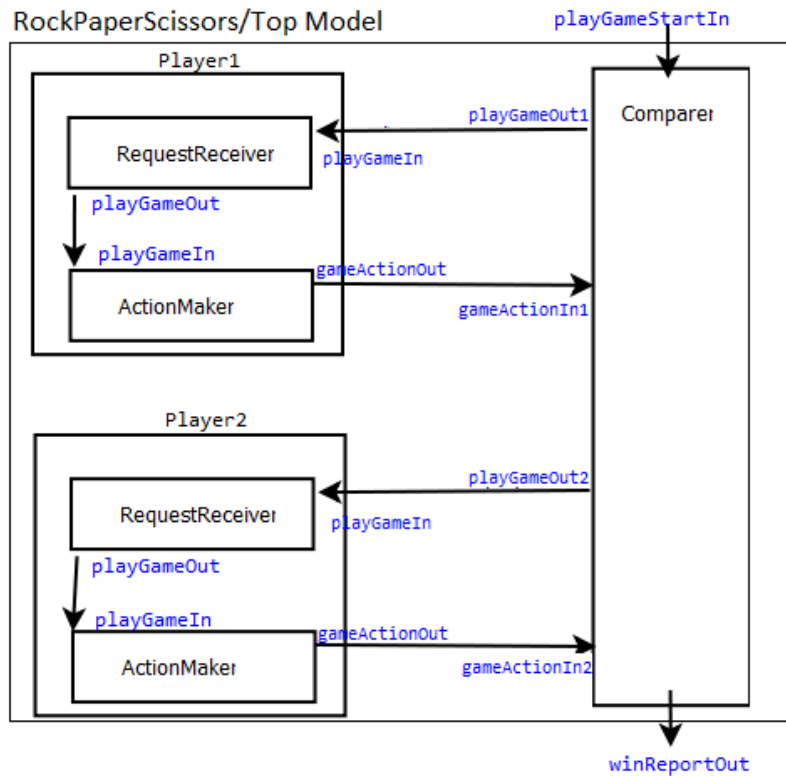# Assignment 1

Cadmium Simulation by Adapting a Rock Paper Scissors CD++ Model

Part 1: Model Selection

This assignment will cover the transformation of a CD++ model to be adapted in Cadmium. The model to be adapted is a CD++ simulation of a Rock-Paper-Scissors game with two players. Refer to the block diagram below for a representation of the models contained within.

There are three distinct atomic models, and two coupled models present within this DEVS model, constituting a complexity level of 2.

Model Name	Type	Purpose (Brief)
Comparer	Atomic	Referee role: Receives input to begin game and signals players to produce an output (rock, paper, or scissors). Upon receiving an input from each player, decides the victor, outputting a winReportOut.

Request Receiver	Atomic	Simulates the thought process within a human, and when the thought process turns from passive to active, sends an action request to Action Maker.
Action Maker	Atomic	Produces an action output to send back to Comparer, in the form of a selection from rock, paper, or scissors.
Player	Coupled	Contains request receiver and action maker, two models that simulate a typical human player.
Rock Paper Scissors Game/Top Model	Coupled	Contains the two player coupled models, as well as the comparer atomic model.

Data Structures:

There are a few data structures that will be used for testing and passing data between our models. Please refer to the table below for more details.

Structure Name	Variable (Type) in Structure	Purpose
PlayGame_t	isTriggerGame (bool)	Signify if a game request has been sent or received; true for sent, false for not sent.
GameAction_t	choice (int)	Rock (1), paper (2), or scissor (3) choice made by player
WinReport_t	winner(int)	Winner player id stored here (1 or 2).

Part 2: Model Description and DEVS Formalism with Sample Test Cases

Comparer (Atomic):

The Comparer atomic model begins the game upon receiving a trigger signal input of type PlayGame_t from the playGameStartIn input port. From there, it waits 5 seconds before triggering the output function to alert the players that the game has begun. Once the players have made their choice, it is entered into the Comparer sub-model through the gameAction1/gameAction2 input ports, of type GameAction_t. In this case, the ta or time advance to an internal transition is 15 seconds, since we are simulating how long a referee would take to make a decision. After 15 seconds, the Comparer will make a decision on the victor using logic coded inside the model. The output port, winReportOut, will deliver the final result of who the winner is once the output function is triggered (0 signifies a tie). This is stored as type

WinReport_t. Every winner from each round will be stored in a vector by the name of winnerTracker, and the player who is leading the score with total wins will be displayed in the state variable leading.

