VISUALIZATION OF CELL DEVS MODEL – 3D PIN BALL

Submitted To

Prof. Gabriel Wainer

SYSC-5104 FALL(2016)

By

Prasanthi Bobbili

Student ID 101057215

CARLETON UNIVERSITY

Yamini Nibhanupudi

Student ID 101050830

CARLETON UNIVERSITY

ABSTRACT

Pinball is a computer game that everyone must have played at least once in their childhood. The primary objective of this game is to score as many high points as a player can. Many pinball games include a player earning high points when the ball strikes certain targets and preventing the ball from falling down the drain. The game is over when the ball falls into the drain. Earlier versions of pinball games were the actual machine and later on simulating the game was attempted. Simulating a pinball game has been done several times in the past. Many simulations included accurate physics of rolling steel balls, tilting and many other graphics. This paper is inspired by this pinball game. Pin Ball game is simulated using CELL-DEVs and then, python and blender are used to create a 3D model of it. There are two versions included in this paper, namely, PINBALL GAME WITH TWO BALLS and PINBALL WITH AN EXIT.

1. INTRODUCTION

Cell Devs is highly used in complex systems to make a model out of it and help make simulations more users friendly. It is all about modeling any real-time scenario to understand their behavior and time taken to accomplish the task at a cellular level. In CD++, the cell-DEVS model is designed with some set of rules and then simulated using Omar Hisham tool to check the simulation in 2D. Then this simulation is used to visualize in 3D using Blender. This gives a real feel and is useful for making creative animations of the cell-DEVS model which we have on hand. This paper talks more about cell-DEVS, its formalism, applications, 3D visualization and tools used to do it. It is an improvement from the previous model as we had only 1 ball in pinball game, where as in the new version we have 2 balls and an another case with 3 balls with an exit

2. RELATED WORK

Since many years there had been a lot of research being done on CELL-Devs. We have used the basic principles of cell-Devsto design the model which we decided to visualize in 3D. The related work is presented in the subsequent sections.

2.1 CELL DEVS

Cell Devs is a result of executing a global transition function which updates the cell states in the space. The total behavior of the transition function depends on the result of each cell how it locally executes that particular function. All the cells in the cell space are computed at a time synchronously and in parallel using the

existing cell and its neighbors' state value. This cell Devs helps in saving time efficiently by not wasting computation time for computing each cell. Below is the figure which represents the cell neighborhood.

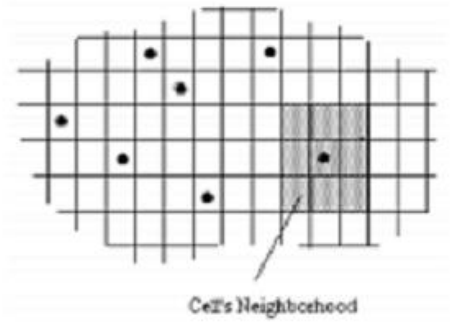

Figure 1: CELL DEVS – CELL NEIGHBOURHOOD

2.2 CELL DEVS FORMALISM

Cell-Devs formalism is based on cellular model's DEVS. It has a space of cells, wherein each cell has a value and its current state along with neighbor's values, defines the next value.
Each cell is defined as DEVS model and the whole cell space is coupled model. A computation function τ (tow) is used to compute the future state of the current cell based on some input values of the cell. Delay d is associated with each cell like inertial delay and transport delay which effects the time the output is sent (this can be seen in Figure 3). Once each cell is computed, whole cell space is built using a coupled Cell-DEVS model.

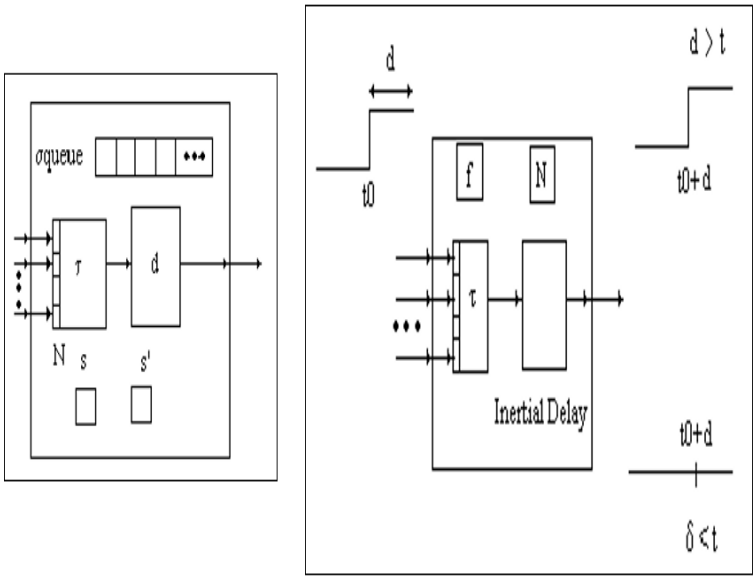

Figure 2: cell-DEVS Atomic Models

Each DEVS model can be built as a behavioral (atomic) or a structural (coupled) model. A DEVS atomic model is described as:

