Methodologies for Discrete-Event Modelling and Simulation

#### Assignment1: Cricket Arena

Antony Anty Kannampilly Jipson Johnson

Student ID : 101053630 Student ID : 101028751

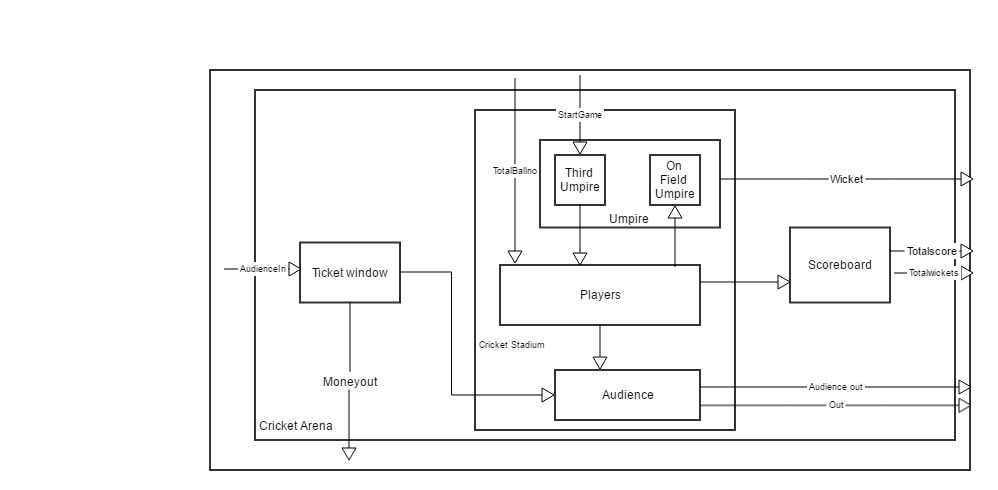
Carleton University

**Cricket Arena**

**Conceptual Model**

The system model describes the whole drama that occurs during a cricket match. A 20-20 cricket match is going on between two teams. The initial phase of the system would be people entering the cricket stadium through the ticket counter and ending phase would be people leaving the arena.

The Cricket Arena model is divided into 3 different sub-models: Ticket counter, Cricket Stadium, Score board. The Couple Model Cricket Stadium has 3 atomic models: Umpires, Players and Audience. The Couple Model Umpires has 2 atomic models: On-filed Umpires and Third Umpires.



**Structure of Cricket Arena**

When its match time, the audience enters the arena through the ticket counter where they are supposed to take tickets that cost $5 per ticket. The ticket counter sells the tickets till the seats in the stadium gets fully occupied. So, we get to know count of the crowd. After receiving the tickets, audience can directly go to the stadium and watch the game.

The cricket stadium is the next sub-model, where the whole cricket match happens. So, inside this sub-model we do have other sub-models that are players, umpires and audience. The players would be constituted into two teams, who would be playing against each other to determine the better side among them. To conduct the match in a smoothly manner we do have Umpires, and they are the final decision makers of the game. The third umpire decides the start time of the game and the onfield umpire decide whether a wicket fall or not. Then there are Audiences, who are considered an important part of the game, and their main job is to support their team by applauding when something amazing happens to their side. Audiences are sub divided into two: bowling team audiences and batting team audiences. So, when the batting team hits a six or a four, fans cheer their team and when they lose a wicket the bowling team fans gets their chance to boost the morale of their side.

The next sub-model is the scoreboard where ball by ball update of the game is provided. The score and the wickets are displayed on the scoreboard, so that the audience can cheer their teams. The scoreboard also shows some special animations when something extra ordinary happens, like when batsmen hits a four or a six or when a wicket falls. This would help the fans to boost their side and would bring a spirit among the audience and the players.

There is another situation where one team performs terribly bad and their supporters gets annoyed and they would leave the arena before end of the match. In any case, the supporters could leave the arena through the exit any time they wish too.