Comparer = < X, Y, S, δint, δext, λ, ta >

X = {(playGameStartIn ∈ {true, false}), (gameActionIn1 ∈ {1, 2, 3}), (gameActionIn2 ∈ {1, 2, 3})};

Y = {(playGameOut1 ∈ {true, false}), (playGameOut2 ∈ {true, false}), (winReportOut ∈ {0 ,1, 2})};

S {(active ∈ {true, false}), (playerResult1 ∈ {-1,1,2,3}), (playerResult2 ∈ {-1,1,2,3}), (received1 ∈ {true, false}), (received2 ∈ {true, false}, (playerIDWin ∈ {-1,0,1,2}), (winnerTracker ∈ {[{1,2},,,]})}, (leading ∈ {0,1,2})}}

default state values in code

```
// default constructor
Comparer() {
        state.active = false; //set to true when input received
        state.playerResult1 = -1; //default setting, meaning no response yet from player
        state.playerResult2 = -1; //default setting, meaning no response yet from player
        state.received1 = false; //set to true when plyer response received
        state.received2 = false; //set to true when plyer response received
        state.playerIDWin = -1; //no winner selected yet
        state.winnerTracker; //tracks the scores of the past rounds with playerIDs
        state.leading=0; //tracks which player has one more games in rounds played

}
```

δint(s){

```
        //return to default settings
        state.playerIDDisplay = state.playerIDWin;
        state.leadingDisplay = state.leading;
        state.active = false;
        state.received1 = false;
        state.received2 = false;
        state.playerIDWin = -1;
        state.playerResult1 = -1;
        state.playerResult2 = -1;
        }
```

δext(s, e, x){

```
        if (get_messages<typename Comparer_defs::playGameStartIn>(mbs).size() > 1) {
            assert(false && "One game at a time!"); //Make sure more than one game is not
        started at once
        }

        //if current game is ongoing, refuse
        if (get_messages<typename Comparer_defs::playGameStartIn>(mbs).size() == 1) {
            //trigger game to start
            if (state.active == false) {
```

```
                        state.active = true;
                    }
                }
                //receiving player input
                if (get_messages<typename Comparer_defs::gameActionIn1>(mbs).size() == 1) {
                    state.active = true;
                    if (state.active == true && state.received1 == false){
                        vector<GameAction_t> player1;
                        player1 = get_messages<typename Comparer_defs::gameActionIn1>(mbs);

                        state.playerResult1 = player1[0].choice;
                        state.received1 = true;
                    }
            }

        λ(s){

    typename make_message_bags<output_ports>::type bags;
    if (state.active == true && state.received1 == true && state.received2 == true &&
    state.playerIDWin != -1) {
        vector<WinReport_t> report;
        report.push_back(state.playerIDWin);
        get_messages<typename Comparer_defs::winReportOut>(bags) = report;


    }
    else if (state.active == true && state.received1 == false && state.received2 == false) {
        vector<PlayGame_t> playerTrigger1;
        vector<PlayGame_t> playerTrigger2;

        //trigger players to provide response, tell them game has begun
        playerTrigger1.push_back(true);
        playerTrigger2.push_back(true);

        get_messages<typename Comparer_defs::playGameOut1>(bags) = playerTrigger1;
        get_messages<typename Comparer_defs::playGameOut2>(bags) = playerTrigger2;
        return bags;

    }

    return bags;

    }

    ta(s){

    TIME next_internal;
    if (state.active == true && state.received1 == false && state.received2 == false) {
        next_internal = TIME("00:00:05:000"); //referee time to alert players
    }
    else if (state.active == true && state.received1 == true && state.received2 == true) {
        next_internal = TIME("00:00:15:000"); //referee time to make winning decision
    }
    else {
        next_internal = numeric_limits<TIME>::infinity();
    }
return next_internal;
```

}

Improvements to CD++ Model:

In the CD++ model, if another input to the model is received while the RequestReceiver is still in the time advance, then the entire model will restart. This causes an issue, as if the model were to keep receiving continuous input, then it would never finish the time advance and execute the state functions or output function. This model stops that from happening, because once the output function has triggered in the Comparer, the internal function will set the active variable to false, with a condition in the output function to only output from the ports when the state is active. As such, the Comparer will only be active until one output has been delivered, or until another game trigger has been received.