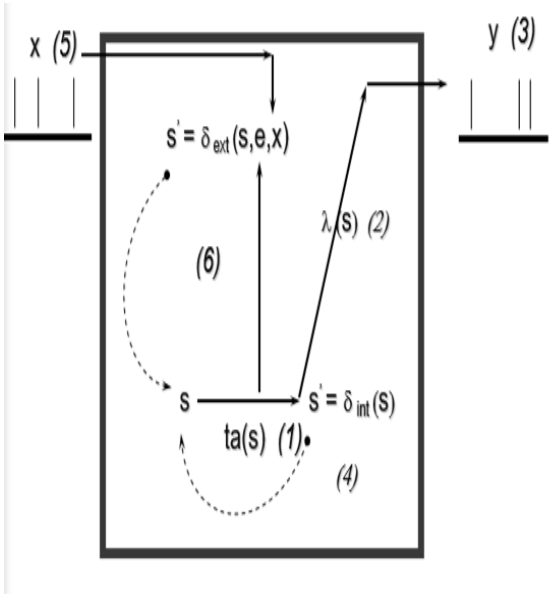

Figure 3: Atomic DEVS model

M = <X, S, Y, δint, δext, λ, D >

In the absence of external events, the model will remain in state s∈S during ta(s). Transitions that occur due to the expiration of ta(s) are called internal transitions. When an internal transition takes place, the system outputs the value λ(s) ∈ Y, and changes to the state defined by δint(s). Upon reception of an external event, δext(s, e, x) is activated using the input value x belongs to X, the current state s and the time elapsed since the

last transition e. Coupled models are defined as:

CM = < X, Y, D, {Mi}, {Ii}, {Zij} , select >

They consist of a set of basic models (Mi, atomic or coupled) connected through their interfaces. Component identifications are stored into an index (D). A translation function (Zij) is defined by using an index of influences created for each model (Ii). The function defines which outputs of model Mi are connected to inputs in model Mj. The select function serves as tiebreaker for two simultaneous models. A Cell-DEVS model is represented as a cell space, where each cell is represented as an atomic DEVS model. Each cell is connected to the

neighboring cells. A delay mechanism in each cell (transport delay or inertial delay) is used to delay the propagation of state change events through the cell space, providing the means for defining complex temporal behavior.

An Atomic Cell-DEVS can be defined as follows:

TDC = < X, Y, I, S, θ, N, d, δint, δext, τ, λ, D >

Where X is the set of external input events; Y is the set of external output events; I represents the definition of the model's modular interface; S is the set of possible states for a given cell; θ is the definition of the cell's state variables. N is the set of values for the input events; d is the delay of the cell; δint is the internal transition function; δext is the external transition function; τ is the local computing function; λ is the output function, and D is the duration function. A Coupled Cell-DEVS model is built by connecting a number of Atomic Cell-DEVS models together into a cell space (including 2D and 3D cell spaces). The borders of the cell space can be either wrapped, in which case the cells at the border from one side of the cell space are considered neighbors to the cells at the border on the opposite side of the cell-space, or non-wrapped, in which case the border cells must have special rules defined by the modeler. A formal definition of Coupled Cell-DEVS is:

GCC = <Xlist, Ylist, I, X, Y, n, {t1,..,tn}, N, C, B, Z, select >

Where Xlist is the input coupling list; Ylist is the output coupling list; I represents the definition of the model's modular interface; X is the set of external input events; Y is the set of external output events; n is the dimension of the cell space; {t1,….,tn} is the number of cells in each of the dimensions. N is the neighborhood set; C is the cell space; B is the set of border cells; Z is the translation function; and select is the tie-breaking function.

3. VISUALIZATION

Visualization in 3D of a Cell-DEVS model gives a real feel of the model that was done in CD++.

This paper talks about a software for 3D visualization called "Blender". Blender is a powerful application for threedimensional modeling and animation. The software GUI is very easy to understand and user friendly. Hence these tools within blender are used to create 3D environment for Cell-DEVS and DEVS models. To do that, user must develop a python script to interface the CD++ ma, log and val files to blender. When the user developed python script is run, it should automatically execute the files and start animation. Below is the figure that represents how the blender works with CD++ files.
Cell-DEVS is an extension to DEVS that uses Cellular Automata. The entire cell space is decomposed into a number of individual cells. Although cell-DEVS is a form of visualization it is only in 2D. Hence, in this project, to visualize in 3D, we used Blender 2.6.

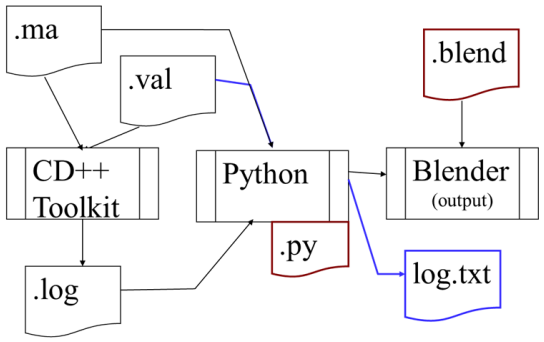

Figure 4: Architecture

Blender is a 2D and 3D visualization tool. It supports Python scripting. A Blender software can be used for 3D animations and graphics. For this project, no changes were made to Blender itself, but rather the Python scripting was used to read and animate the cell-DEVS model. The python software used in this project is Python 2.4.4. Python is a high-level object-oriented programming language.

3.1 PYTHON SCRIPT

For the CD++ files to be blended with blender tool, an interface APIs are required to this job. That's where a python script is required. Hence, we have two scripts written to import the files

from CD++ and then dictate the blender tool on how to place the objects based on ma, log and val files in the working directory. Hence, we need cdpp and filecontrol file.

3.1.1 FILE CONTROL FILE

This program is designed to provide the interface to the MA, VAL and LOGfile generated for/by the CD++ simulation tool.
This file is the file that is run as a script and will dynamically import a selected file name contained within the *.blend file directory.
This file will create a reference to the class defined in the selectedfile called MyClass. If the class is not present or renamed, this file will fail to execute and generate errors. The file selection 'drop down' or combo box for the MA, VAL, and LOG files are dynamically generated and are regenerated each time the script is run or when the script is moved to another directory. It is required that valid ma, val and log files to be present in the working directory of blender where the .blend file is located. Else, the script is written in such a way that it throws errors in case the files not found in the working directory.

3.1.2 CDPP File

A cdpp script is unique for its application. A cdpp file is used to extract the .MA, .VAL and .LOG file of the cell-DEVS model from the directory (the .py, .ma, .val and .log files should be present in the same directory）.

