main
{
    switch StepBound = 1280;
    switch PathDepthBound = 1280;
    switch StateBound = 1280;

    action abstract static void SUT.NewGame();

    action abstract static void SUT.RollAll(int i1, int i2, int i3, int i4, int i5);

    action abstract static void SUT.hold(int dnumber)
    where dnumber in {1..5};

    action abstract static int SUT.Score(ScoreType scorePosition);
}

config RollConstraint : Main
{
    action abstract static void SUT.RollAll(int i1,int i2,int i3,int i4,int i5)
    where{.
        Condition.In(i1,1,2,3,4,5,6);
        Condition.In(i2,1,2,3,4,5,6);
        Condition.In(i3,1,2,3,4,5,6);
        Condition.In(i4,1,2,3,4,5,6);
        Condition.In(i5,1,2,3,4,5,6);
    .};
}

config FixedDice : Main
{
    action abstract static void SUT.RollAll(int i1,int i2,int i3,int i4,int i5)
    where{.
        int[] values = new int[]{1,2,3,4,5,6};
        Condition.IsTrue(i1 == Probability.Choose(values));
        Condition.IsTrue(i2 == Probability.Choose(values));
        Condition.IsTrue(i3 == Probability.Choose(values));
        Condition.IsTrue(i4 == Probability.Choose(values));
        Condition.IsTrue(i5 == Probability.Choose(values));
    .};

}

//This machine is NOT to be explored
machine FixedDice() : FixedDice
{
    construct model program from FixedDice
}

//Roll test: should be ended after 3rd roll with 3 random rolls
machine RollTest() : FixedDice
{
    //(NewGame; RollAll*) || (construct model program from RollConstraint)
    /*
     * This test seems to more accurately show what this test wants, which is that the number of rolls
     * allowed are limited to 3 per round.
     */
    (NewGame; RollAll(1,1,1,1,1)*) || (construct model program from RollConstraint)
}
```

```
//Hold test, should vary only 4th and 5th dice (ie 36 end states)
machine hold1() : RollConstraint
{
    (NewGame; RollAll(1,1,1,1,1); hold(1); hold(2); hold(3); RollAll)
    || (construct model program from RollConstraint)
}

//hold 4 dice and score each of the six possible resulting throws
//(showing all possible scoring for each of these throws
machine hold2() : RollConstraint
{
    (NewGame; RollAll(1,1,1,1,1); hold(1); hold(2); hold(3); hold(4); RollAll; Score(_))
    || (construct model program from RollConstraint)
}

// 36 combinations with 3, 3, 3 (as last dices) to score
// + 1 for 2, 2, 1, 1, 3 which scores 0
machine ScoreThreeOfAKind() : RollConstraint
{
    (NewGame; (RollAll(_, _, _, 3, 3); Score(ScoreType.ThreeOfAKind)
    ))
    || (construct model program from RollConstraint)
}

//Test for two rounds. Checks if the already scored category will incorrectly show
// in next round
machine TwoFixedRounds() : RollConstraint
{
    (NewGame; RollAll(1,2,3,4,5); Score(ScoreType.LargeStraight);
      RollAll(2,3,4,5,6); Score(_)) || (construct model program from RollConstraint)
}

machine junk1() : RollConstraint
{
    (NewGame; RollAll(1,1,1,4,5); Score(ScoreType.ThreeOfAKind);
      RollAll(2,2,2,5,6); Score(_)) || (construct model program from RollConstraint)
}

//Test upper section bonus
machine upperBonus() : RollConstraint
{
    (NewGame;
        RollAll(1, 1, 1, 1, 2); Score(ScoreType.One);
        RollAll(2, 2, 2, 2, 1); Score(ScoreType.Two);
        RollAll(3, 3, 3, 3, 4); Score(ScoreType.Three);
        RollAll(4, 4, 4, 4, 5); Score(ScoreType.Four);
        RollAll(5, 5, 5, 5, 6); Score(ScoreType.Five);
        RollAll(6, 6, 6, 6, 1); Score(ScoreType.Six);
        RollAll(6, 6, 6, 6, 5); Score(ScoreType.ThreeOfAKind);
        RollAll(6, 6, 6, 6, 5); Score(ScoreType.FourOfAKind);
        RollAll(6, 6, 6, 5, 5); Score(ScoreType.FullHouse);
        RollAll(1, 2, 3, 4, 6); Score(ScoreType.SmallStraight);
        RollAll(1, 2, 3, 4, 5); Score(ScoreType.LargeStraight);
        RollAll(6, 6, 6, 6, 5); Score(ScoreType.Chance);
        RollAll(6, 6, 6, 6, 5); Score(ScoreType.Yahtzee))
    || (construct model program from RollConstraint)
}

//Test for highest score (including the optional Yahztzee bonuses)
//IF you have the logic for these bonuses, the expected total is 1375
machine HighestScore() : RollConstraint
{
    (NewGame; RollAll(1, 1, 1, 1, 1); Score(ScoreType.Yahtzee);
        RollAll(1, 1, 1, 1, 1); Score(ScoreType.One);
        RollAll(2, 2, 2, 2, 2); Score(ScoreType.Two);
```

```
        RollAll(3, 3, 3, 3, 3); Score(ScoreType.Three);
        RollAll(4, 4, 4, 4, 4); Score(ScoreType.Four);
        RollAll(5, 5, 5, 5, 5); Score(ScoreType.Five);
        RollAll(6, 6, 6, 6, 6); Score(ScoreType.Six);
        RollAll(6, 6, 6, 6, 6); Score(ScoreType.ThreeOfAKind);
        RollAll(6, 6, 6, 6, 6); Score(ScoreType.FourOfAKind);
        RollAll(5, 5, 6, 6, 6); Score(ScoreType.FullHouse);
        RollAll(1, 2, 3, 4, 6); Score(ScoreType.SmallStraight);
        RollAll(2, 3, 4, 5, 6); Score(ScoreType.LargeStraight);
        RollAll(6, 6, 6, 6, 6); Score(ScoreType.Chance))
    || (construct model program from RollConstraint)
}

machine TestRollTest() : Main where TestEnabled = true
{
    construct test cases for RollTest
}

machine TestHold1() : Main where TestEnabled = true
{
    construct test cases for hold1
}

machine TestHold2() : Main where TestEnabled = true
{
    construct test cases for hold2
}

machine TestScoreThreeOfAKind() : Main where TestEnabled = true
{
    construct test cases for ScoreThreeOfAKind
}

machine TestTwoFixedRounds() : Main where TestEnabled = true
{
    construct test cases for TwoFixedRounds
}

machine TestUpperBonus() : Main where TestEnabled = true
{
    construct test cases for upperBonus
}

machine TestHighestScore() : Main where TestEnabled = true
{
    construct test cases for HighestScore
}
```