Additionally, this comparer model also tracks which player is in the lead out of the current rounds that have been played. This will be mentioned in the state log files.

Testing Strategy:

There are three inputs to the Comparer model. The first input, playGameStartIn, triggers the comparer to begin the sequence. At 28 seconds past the game was first initialized, both players should return their choice for the game, from which the Comparer will take 15 seconds to make a decision on which player won the game. Three inputs will be given to each input port, representing three different rounds of rock, paper, scissors. One game takes 43 seconds to complete. The input data for each port can be seen below. Note: The requestReceiver input and Comparer playGameStartIn input ports use the same test file, since the input types are both of type PlayGame_t.

requestReceiver_input_test	player1_choice_input_test	player2_choice_input_test
File Edit Format View H	File Edit Format View H	File Edit Format View H
00:02:00:000 1	00:02:28:000 1	00:02:28:000 1
00:04:00:000 1	00:04:28:000 2	00:04:28:000 3
00:06:00:000 1	00:06:28:000 3	00:06:28:000 2

The expected output is that the first game will be a tie, the second game will be won by player 2, and the last game will be won by player 1. Hence there overall leading player variable will be set to 0, since both players won an equal amount of times. The playerID of the winning player will output every round.

RequestReceiver (Atomic):

The RequestReceiver model is simply meant to imitate the human instinct of receiving a request, pondering about the decision, and then sending a request a notification to the next module to make an action. No action is made in this model. A trigger is received from the input port playGameIn of the type PlayGame_t. The state of the model is then set to active. After a

time advance of twenty seconds, the output function is executed, sending an output message from the port playGameOut to the ActionMaker model of type PlayGame_t. The internal transition function then executes, changing the state variable sent to true, and resetting the active variable, signifying a request to ActionMaker has been sent.

RequestReceiver = < X, Y, S, δint, δext, λ, ta >

X = {(playGameIn ∈ {true, false})};

Y = {(playGameOut ∈ {true, false})};

S {(active ∈ {true, false}), (sent ∈ {true, false})};

```
// default constructor
RequestReceiver() {
    state.active = false; //true if game request from Comparer has been received
    state.sent = false; // true if game request has been sent from RequestReceiver to
ActionMaker
}
```

δint(s){

```
state.active = false;
state.sent = true;
```

}

δext(s, e, x){

```
if (get_messages<typename RequestReceiver_defs::playGameIn>(mbs).size() > 1) {
            assert(false && "One request at a time!"); //Make sure more than one request
is not made at the same time
        }
        else {
            if (state.active == false) {
                state.active = true;
                state.sent = false;
            }
        }
}
```

λ(s){

```
typename make_message_bags<output_ports>::type bags;

if (state.active == true) {
    vector<PlayGame_t> playerTrigger;

    //tell next model to stay ready to make decision
    playerTrigger.push_back(true);
    get_messages<typename RequestReceiver_defs::playGameOut>(bags) = playerTrigger;

}
return bags;

}
```

```
ta(s){

TIME next_internal;
if (state.active == true) {
   next_internal = TIME("00:00:20:000"); //decision making time
}else {
   next_internal = numeric_limits<TIME>::infinity();
}
return next_internal;
}
```

Improvements to CD++ Model:

The original simulation only requires the RequestReceiver to have a time advance, or thinking time, of 1 second. However, that is not a legitimate representation of a realistic time to ponder. As such, the thinking time for this model was set to 20 seconds. Also, there is another state variable, sent, that signifies if the request has been sent to the ActionMaker, once the 20 seconds are up.

Testing Strategy:

There is only one input port, of type PlayGame_t to the RequestReceiver model. The only expected out is that after 20 seconds of receiving the input request, the model should output the trigger to ActionMaker to make a request through the playGameOut port.
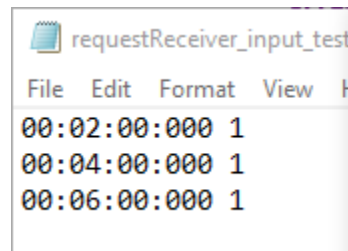
ActionMaker (Atomic):

The ActionMaker model simulates receiving a request to make an action, spending three seconds to simulate human hand motion, ("Rock, Paper, Scissors!"), and present their final decision. The playGameIn input port of type PlayGame_t is inputted to the model, and once a decision is made, the gameActionOut carries out a decision of type GameAction_t, which essentially holds the choice as an integer value: Rock -1, Paper-2, Scissors-3. The internal transition will reset the states to their default values.