**Formal Specifications**

The formal specifications <S, X, Y, δint, δext, λ, ta> for the atomic models are defined as follows:

**Audience:**

State:

count1 = 0; **//if count1 >=0, out=1 ,which means batsman hits a six or four and batting team audience behavior: excited!**

count2 =0; **//if count2>=0, out=0 ,which means a wicket falls and bowling team audience behavior: excited!**

**//10 bowling team audiences will decide to leave if 7 wickets fall**

audienceNum=0;

S = {passive, active,count1,count2}

X = {in,audienceIn}

count2

audienceIn

Y = {out,audienceOut}

in

in

audienceOut

audienceNum

out

count1

δext (count1, count2, audienceNum ,e, x)

{ case phase

passive:

if x is from audieneIn and x>0{ **//receive audience**

audienceNum=audienceNum+audienceIn;

phase = active;}

if x is from in { **//receive current ball**

if(x=6 or 4){

count1++;

count2=0;}

if(x=0){

count2++;

count1=0}

phase = active;}

if(count2>=7){

if(audienceNum<10)

audienceNum=0;

if(audienceNum>=10)

audienceNum= audienceNum-10**;}//if 7 wickets fall, 10 of bowling team audience will decide to leave**

}

δint (phrase,sigma, audienceNum){

passivate();

}

λ(active)

{

send audienceNum to the port audienceOut**//output current //audiences who still want to watch**

if(count1>=0)

send 1to the port out **//batting team audience excited!**

if(count2>=0)

send 0 to the port out **//bowling audience excited!**

}

**TicketWindow:**

State Variables:

sigma = INFINITY, phase = Passive;

currentQueue1Num = 0; //the current number for audience

totalQueue1Num=0; // total audienceIn number, if the accumulated audience number //is equal or larger than 500, the TicketWindow will close //because seats are full

money=0; //money =5\*audience number

Formal specification:

X = {audienceIn}

Y = {moneyOut,audienceOut}

S = {phase, sigma, currentQueue1Num,money }

audienceInnn

moneyOut

Sigma

moneye

audienceOut

currentQueue1Numq

phase

δext (money, currentQueue1Num,e, x)

{ case phase

passive:

if x is from audieneIn and x>0{

totalQueue1Num = audienceIn; **//get input**

currentQueue1Num = totalQueue1Num-1; **//set initial queue num**

money=5; //set initial money

sigma = (0,0,3,0); **//transit to the state of sending moneyout**

phase = active;}

}

δint (money, currentQueue1Num,e, x)

{ case phase

active:

if (currentQueue1Num >0)

{ currentQueue1Num--;

sigma= (0,0,3,0);

money=money+5; **//set total moneyOut**

if(money>=2500){

phrase=passive;

sigma=INFINITY;

} **//the stadium is full now!**

}

}

λ(active)

{ send money to the port moneyOut**//send total money out**

send audience(money/5) to the port audienceOut **//send audience to the stadium**

}

**Players:**

State Variables:

sigma = INFINITY, phase = Passive;

currentBallNum = 0; //current ball number

totalBallNum = 0; //total balls number

temp1; //decides if the batsman score a run

tempS; //decides if the batsman hits a six

tempC; //decides if the bowling team gets a wicket

tempF; //decides if the batsman hits a boundary

score = 0; // 0 means the bowling team gets a wicket ,1 means the batting team score a run, 4 means batting team scored boundary and 6 means batting team scored a six

totalScore=0; //total runs scored

count = 0; //total wicket falled

count1 = 0;

wicket = 1;

Formal specification:

X = {totalBallNumIn, startMatch }

Y = {scoreOut,totalOut,totalW, startGame}

S = {phase, sigma, currentBallNum, score ,totalScore, wicket}

totalBallNumIn

totalW

Sigma

score

currentRoundNum

totalBallNum

totalOut

totalScore

Phrase

startMatch

startGame

wicket

δext (score, currentRoundNum, totalScore,e, x)

/\* totalBallNumIn ( the input )contains not only the total balls number, but other information. The format of the totalBallNumIn is a 6 or 7 bit number. The left 2 (6bit) or 3bit number indicates the total ball number. The fourth number from the right stands for temp1, and the third number ,the second number , the first number from the right represents tempS, tempC, tempF respectively.