3.2 APPROACH AND RESULTS

Since, both blender and python are new to us, we need to first understand the blender tool very well by running old projects and creating simple projects. We got hands-on on blender by designing few new objects and linking and de-linking them from scene to scene. Then, we have gone through the python syntax and various APIs included for python-blender. Understood the previous projects where python scripts were designed using those APIs. Then created a sequence of jobs to be done to accomplish the visualization of our project. Hence our approach we followed was to take an existing PIN Ball CELL-DEVS model, make modifications for it, develop the test cases and then create visualization model.

3.2.1 CD++ Tool

CD++ toolkit as workbench (Wainer 2009) was used in this project to design a Cell Devs model. CD++ implements DEVS and Cell-DEVS theories. The defined models are built as a class hierarchy, and each of them is related with a simulation entity that is activated whenever the model needs to be executed. New models can be incorporated into this class hierarchy by writing DEVS models in C++, overloading the basic methods representing DEVS specifications: external transitions, internal transitions and output functions. The design of PIN ball was done in CD++ using some rules which are presented in the subsequent sections. They also include devs formalism.

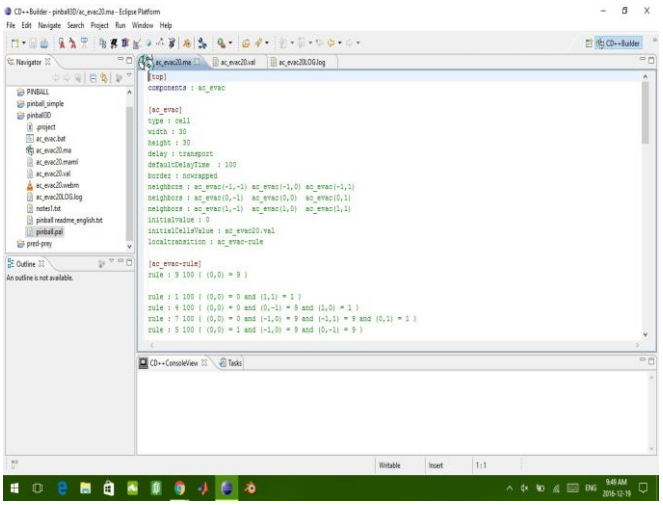

Figure 5: CD++ Screenshot

3.2.2 PIN BALL CELL DEVS FORMALISM

The inputs, outputs and neighbors of the pin ball cell devs model are as follows:

M : <Xlist, Ylist, I, X, Y, η, N, {r, c}, C, B, Z, select >

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

Ylist : {0,1,2,3,4,5,6,7,8}

I : <Px, Py>, Px= Null , Py = Null

X : Null

Y : Null

η : 9

N : { (-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1) }

r : 19

c : 19

S : {0, 1}, i.e. either occupied or empty

B : nowrapped
Dimensions: 19 x 19 cells

Delay : transport

Border : nowrapped

3.2.3 MODIFICATIONS TO EXISTING CELL DEVS MODEL

We modified the existing CELL-Devs model where only one ball moves within the walls in the pin ball game. We generated a val file which was missing in the previous version of the project as follows:

(0,5) = 9, (0,6) = 9, (0,7) = 9, (0,8) = 9,

(0,9) = 9, (0,10) = 9, (0,11) = 9, (0,12) = 9,

(0,13) = 9, (0,14) = 9, (1,4) = 9, (1,5) = 9,

(1,14) = 9, (1,15) = 9, (2,3) = 9, (2,4) = 9,

(2,15) = 9, (2,16) = 9, (3,2) = 9, (3,3) = 9,

(3,16) = 9, (3,17) = 9, (4,1) = 9, (4,2) = 9,

(4,17) = 9, (4,18) = 9, (5,0) = 9, (5,1) = 9,

(5,9) = 9, (5,10) = 9, (5,11) = 9, (5,12) = 9,

(5,17) = 9, (5,18) = 9, (6,0) = 9, (6,10) = 9,

(6,11) = 9, (6,12) = 9, (6,13) = 9, (6,18) = 9,

(7,0) = 9, (7,11) = 9, (7,12)=9, (7,13)=9,

(7,14)=9, (7,18)=9, (8,0)=9, (8,4)=9,

(8,18)=9, (9,0)=9, (9,4)=9, (9,5)=9,

(9,14)=9, (9,15)=9, (9,18)=9, (10,0)=9,

(10,5)=9, (10,14)=9, (10,15)=9, (10,16)=9,

(10,18)=9, (11,0)=9, (11,14)=9, (11,15)=9,

(11,18)=9, (12,0)=9, (12,9)=9, (12,10)=9,

(12,11)=9, (12,12)=9, (12,18)=9, (13,0)=9,

(13,8)=9, (13,19)=9, (13,10)=9, (13,11)=9,

(13,12)=9, (13,18)=9, (14,0)=9, (14,1)=9,

(14,10)=9, (14,11)=9, (14,12)=9, (14,18)=9,

(15,1)=9, (15,2)=9, (15,17)=9, (15,18)=9,

(16,2)=9,(16,3)=9 (16,16)=9 (16,17)=9

(17,3)=9 (17,4)=9 (17,8)=2 (17,9)=9

(17,10)=9 (17,14)=9 (17,15)=9 (17,16)=9

(18,4)=9 (18,5)=9 (18,6)=9 (18,7)=9

(18,8)=9 (18,9)=9 (18,10)=9(18,11)=9

(18,12)=9 (18,13)=9 (18,14)=9 (18,15)=9

The position of the ball is defined as : (7,5) = 4,

We created 2 other versions of game apart from the original version. One case is where we created 3 balls instead of only one ball and they run without hitting each other for given time of simulation. The other case is 3 balls with one exit/drain. After a giving simulation time, the one of the balls cannot sustain any more in the field and go out of the field through an exit.