ActionMaker $= <$ X, Y, S, δint, δext, λ, ta $>$

X = {(playGameIn $\in$ {true, false})};

Y = {(gameActionOut $\in$ {1, 2, 3})};

S {(active $\in$ {true, false}), (choice $\in$ {-1, 1, 2, 3})};

```
// default constructor
ActionMaker() {
    state.active = false; // set to true once game request received
    state.choice = -1; //default is -1, means no choice, otherwise 1-3
}
```

$\delta int(s)\{$

```
state.active = false;
state.choice = -1;

}
```

$\delta ext(s, e, x)\{$

```
if (get_messages<typename ActionMaker_defs::playGameIn>(mbs).size() > 1) {
        assert(false && "One request at a time!"); //Make sure more than one request
is not made at the same time
    }
    else {
        if (state.active == false) {
            state.active = true;
            std::random_device rand;
            std::mt19937 generate(rand());
            std::uniform_int_distribution<> distribute(1, 3);
            state.choice = distribute(generate); //generate value between 1-3
        }
    }

}
```

$\lambda(s)\{$

```
typename make_message_bags<output_ports>::type bags;
if (state.active == true) {
    vector<GameAction_t> gameChoice;

    //output choice
    gameChoice.push_back(state.choice);
    get_messages<typename ActionMaker_defs::gameActionOut>(bags) = gameChoice;
}
return bags;

}
```

```
ta(s){

TIME next_internal;
if (state.active) {
    next_internal = TIME("00:00:03:000"); //time to provide signal
}else {
    next_internal = numeric_limits<TIME>::infinity();
}
return next_internal;
}
```

Improvements to CD++ Model:

The original simulation only requires the ActionMaker model to have a time advance, or time to make an action, 0.5 seconds. This does not represent a real rock, paper, scissors game accurately, so the time advance was set to 3 seconds for this model.

Testing Strategy:

There is only one input port, of type PlayGame_t to the ActionMaker model. After the time advance of three seconds, the model will output an integer choice back to the Comparer model through the gameActionOut port. Once again, the same input file as before is used.
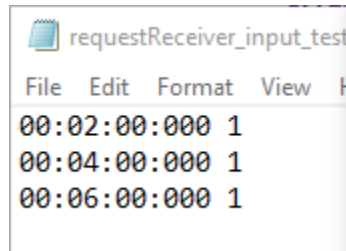
Player (Coupled):

There are two player coupled models present in this simulation, Player1 and Player2. The player model contains the RequestReceiver and ActionMaker atomic models, with a total of one input from Comparer to signify the game has started of type PlayGame_t, and an output returning to Comparer with the player's choice of type GameAction_t.

Player = < X, Y, D, {Mi}, IC, EIC, EOC, select >

X = {(playGameIn ∈ {true, false})};

Y = {(gameActionOut ∈ {1, 2, 3})};

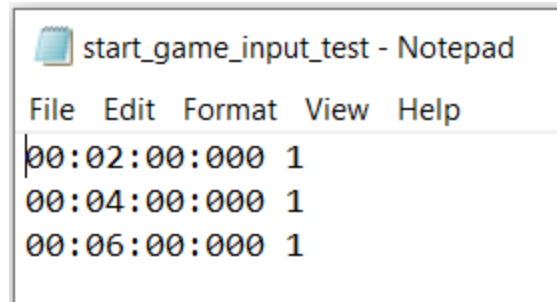D = {RequestReceiver, ActionMaker};

Mi = {M_{RR}, M_{AM}};

IC = {playGameOut@RequestReceiver → playGameIn@ActionMaker};

EIC = {playGameIn → playGameIn@RequestReceiver};

EOC = {gameActionOut@ActionMaker → gameActionOut};

Testing Strategy:

There is only one input port, of type PlayGame_t to the Player coupled model (which directly feeds to the requestReceiver atomic model inside). Once the transition is performed to ActionMaker, the output should be of type GameAction_t. The Player model should take 23 seconds to output a choice. The input file is shown below:

start_game_input_test - Notepad

File Edit Format View Help

```
00:02:00:000 1
00:04:00:000 1
00:06:00:000 1
```

Top Model (Coupled):

The top model ties all models together as shown in the diagram at the beginning of this report. This includes the two player coupled models and the comparer atomic model.