Take number n=5005350 as an example: n/10000 is the total ball number; (the fourth number from the right \*20)/100 is the opportunity to score a run;(the third number from the right \*20-the first number from the right\*10)/100 is the opportunity to hit a six ;( the second number from the right\*4)/100 is the opportunity to loose a wicket.\*/

{ case phase

passive:

if x if from startMatch

{

get input into start;

count1 = 1;

if x is from totalBallNumIn and x>0{

totalBallNum = totalBallNumIn/10000; **//get input**

temp1=20\*( int((temp-totalBallNum\*10000)/1000));

tempS=20\*( int((temp-totalBallNum\*10000-temp1\*50)/100));

tempC=20\*( int((temp-totalBallNum\*10000-temp1\*50-tempS\*5)/10));

tempF=20\*((temp-totalBallNum\*10000-temp1\*50-tempS\*5-tempC/2));

currentBallNum = 1;

if(rand()%100>temp1){

score=1;

totalScore+=1; **// Batsman scores a run**

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else if(rand()%100>= tempS/2){

if(count<=10)

{

score=0;

totalScore+=0; **//Bowling team gets a wicket**

count++;

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else passivate();

}

else if (rand()%100>=tempS-tempF/2) {

score=6;

totalScore+=6; **//Batsman hits a six**

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else if(rand()%100>=tempF/2){

score=4;

totalScore+=4; //**Batsman hits a boundary**

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

}

}

}

δint (score ,currentBallNum,totalScore, e, x)

{ case phase

active:

if (currentBallNum < totalBallNum) **//begin next ball**

{ currentBallNum ++ ;

if(rand()%100>temp1)

{

score=1;

totalScore+=1;

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else

{

if(rand()%100>tempS-tempF/2)

{

score=6;

totalScore+=6;

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else if(rand()%100>=tempS/2)

{

if(count<=10)

{

score=0;

totalScore+=0;

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

else passivate();

}

else if(rand()%100>=tempF/2)

{

score=4;

totalScore+=4;

holdIn(active, Time( static\_cast<float>( fabs( distribution().get() ) ) ) );

}

}

else // all balls are over

{ phase = passive; **//change back to initial passive // state**

sigma = INFINITY;

}

}

λ(active)

{

if ( count1 == 1)

{

sendOutput( msg.time(), startGame, start ) ; //**send game started**

count1 = 0;

}

sendOutput( msg.time(), scoreOut, score ) **; //send current score**

if(currentBallNum>= totalBallNum)

sendOutput( msg.time(), totalOut, totalScore ) **;//send total score**

sendOutput( msg.time(), totalW, count-1 ) ; **send total wicket**

**//end**

}

**Score:**

S = {passive, active}

X = {inscore,inwicket}

Y = {outscore,outwicket}

out

in

Inscore

outscore

δint (active) = passive

δext (inscore,inwicket, passive) = active

outwicket

Inwicket

δext (inscore,inwicket, active) = active

λ(active)

{ send *inscore* to port *outscore* //**extract the final score**

send *inwicket* to port *outwicket* //**extract the final wicket**

}

ta(passive) = INFINITY

ta(active) = receiving\_time

**Onumpire:**

State:

out; //**0 means a wicket, 1 means not a wicket**

wicketin = 1; //**0 means a wicket, 1 means not a wicket**

S = {passive, active,wickketin)

X = {wicket}

Y = {yes\_wicket}

δext (wicketin)

{

case phase

passive :

if x is from port wicket

{

if wicketin == 0

out = 0;

else out = 1;

}

phase = active;

}

δint (wicketin,active)

{

passivate();

}

λ(active)

{

if (out == 0)

send out to the port yes\_wicket; //**decision is sent into output**

}

**Rdumpire:**

State:

game\_started = false; //**false means not started, true means game started**

period = 1;

S = {passive, active,game\_started)

X = {start\_game}

Y = {start\_period}

δext (game\_started)