3.2.4 MOORE NEIGHBOURHOOD

The ball can be moved in top, bottom, left, right and all the four diagonal directions. For this reason, considered neighborhood is the 9-cell:
pinball(-1,-1), pinball(-1,0), pinball(- 1, 1),
pinball(0,-1), pinball(0,0), pinball(0,1),
pinball(1,-1), pinball(1,0), pinball(1,1).

3.2.5 RULES DEFINED

The ball can move in any of the 8 directions from (0,0). The cube (i.e. the wall) stays in one place.
There are 11 possible types of values for a cell:
0 = There is nothing (neither ball nor wall)
1 = There is a little ball in the NE direction
2 = There is a little ball in direction N
3 = There is a little ball in the direction NW
4 = There is a little ball in the direction W
5 = There is a little ball in direction SW
6 = There is a little ball in the direction S
7 = There is a little ball in the direction SE
8 = There is a little ball in the direction E
9 = There is a wall
10 = There is an exit

The rule for the wall is:

rule: 9 100 { (0,0) = 9 }

This rules checks if a cube is present in the current cell. If so, the cube remains there.

The rule for an exit is:

rule: 0 100 { (0,0) = 1 and (-1,0) = 10 and (0,-1) = 9 and (0,1) = 9 } %exit up

This rule checks if the ball moving in the NE direction is near an exit. If so, the ball is sent out of the cell space through the exit.
This rule is applied to a ball moving in any direction and if the exit is anywhere in its neighborhood.
The above rule is added later on in order to create an exit for pinball 3D visualization that contains a drain.

A few more examples for this rule are:

rule: 0 100 { (0,0) = 2 and (1,0) = 10 and (0,-1) = 9 and (0,1) = 9 } %exit down

rule: 0 100 { (0,0) = 7 and (0,-1) = 10 and (0,-1) = 9 and (0,1) = 9 } %exit left

rule: 0 100 { (0,0) = 8 and (-1,0) = 10 and (0,-1) = 9 and (0,1) = 9 } %exit right

A few rules for rolling a ball are given in the algorithm as follows:
- If an empty cell has a ball in its neighborhood that is moving in the NEdirection (cell value = 1), then a ball in that cell will also move in the NE direction.

rule: 1 100 { (0,0) = 0 and (1,1) = 1 }

-If an empty cell has a ball in its neighborhood that is rolling in the NE direction (cell value = 1) but also there is a wall (cell value = 9) in its path, the ball will hit the wall and the switch to rolling in the west direction (cell value = 4).

rule: 4 100 { (0,0) = 0 and (0,-1) = 9 and (1,0) = 1}

Similar test cases are made for all possibilities. Therefore, there are a total of 69 rules for this cell-DEVS model.

The required files to be generated are:
-.MA file : This file includes the declaration of the environment, neighborhood and the rules for the cell-DEVS model.
-.VAL file : This file contains the initial state of the cell space. This is seen in the figure below (this is obtained as the cell-DEVS model is run in
http://www.omarhesham.com/arslab/webviewer/)

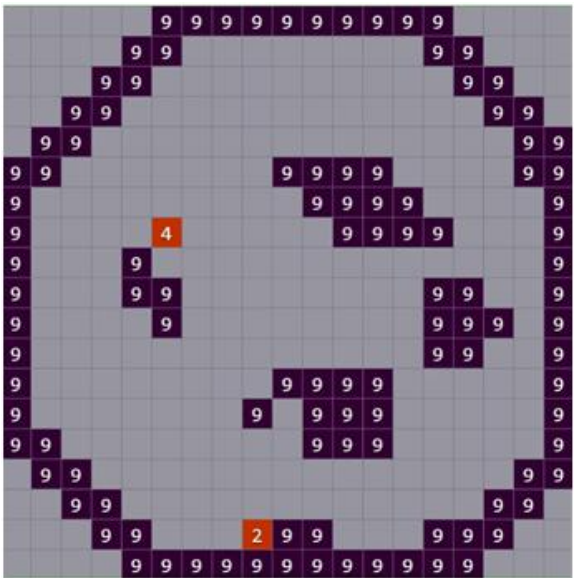

Figure 6: Pinball CELL-DEVS model Simulation

-.LOG file : This file shows how the model runs. This file is very important in order to visualize a given cell-DEVS model in blender.
-.PAL file : This is the color palate. As seen in figure 1, the colors violet for walls and red for balls are done using this .pal file.

3.2.6 Visualization of Pin Ball

In this project we used, Blender 2.6 version and python 2.4.4. Python comes with Blender as a package when we download. All the log, val and ma files generated using Cell Devs model, have to be imported into Blender tool. For this implementation, we need to develop a python script which can read the files, place the objects per the log file and help in rendering the scenes in blender. All the settings in blender are saved as .blend file which includes the initial objects required for the visualization. All these are accomplished by two python scripts. In the subsequent sections, these python scripts are explained in detail along with blender 2.6.

3.2.7 BLENDER 2.6

Blender 2.6 is the latest version of blender which we used. It is more flexible than the older versions to create animations and visualization. That is due to following unique features

1. 3D audio and speaker objects were added, along with various enhancements to the already existing sound features. It's now possible to include sound effects to animations and include audio files.

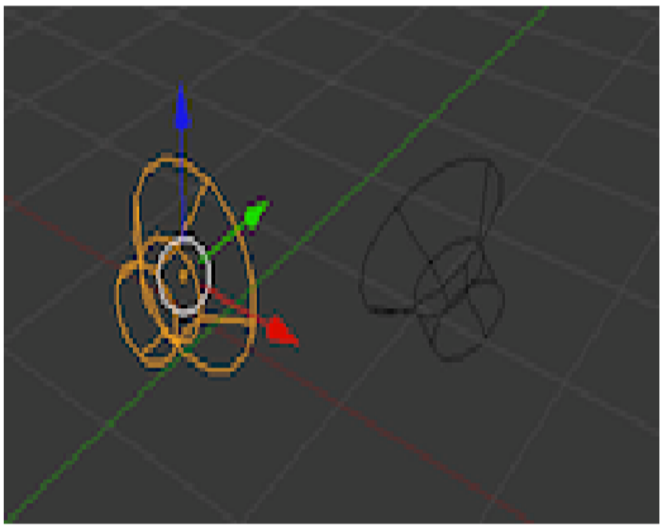

Figure 7: Audio and Video Performance of Blender 2.6

2. Animation system improvements were done, specifically related to the usability and the interface. Includes changes to the graph editor, drop sheet editor, etc.