Player = < X, Y, D, {Mi}, IC, EIC, EOC, select >

X = {(playGameStartIn ∈ {true, false})};

Y = {(winReportOut ∈ {0, 1, 2})};

D = {Player1, Player2, Comparer};
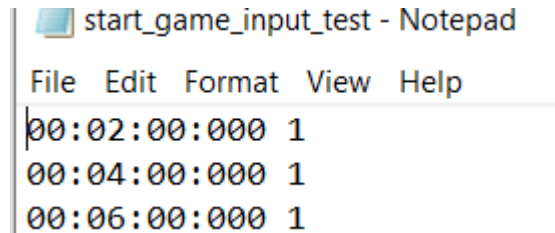
Mi = {M_{P1}, $M_{P2,}$ M_C };

IC = {playGameOut1@Comparer → playGameIn@Player1, playGameOut2@Comparer → playGameIn@Player2, gameActionOut@Player1 → gameActionIn1@Comparer, gameActionOut@Player2 → gameActionIn2@Comparer};

EIC = {playGameStartIn → playGameStartIn@Comparer};

EOC = {winReportOut@Comparer → winReportOut};

Testing Strategy:

Testing this coupled model is synonymous to testing the entire sequence and simulation. The total time taken by the top model should be 43 seconds and required an input of type PlayGame_t (feeding into Comparer), and an output of type WinReportOut_t (feeding out of Comparer). The input file is shown below, with three rounds starting 2 minutes apart from each other.

start_game_input_test - Notepad

File Edit Format View Help

```
00:02:00:000 1
00:04:00:000 1
00:06:00:000 1
```

Part 3: Model Simulations, Results and Discussion

Comparer (Atomic):

```
00:00:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {}] generated by model input_reader_play_game
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {}] generated by model input_reader_player1_choice
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {}] generated by model input_reader_player2_choice
00:02:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:02:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:02:28:000
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {1}] generated by model input_reader_player1_choice
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {1}] generated by model input_reader_player2_choice
00:02:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {0}] generated by model comparer1
00:04:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:04:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:04:28:000
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {2}] generated by model input_reader_player1_choice
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {3}] generated by model input_reader_player2_choice
00:04:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {2}] generated by model comparer1
00:06:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:06:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:06:28:000
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {3}] generated by model input_reader_player1_choice
[cadmium::basic_models::pdevs::iestream_input_defs<GameAction_t>::out: {2}] generated by model input_reader_player2_choice
00:06:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {1}] generated by model comparer1
```

Looking over the simulation results, the results were as expected. The time for each round is 43 seconds, the first round was a tie, and the later rounds were won by player 2 and then player 1 respectively. Hence the overall leader is 0, since each player won the rounds an equal amount of times. Refer to *Current round leader* in the output below.

```
00:06:43:000
State for model input_reader_play_game is next time: inf
State for model input_reader_player1_choice is next time: inf
State for model input_reader_player2_choice is next time: inf
State for model comparer1 is Active? : 0 & Response from P1 : -1 & Response from P2 : -1 & winner is player : 1 Current round leader is: 0
```

RequestReceiver (Atomic):

```
00:00:00:000
00:00:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {}] generated by model input_reader
00:02:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:02:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
00:04:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:04:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
00:06:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:06:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
```

Simulation results were correct, since after initialization, the playGameOut port was triggered after twenty seconds. Likewise, the state variables were reset appropriately.

```
00:02:00:000
State for model input_reader is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 1 Sent? :0
00:02:20:000
State for model input_reader is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 0 Sent? :1
00:04:00:000
State for model input_reader is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 1 Sent? :0
00:04:20:000
State for model input_reader is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 0 Sent? :1
00:06:00:000
```

ActionMaker (Atomic):

```
00:00:00:000
00:00:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {}] generated by model input_reader
00:02:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:02:03:000
[ActionMaker_defs::gameActionOut: {1}] generated by model actionMaker1
00:04:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:04:03:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker1
00:06:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:06:03:000
[ActionMaker_defs::gameActionOut: {1}] generated by model actionMaker1
```