{

if x is from port start\_game

{

if game\_started == false; //**if game not started**

{ //**then game\_statred is set true**

game\_started = true;

passivate ();

}

}

}

δint (game\_started)

{

if (period++< 3)

passivate();

}

λ(active)

{ if (period >= 2)

passivate();

else

send period to the port start\_period; //**game started output sent to the output**

}

The formal specifications <X, Y, D, {Mi}, {Ii}, {Zij}, SELECT > for the coupled model Cricket Stadium and Cricket Arena are defined as follows:

**Umpire:**

X = { c\_wicket, c\_start\_game};

Y = { c\_start\_period, c\_yes\_wicket};

D = { onumpire, rdumpire };

I( onumpire ) = players;

I( rdumpire ) = self;

Z( onumpire ) = self;

Z( rdumpire ) = self;

SELECT: ({onumpire, rdumpire) = rdumpire;

**Cricket Stadium:**

X = { totalBallNumIn,audienceIn,c\_start\_game };

Y = { totalScoreOut, totalWicketsout, c\_start\_period, c\_yes\_wicket};

D = { players, audience, umpire };

I(players) = {self, umpire};

I(audience) = {self,players};

I(umpire) = {self,players};

Z(players) = audience; Z(players) = umpire;

Z(players) = self;

Z(umpire) = self;

Z(umpire) = players;

Z(audience) = self ;

SELECT: ({players, umpire, audience}) = audience;

({players, umpire}) = umpire;

**Cricket Arena:**

X = { totalBallNumIn, audienceIn, c\_start\_game};

Y = { audienceOut, moneyOut ,out, totalScoreOut ,totalWicketOut, c\_start\_period, c\_yes\_wicket};

D = { score, ticketwindow, cricketstadium};

I(Cricketstadium) = (self,ticketwindow);

I(ticketwindow) = self ;

I(score) = cricketstadium;

Z(Cricketstadium) = score; Z(Cricketstadium) = self;

Z(ticketwindow) = cricketstadium; Z(ticketwindow) =self;

Z(score) = self;

SELECT: ({score, ticketwindow, cricketstadium}) = ticketwindow;

({score, cricketstadium }) = cricketstadium;

**Test Strategies**

The atomic models and coupled models will be tested using the “black box” testing method. Test cases are created by adding different combinations of inputs to the event file (*.ev*). The outputs in the output file (*.out*) will be checked to see if the result is expected.

**Atomic Model** **ticketwindow:**

The inputs of the ticketwindow are positive integers. The audiences will enter the arena through the ticket window where they pay $5 per ticket. Due to the limited seats inside the park, only 500 audiences can enter to the arena. In case, if the ‘audienceIn’ is larger than 500, after the 500th audience comes in, the ticketwindow will be passive.

The *.ev* file is created as follows:

00:20:00:00 audienceIn 500

The following is an example of the output file *.out*. In this output file.

.out

**00:20:03:000 moneyout 5**

**00:20:03:000 audienceout 1**

**00:20:06:000 moneyout 10**

**00:20:06:000 audienceout 2**

**00:20:09:000 moneyout 15**

**00:20:09:000 audienceout 3**

**00:20:12:000 moneyout 20**

**00:20:12:000 audienceout 4**

**00:20:15:000 moneyout 25**

**00:20:15:000 audienceout 5**

**00:20:18:000 moneyout 30**

**.**

**.**

**.**

**00:44:30:000 audienceout 490**

**00:44:33:000 moneyout 2455**

**00:44:33:000 audienceout 491**

**00:44:36:000 moneyout 2460**

**00:44:36:000 audienceout 492**

**00:44:39:000 moneyout 2465**

**00:44:39:000 audienceout 493**

**00:44:42:000 moneyout 2470**

**00:44:42:000 audienceout 494**

**00:44:45:000 moneyout 2475**

**00:44:45:000 audienceout 495**

**00:44:57:000 audienceout 499**

**00:45:00:000 moneyout 2500**

**00:45:00:000 audienceout 500**

**Atomic Model** **audience:**

One input to the atomic model Audience is ‘audience’, which should always be an integer. The other input is the current ball result which should be ‘6’,’4’,’1’ or ‘0’. Accordingly, one of the output is the current number of audiences who still want to watch the game (they may leave if the other team gets excited!) and the audience’s behavior(i.e., 0:batting team excited, 1:bowling team excited)the output of this model will be the batting or bowling teams audience gets excited. When the batting team scores a ‘6’ or ‘4’ the batting team audiences gets excited and when the batting team loses a wicket the bowling team audience gets excited.