Figure 8: Smooth Animation in Blender 2.6

Below is the screen shot of the Blender tool in figure 9.

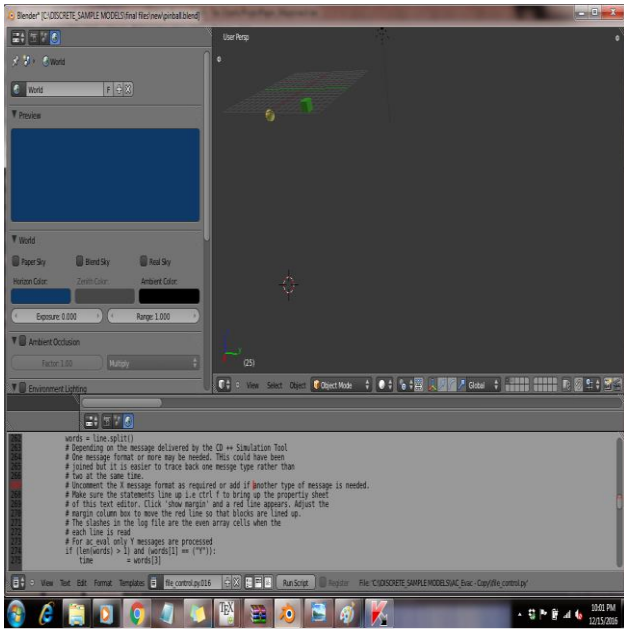

Figure 9: Blender 2.6

When the tool is opened, ctrl+O opens the .blend file from the specified directory. It is shown below in figure 10. pinball.blend is our file.

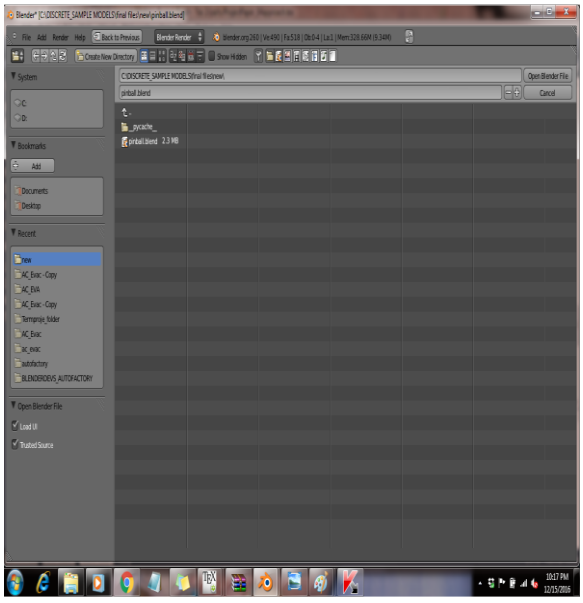

Figure 10: Opening of .blend file

Once the .blend file is loaded, the filecontrol.py script is executed as in the figure 11.

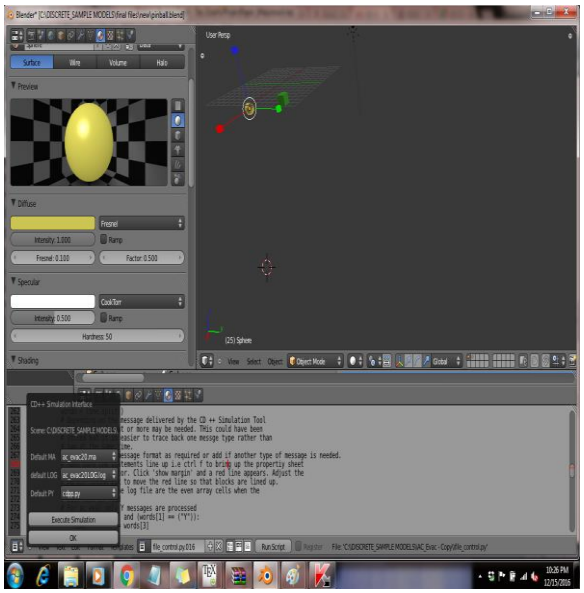

Figure 11: File Loading

3.2.8 CREATION OF BLEND FILE WITH OBJECTS

Blender has some inbuilt objects which can be pulled in when required into the scene. If we need any new objects, we can generate them using the basic objects given by default in the tool. For our project, we need a cube and a sphere. We tried to get hands-on on the tool by creating some objects using the already existing objects. For example, we designed a chair using a cube and some other animation tools. But, since we need only a cube and a sphere, we imported them from the list available. We need to click space button on the keyboard by placing the current cursor on the 3D view pane and then type in "add cube" to add a cube. The cube was re-sized by clicking 's' on the keyboard. The color of the cube can be selected from the materials tab in the pane available in blender. Similarly a ball was created from "UV Sphere" available in the default objects provided. Size and color of the ball object was modified for our requirement as mentioned above. Then the initial scene was rotated for a better view and saved the settings in a .blend file. Hence when a blend file is opened for visualization, the objects created with the default settings are opened.

3.2.9 PYTHON SCRIPT

Python script is needed to interface CD++ with Blender software. It converts the ma, val and log files generated by CD++ into a language blender can understand. We already have a filecontrol script from previous projects, where as we designed a new cdpp file for our project. The brief explanation of these scripts are explained in the subsequent sections.