Simulation results were correct, since after initialization, the gameActionOut port was triggered after three seconds with a random value between 1-3. Likewise, the state variables were reset appropriately after triggering the internal transition function once the time advance was complete.

```
00:02:00:000
State for model input_reader is next time: 00:02:00:000
State for model actionMaker1 is Active? : 1
00:02:03:000
State for model input_reader is next time: 00:02:00:000
State for model actionMaker1 is Active? : 0
```

Player (Coupled):

The total simulation time was 23 seconds as expected, with the RequestReceiver and ActionMaker taking their expected time to transition. The output port, gameActionOut was also verified, with a random number generated between 1-3 each round. This ensured all external and internal couplings were done correctly.

```
00:00:00:000
00:00:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {}] generated by model input_reader
00:02:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:02:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:02:23:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker2
00:04:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:04:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:04:23:000
[ActionMaker_defs::gameActionOut: {3}] generated by model actionMaker2
00:06:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader
00:06:20:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:06:23:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker2
```

Top Model (Coupled):

This the main simulation that covers the entire three rounds of the Rock Paper Scissors Game. The total time for each round was 43 seconds, and each started at 00:02:00, 00:04:00, and 00:06:00, as mentioned before. The state variables were reset where required to not affect future simulations as expected. The current round leader was calculated after every time the Comparer chose a round winner, to display which player was in the lead.

Round 1:

```
00:02:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:02:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:02:25:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:02:28:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker1
[ActionMaker_defs::gameActionOut: {3}] generated by model actionMaker2
00:02:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {2}] generated by model comparer1
00:04:00:000
```

-   Player 2 won the first round, hence, player 2 is in the lead

```
00:02:43:000
State for model input_reader_play_game is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 0 Sent? :1
State for model actionMaker1 is Active? : 0
State for model requestReceiver2 is Active? : 0 Sent? :1
State for model actionMaker2 is Active? : 0
State for model comparer1 is Active? : 0 & Response from P1 : -1 & Response from P2 : -1 & winner is player : 2 Current round leader is: 2
```

Round 2:

```
00:04:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:04:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:04:25:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:04:28:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker1
[ActionMaker_defs::gameActionOut: {1}] generated by model actionMaker2
00:04:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {1}] generated by model comparer1
```

- Player 1 won this round, so no player is in the lead (0)

```
00:04:43:000
State for model input_reader_play_game is next time: 00:02:00:000
State for model requestReceiver1 is Active? : 0 Sent? :1
State for model actionMaker1 is Active? : 0
State for model requestReceiver2 is Active? : 0 Sent? :1
State for model actionMaker2 is Active? : 0
State for model comparer1 is Active? : 0 & Response from P1 : -1 & Response from P2 : -1 & winner is player : 1 Current round leader is: 0
```

Round 3:

```
00:06:00:000
[cadmium::basic_models::pdevs::iestream_input_defs<PlayGame_t>::out: {1}] generated by model input_reader_play_game
00:06:05:000
[Comparer_defs::playGameOut1: {1}, Comparer_defs::playGameOut2: {1}, Comparer_defs::winReportOut: {}] generated by model comparer1
00:06:25:000
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver1
[RequestReceiver_defs::playGameOut: {1}] generated by model requestReceiver2
00:06:28:000
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker1
[ActionMaker_defs::gameActionOut: {2}] generated by model actionMaker2
00:06:43:000
[Comparer_defs::playGameOut1: {}, Comparer_defs::playGameOut2: {}, Comparer_defs::winReportOut: {0}] generated by model comparer1
```

- Both players, picked the same choice so round was tie, meaning no one is in the lead still
- Last round, end of simulation

```
00:06:43:000
State for model input_reader_play_game is next time: inf
State for model requestReceiver1 is Active? : 0 Sent? :1
State for model actionMaker1 is Active? : 0
State for model requestReceiver2 is Active? : 0 Sent? :1
State for model actionMaker2 is Active? : 0
State for model comparer1 is Active? : 0 & Response from P1 : -1 & Response from P2 : -1 & winner is player : 0 Current round leader is: 0
```

Part 4: Conclusion

In short, the rock, paper, scissors game simulation in Cadmium was successfully implemented and simulation results were generated. The Cadmium model was improved in many ways compared to the original CD++ model, as mentioned in Part 2 of this report.