The following is an example of the output file *.ev*. In this output file,audiences will come first. Then the game will begin. With game going on, batting audiences will be excited at certain timeand at some other time the bowling team get excited.

Example:

**.ev**

00:00:01:00 audienceIn 30

01:01:51:00 in 6

01:02:51:00 in 6

01:03:51:00 in 4

01:04:51:00 in 0

01:05:51:00 in 4

01:06:51:00 in 6

01:09:51:00 in 4

01:10:51:00 in 0

01:14:51:00 in 0

01:15:51:00 in 1

01:16:51:00 in 6

01:19:51:00 in 0

02:00:51:00 in 0

02:01:51:00 in 4

02:02:51:00 in 0

02:03:51:00 in 0

02:04:51:00 in 0

02:05:51:00 in 6

02:06:51:00 in 4

The following is an example of the output file *.out*. As it shows, the result is expected.

**.out**

**00:00:01:000 audienceout 30**

**01:01:51:000 audienceout 30**

**01:01:51:000 out 1**

**01:02:51:000 audienceout 30**

**01:02:51:000 out 1**

**01:03:51:000 audienceout 30**

**01:03:51:000 out 1**

**01:04:51:000 audienceout 30**

**01:04:51:000 out 0**

**01:05:51:000 audienceout 30**

**01:05:51:000 out 1**

**01:06:51:000 audienceout 30**

**01:06:51:000 out 1**

**01:09:51:000 audienceout 30**

**01:09:51:000 out 1**

**01:10:51:000 audienceout 30**

**01:10:51:000 out 0**

**01:14:51:000 audienceout 30**

**01:14:51:000 out 0**

**01:16:51:000 audienceout 30**

**01:16:51:000 out 1**

**01:19:51:000 audienceout 30**

**01:19:51:000 out 0**

**02:00:51:000 audienceout 30**

**02:00:51:000 out 0**

**02:01:51:000 audienceout 30**

**02:01:51:000 out 1**

**02:02:51:000 audienceout 30**

**02:02:51:000 out 0**

**02:03:51:000 audienceout 30**

**02:03:51:000 out 0**

**02:04:51:000 audienceout 30**

**02:04:51:000 out 0**

**02:05:51:000 audienceout 30**

**02:05:51:000 out 1**

**02:06:51:000 audienceout 30**

**02:06:51:000 out 1**

**Atomic Model** **Players:**

This model is considered as the most important model because it’s the place the actual games takes places and all the actions and reactions are based on this model. The batting team has the options to score ‘1’, ‘4’, ‘6’ and the bowling team can take wickets of the batsmen’s.

The input of the Players is a 6 or 7-bit number which contains many information about the game. Take number n=1501220 as an example: n/10000 is the total round number; (the fourth number from the right \*20)/100

is the opportunity to get one run;(the third number from the right \*20-the first number from the right\*10)/100 is the ratio of scoring a six;( the second number from the right\*4)/100 is the opportunity to get a wicket;( the first number from the right\*4)/100 is the opportunity to get a four.