3.2.9.1 Filecontrol script

Primary motive of filecontrol script is to allow the blender tool to read the files required from the directory where the .blend file resides. This scrip provides a dynamic import of the files from blender. This file will create an reference to the class defined in the selectedfile called MyClass. If the class is not present or renamed, this file will fail to execute and generate errors. The file selection 'drop down' or combo box for the MA, VAL, and LOG files are dynamically generated and are regenerated each time the script is run or when the script is moved to another directory. The valid ma, val and log files should be saved in the same directory where the blend file is there, so that the visualization is successful in Blender. A snapshot of reading the variables in script as below:

```
def initSceneProperties(defaultPath, ma_file, log_file, py_file):
    bpy.types.Scene.MyMaString = bpy.props.StringProperty(name="String")
    bpy.context.scene['MyMaString'] = defaultPath+ma_file
    bpy.types.Scene.MyLogString = bpy.props.StringProperty(name="String")
    bpy.context.scene['MyLogString'] = defaultPath+log_file
    bpy.types.Scene.MyPyString = bpy.props.StringProperty(name="String")
    bpy.context.scene['MyPyString'] = defaultPath+py_file
    bpy.types.Scene.defaulthPath = bpy.props.StringProperty(name="String")
    bpy.context.scene['defaultPath'] = defaultPath
    return
# ------------------------------------------------------------------------
```

Figure 12: Code Snippet from filecontrol.py

3.2.9.2 CDPP File

cdpp script is designed for our application. The script reads the val, log and ma file. val file is not mandatory as the initialization can be done in the ma file itself. The previous project's code has been used to implement the code for our application. The log file is read word by word, and checks the cell value at the given time instance as shown in the figure below

```
pattern = '%H:%M:%S'
epoch =  time.strptime(myTime, pattern)
totalTime = int((epoch[3] * 60 + epoch[4]) * 60.0 + epoch[5])

bpy.context.scene.frame_set(totalTime)


scn = bpy.data.scenes["Scene"]

# Get cell description currently works for three dimensions
# cells are always described as 'modelName'(x,y)(..)
# zeroize position counters to be more versatile than previous version
a = b = c = d = 0
for i in range(len(cell)):
    if cell[i]=='(' and cell[i-1]!=')':
        a = i
    elif cell[i]==',' and b==0:
        b = i
    elif cell[i]==')' and c==0:
        c = i


#Transform description into coordinates
xcoord = (cell[a+1:(a+len(range(a,b)))])
ycoord = (cell[b+1:(b+len(range(b,c)))])
```

Figure 13: Snippet of code reading coordinates

The 3D code was modified to 2D by defining the logic in the given snapshot above. The 2D coordinate is like (2,3). A for loop is used to detect the braces and store the coordinates in a temporary variable. Once the coordinate is found, the log value is again stored in another temporary variable. For our project, since it is a pin ball game, we need some boundaries and a ball. To build a boundary, a cube is designed and a ball is designed. Once an output 9 is seen in the val file, a cube is placed at the given cell address. When it starts reading the log file, if a log value 2 to 8 is seen, a ball is placed, else if a 0 is found it is left empty. And if a 4 is seen, an empty space in the wall is made. The snippet of code for this shown the figure below

```
try:
    # Process state values that are passed as logValue for the current cell
    if ((logValue == 1) or (logValue == 2) or (logValue == 3) or (logValue == 4) or (logValue == 5) or (logVal
        # Occupied cell (ball)

        ob = returnObjectByName(objectname)
        if not ob:
            myClasslog.info("Cell was not present and was linked in for display")
            #myClasslog.info("cell value is", logValue)
            myClasslog.info("Sphere not found in scene ... adding to scene")
            activeObject = SelectAndDuplicate('Sphere', objectname)
            self.linkGrpObjs(objectname,self.group.name)
            myClasslog.info("Sphere set to xy coord "+ xcoord+ycoord)
            activeObject.location.xy = [int(xcoord), int(ycoord)]

            bpy.ops.object.select_all(action='DESELECT')
            myClasslog.info("Object Sphere created ")

    elif (logValue == 9): # Wall or obstacle
        try:

            activeObject = SelectAndDuplicate('Cube', objectname)
            self.linkGrpObjs(objectname,self.group.name)
            activeObject.location.xy = [int(xcoord), int(ycoord)]

            myClasslog.info("Object cube created")
        except ValueError:

            myClasslog.info("Object cube already exist or error occured**\n")

    elif (logValue == 0): # Empty cell
        try:
            #works great now to remove it
```

Figure 14: Code Snippet for placing objects

3.3 RESULTS

We first took the simple pinball which is already existing, and did the required modifications to missing data like .valfile which was mentioned in the previous section. The simplepin ball simulation looks like below figure.

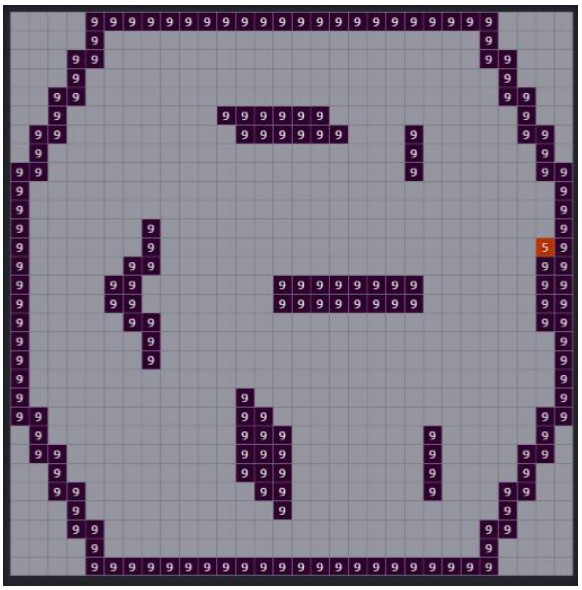

Figure 15: Screenshot of simple pin ball simulation

Then we added 3 balls to make it little more complicated and did the simulation to find the result as in the below figure