Several *.ev* files will be set to test as follow:

Example:

.ev

00:00:00:00 start 1

00:00:00:00 totalBallNumIn 1501220

.out

**00:00:00:000 start\_yes 1**

**00:00:23:707 scoreout 6**

**00:00:28:600 scoreout 6**

**00:00:29:612 scoreout 0**

**00:00:54:462 scoreout 1**

**00:00:58:843 scoreout 1**

**00:01:26:699 scoreout 1**

**00:02:11:096 scoreout 0**

**00:02:50:821 scoreout 1**

**00:03:25:529 scoreout 1**

**00:03:45:463 scoreout 1**

**00:03:55:082 scoreout 1**

**00:04:35:572 scoreout 1**

**00:04:58:243 scoreout 1**

**00:05:38:520 scoreout 6**

**00:06:18:836 scoreout 1**

**00:08:25:070 scoreout 1**

**00:08:51:097 scoreout 1**

**00:09:03:899 scoreout 1**

**00:09:11:000 scoreout 1**

**00:09:21:061 scoreout 1**

**00:09:50:501 scoreout 1**

**00:11:59:341 scoreout 1**

**00:12:16:773 scoreout 1**

**.**

**.**

**.**

**01:21:11:157 scoreout 6**

**01:21:46:290 scoreout 1**

**01:22:01:101 scoreout 6**

**01:22:37:314 scoreout 1**

**01:22:37:314 totalout 242**

**01:22:37:314 totalw 8**

That is an expected result.

**Atomic Model** **Score:**

The only input of Score is the totalScoreIn and the totalWicketIn which are integers.While receiving the inputs, it will output the result. A simple example is as below:

.ev:

00:20:00:00 totalScoreIn 242

00:20:00:00 totalWicketIn 8

.out:

**00:20:00:000 totalscoreout 242**

**00:20:00:000 totalwicketout 8**

**Atomic Onfiled Umpires:**

The inputs of the onfiled umpires comes from the players and it provides an output ‘wickets’ the ‘out’ of the arena. The onfiled umpire deices whether the wicket is out or not. The example is shown:

Example:

.ev

00:00:00:00 c\_wicket 0

00:05:00:00 c\_wicket 1

00:07:00:00 c\_wicket 1

00:09:00:00 c\_wicket 0

.out

**00:00:00:000 c\_yes\_wicket 0**

**00:09:00:000 c\_yes\_wicket 0**

**Atomic model for Third Umpire:**

The third umpire is the one who decides when to start the game. The input for the third umpire comes from outside and the output from the third umpire goes to the players, stating to start the game. The example is shown below:

Example:

.ev

00:03:00:000 c\_start\_game 1

.out

**00:03:00:000 c\_start\_period 1**

**Coupled model for umpires:**

The inputs to this model comes from outside the arena which provides the information to start the game. The output to model provides an output to ‘start game’ to players and also it provides another output ‘wickets’ to outside the arena. The example is shown below:

Example:

.ev

00:03:00:00 c\_start\_game 1

00:04:00:00 c\_wicket 0

00:05:00:00 c\_wicket 1

00:07:00:00 c\_wicket 1

00:09:00:00 c\_wicket 0

.out

**00:03:00:000 c\_start\_period 1**

**00:04:00:000 c\_yes\_wicket 0**

**00:09:00:000 c\_yes\_wicket 0**

**Coupled Model Cricket Stadium:**

The inputs are from outside to the third umpires to ‘start game’, an input from the ticket counter to the audience and also an input ‘totalBallnumIn’ . The output is wickets, audienceout, totalwickets and totalscoreout and c\_start\_game, c\_yes\_wicket. A simple example is as below:

.ev:

00:00:00:00 c\_start\_game 1

00:00:00:00 totalBallNumIn 1501220

00:00:00:00 audienceIn 30

.out:

……

**00:00:00:000 audienceout 30**

**00:00:00:000 c\_start\_period 1**

**00:04:58:243 audienceout 30**

**00:04:58:243 out 0**

**00:04:58:243 c\_yes\_wicket 0**

**00:08:25:070 audienceout 30**

**00:08:25:070 out 1**

**00:16:28:752 audienceout 30**

**00:16:28:752 out 1**

**00:20:18:723 audienceout 30**

**00:20:18:723 out 1**

**00:20:32:074 audienceout 30**

**00:20:32:074 out 0**

**00:20:32:074 c\_yes\_wicket 0**

**00:22:38:475 audienceout 30**

**00:22:38:475 out 1**

**00:22:45:234 audienceout 30**

**00:22:45:234 out 1**

**00:23:25:952 audienceout 30**

**00:23:25:952 out 1**

**00:29:19:687 audienceout 30**

**00:29:19:687 out 1**

**00:31:43:910 audienceout 30**

**00:31:43:910 out 1**

**00:31:59:035 audienceout 30**

**00:31:59:035 out 1**

**00:33:33:708 audienceout 30**

**00:33:33:708 out 0**

**00:33:33:708 c\_yes\_wicket 0**

**00:44:44:091 audienceout 30**

**00:44:44:091 out 0**

**00:44:44:091 c\_yes\_wicket 0**

**00:45:09:231 audienceout 30**

**00:45:09:231 out 1**

**01:06:51:500 audienceout 30**

**01:06:51:500 out 1**

**01:08:59:388 audienceout 30**

**01:08:59:388 out 1**

**01:09:28:133 audienceout 30**

**01:09:28:133 out 1**

**01:10:38:241 audienceout 30**

**01:10:38:241 out 0**

**01:10:38:241 c\_yes\_wicket 0**

**01:12:49:887 audienceout 30**

**01:12:49:887 out 1**

**01:16:53:363 audienceout 30**

**01:16:53:363 out 0**

**01:16:53:363 c\_yes\_wicket 0**

**01:17:19:024 audienceout 30**

**01:17:19:024 out 0**

**01:17:19:024 c\_yes\_wicket 0**

**01:18:11:563 audienceout 30**

**01:18:11:563 out 1**

**01:22:37:314 totalout 209**

**01:22:37:314 totalw 7**

**Coupled Model Cricket arena:**

The inputs are totalBallNumIn, audienceIn, c\_start\_game.

The outputs are audienceOut moneyOut totalScoreOut totalWicketsout,out, c\_start\_period c\_yes\_wicket.

Example:

.ev:

00:00:00:00 audienceIn 520

02:00:00:00 c\_start\_game 1

02:00:00:00 totalBallNumIn 1501220

.out:

**00:00:03:000 moneyout 5**

**00:00:03:000 audienceout 1**

**00:00:06:000 moneyout 10**

**00:00:06:000 audienceout 2**

**00:00:09:000 moneyout 15**

**00:00:09:000 audienceout 3**

**00:00:12:000 moneyout 20**

**00:00:12:000 audienceout 4**

**00:00:15:000 moneyout 25**

**00:00:15:000 audienceout 5**

**00:00:18:000 moneyout 30**

**00:00:18:000 audienceout 6**

**00:00:21:000 moneyout 35**

**00:00:21:000 audienceout 7**

**00:00:24:000 moneyout 40**

**00:00:24:000 audienceout 8**

**00:00:27:000 moneyout 45**

**00:00:27:000 audienceout 9**

**00:00:30:000 moneyout 50**

**00:00:30:000 audienceout 10**

**00:00:33:000 moneyout 55**

**00:00:33:000 audienceout 11**

**00:00:36:000 moneyout 60**

**.**

**.**

**.**

**03:01:44:877 audienceout 500**

**03:01:44:877 out 0**

**03:01:44:877 c\_yes\_wicket 0**

**03:03:00:966 audienceout 500**

**03:03:00:966 out 1**

**03:10:18:367 audienceout 500**

**03:10:18:367 out 1**

**03:10:20:327 audienceout 500**

**03:10:20:327 out 1**

**03:11:33:507 audienceout 500**

**03:11:33:507 out 1**

**03:17:19:024 audienceout 500**

**03:17:19:024 out 1**

**03:20:00:590 audienceout 500**

**03:20:00:590 out 0**

**03:20:00:590 c\_yes\_wicket 0**

**03:20:37:543 audienceout 500**

**03:20:37:543 out 1**

**03:22:37:314 totalscoreout 237**

**03:22:37:314 totalwicketout 5**

**Summary:**

Based on the Cricket arena’s simple behavior, different models are created accordingly. The testing cases verify the specifications of models. The Cricket Arena works exactly as we expected according to the specifications.