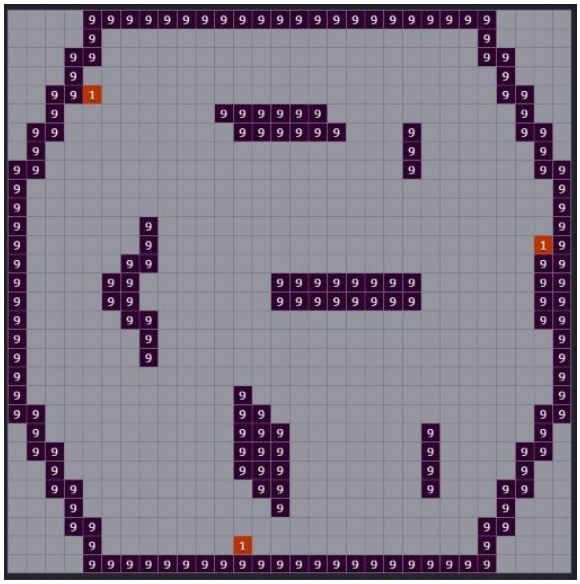

Figure 16: Screenshot of 3 balls in pin ball simulation

Once the 3 ball pin ball game is done, we tried a new one to see if a ball can go out of the court after a while. We created an exit at one place and watched to check if the ball goes it out. After a given simulation time, one of the balls goes out.

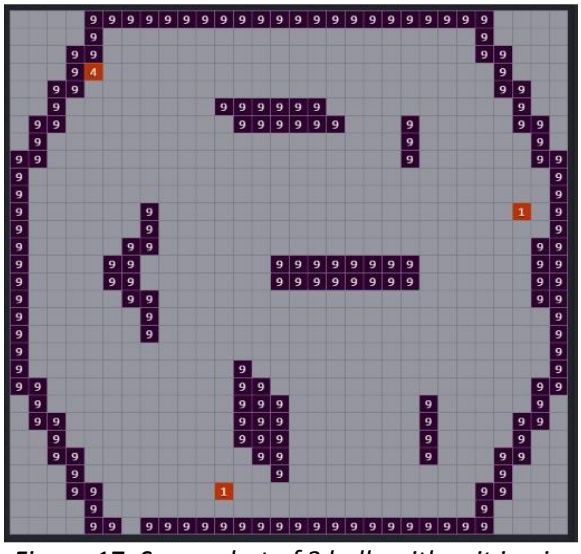

Figure 17: Screenshot of 3 balls with exit in pin ball game

After we have all the ma, val and log file in all three different cases, we proceeded with blender tool. Blender needs a blend file where all the objects are saved. Hence the blend file was designed with a cube, ball and a plane in a project folder where the ma, log and val files are present. We tried to create a glowing cube and a sphere

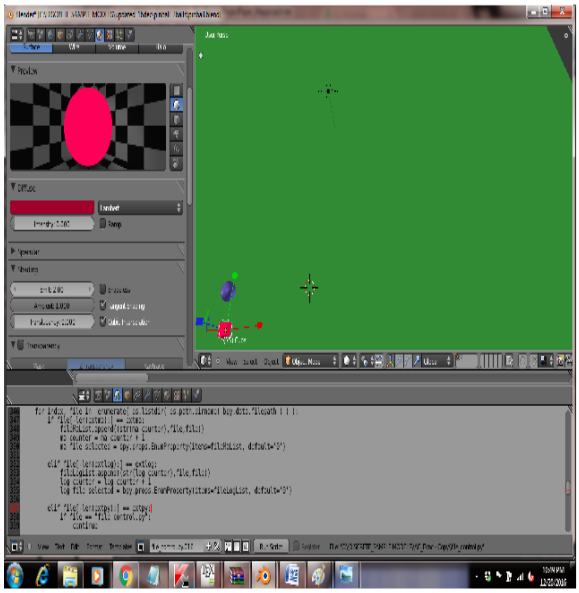

Figure 18: Screenshot of making an object glow1

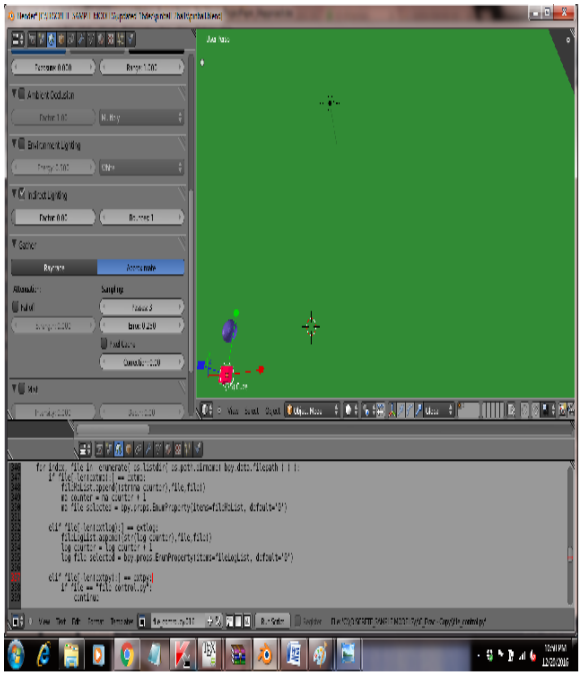

Figure 19: Screenshot of making an object glow2

Once the blend file is opened, we just need to click the execute button on the screen in blender 2.6. In the figures shown, 3D visualization of all two scenarios below:

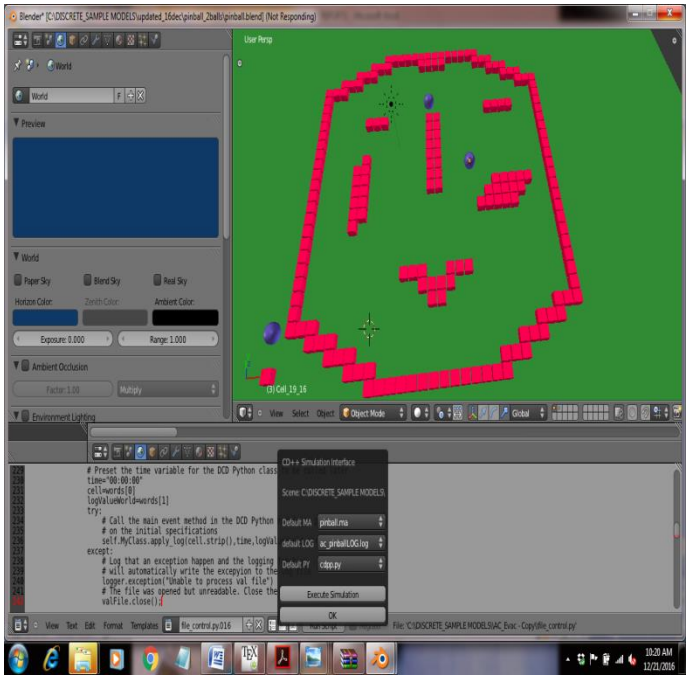

Figure 20: Picture of a 3 ball pin ball game

In the figure 21, the screenshot shows an exit in the wall on the right side. That is an empty cell created in the val file with a value 10. When a python script sees a 10, there is a an empty cell created in the wall. Then the rules are created in ma file to help the ball exit from the wall after certain time of duration.

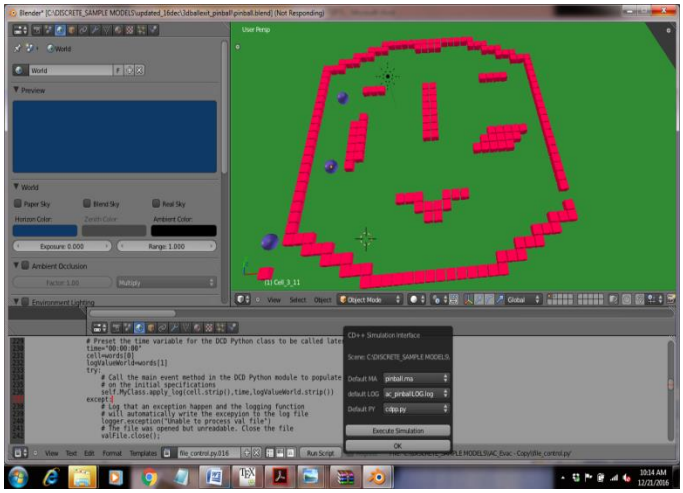

Figure 21: Picture of a 3 ball with exit in a pin ball game

4. VALIDATION

We validated our design by modifying the existing CELL-Devs model with different test cases and tried them in Blender with various ma, val and log files. And all are working perfect without any fault. All the results are explained in the previous sections.

5. CONCLUSION

Cell Devs models can be visualized in 3D using blender. Visualization of any cell devs models needs a log file to place the objects in the defined cell coordinate. A val file is needed to place the objects during initialization. Hence a blender is an ideal tool to develop any 3D models for Cell devs and devs models.

6. REFERENCES

[1] Wainer, G. and Giambiasi, N. "Application of the Cell-
DEVS Paradigm for Cell Spaces Modeling and Simulation".
Simulation, vol. 71, No. 1, pp. 22-39, January 2001.

[2] Wainer, G. “Discrete-Event Modeling and Simulation: a
Practitioner’s approach”. Taylor and Francis. 2009.

[3] Wainer, G. "CD++: a toolkit to define discrete-event
models". Software, Practice and Experience. 32(3), 1261-1306.
November 2002.

[4] Wainer, G. and Liu, Q. "Tools for Graphical Specification
and Visualization of DEVS Models". Accepted for publication
in *Simulation, Transactions of the SCS*. 2009.

[5] Blender Foundation, http://www.blender.org/

[6] Wainer, G., Poliakov, E., Hayes, J. and Jemtrud, M. "A Busy
Day at the SAT Building". Proceedings of the International
Modeling and Simulation Multiconference, Buenos Aires. 2007.

[8] Amos, M. & Woods, A. "Effect of Door Delay on Aircraft
Evacuation time".
http://arxiv.org/abs/cs/0509050. 2005.

[9] Blender (software). Wikipedia, the free encyclopedia.
http://en.wikipedia.org/wiki/Blender_(software). (Accessed 14 Dec 2011)

[10] Blender 3D: Blending Into Python/2.5 quickstart.Wikibooks.org.
http://en.wikibooks.org/wiki/Blender_3D:_Blending_Into_Python/2.5_quickstart. (Accessed 14 Dec 2011)

[11] Doc:2.6/Manual/Introduction. Blender Foundation.
http://wiki.blender.org/index.php/Doc:2.6/Manual/Introduction. (Accessed 14 Dec 2011)

[12] Gotcha’s. Blender v2.59.0 - API documentation. Blender Foundation.
http://www.blender.org/documentation/blender_python_api_2_59_0/info_gotcha.html.
(Accessed 14 Dec 2011)

[13] Keyframing and Ipo Curves. Doc:2.4/Manual/Animation/Editors/Ipo/Curves. Blender Foundation.
http://wiki.blender.org/index.php/Doc:2.4/Manual/Animation/Editors/Ipo/Curves. (Accessed 14 Dec 2011)

[14] Bézier curve. Wikipedia, the free encyclopedia.
http://en.wikipedia.org/wiki/B%C3%A9zier_curve. (Accessed 14 Dec 2011)

[15] Plugins Upload [#28802] Breakpoint. Blender Foundation.
http://projects.blender.org/tracker/index.php?func=detail&aid=28802&group_id=153&atid=472.
(Accessed 14 Dec 2011)