3D Visualization of a CD++ Cell-DEVS Football Stadium Simulation Using Blender

Jipson Johnson (101028751)
Dept.of Systems and Computer Engineering
Carleton University
jipsonjohnson@cmail.carleton.ca

Antony Anty Kannampilly (101053630)
Dept.of Systems and Computer Engineering
Carleton University
antonyantykannampill@cmail.carleton.ca

Abstract-

Cell-DEVS is an expansion of the DEVS formalism which consolidates DEVS with the formalism for Cellular Automata. It is especially helpful for characterizing spaces by deteriorating them into individual cells. The CD++ Toolkit empowers one to model and simulate a genuine or manufactured framework utilizing either DEVS or Cell-DEVS. It contains a Modeler device that grants just 2D visualization of the simulation. This specific paper centers upon a push to create a 3D representation of a current CD++ Cell-DEVS reproduction utilizing the open source 3D perception programming called Blender. The simulation scenario utilized was one that included the visualization of a football stadium and 4 robots securing the stadium. Stadium is under attack by ISIS and bombs are planted in the ground. The individual robots were compelled to move about the stadium with a specific end goal to filter the ground to identify the appearance or nonappearance of bomb. This paper talks about the conceptual design and employment processes, issues experienced alongside their determination.

Keywords: Cell-DEVS, CD++ Toolkit, Cell-DEVS visualization, Blender, football ground stadium

I. INTRODUCTION

A definitive objective of this work was to take the yield from a CD++ Cell-DEVS simulation and utilize it to create a 3D visualization. The approach that was picked included the utilization of the [freely available] and open source 3D content creation suite [called Blender]

This paper will examine in detail the means taken to accomplish this objective, from beginning acquaintance with past work and the apparatuses required, through applied plan, lastly to usage. Issues experienced and areas for future work will likewise be examined. The scenario for the simulation is robots searching for bombs in a football stadium.

In June 2016 France was attacked by ISIS, and there was scattered bomb explosion all over France. And at that time Euro 2016 was held at France. And here in our case the final match is going on between France and Portugal in Stad de France. And ISIS have planted bombs in the ground, so in order to secure the stadium and audiences we are trying to search the bomb and diffuse it using Robots. This have been simulated in Cell-DEVS and implemented in Blender for a 3D view of the model.

II. BACKGROUND

A. Discrete Event Systems (DEVS) Specification formalism

The Discrete Event systems Specification (DEVS) formalism gives a strong scientific displaying establishment and it is based upon Systems Theory ideas. While applying the DEVS formalism to the demonstrating of a genuine or counterfeit framework, one can break down it into atomic and coupled segments, or models. A atomic model might be thought to be the most fundamental of building block with which to speak to a framework. Particular states are characterized for each atomic model, and the model can exist in just a single state at a given point in time. A coupled model is made out of a few atomic or coupled models. The DEVS formalism is utilized to characterize every model and their various leveled interconnections [2]

B. Cell-DEVS

Cell-DEVS is an augmentation of the DEVS formalism which consolidates DEVS with the formalism for Cellular Automata (CA). Utilizing this way to deal with displaying permits an expert to characterize a cell space that is comprised of individual cells. Every cell is characterized as a atomic model and the technique for the atomic of these cells is additionally characterized. [4]

A cell's conduct is characterized utilizing the data inputs (N) from its neighbors (which are characterized by a specific kind of neighborhood). It is these inputs of info that actuate a cell's neighborhood computing function (τ). The yields from a cell are controlled by the kind of delay (d). Figure 1 gives a portrayal of this.

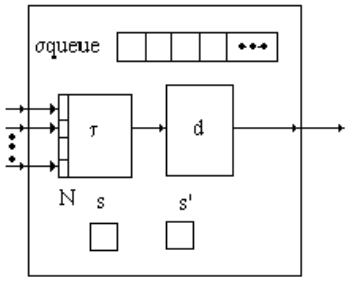

Figure 1. Cell-DEVS atomic model with transport delay

A coupled Cell-DEVS model is the resulting array of cells (atomic models) with a given dimensions, border, and zones (if applicable).

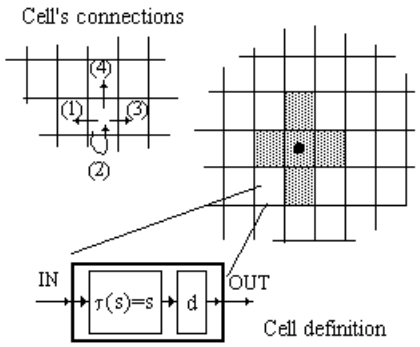

Figure 2. A Cell-DEVS coupled model

The key parameters that are used to define a Cell-DEVS model are discussed below:

Neighbourhood: A cell's neighborhood is comprised of the neighbors whose inputs of info the cell will get. The most broadly utilized neighborhoods from CA include: Moore (incorporates the 8 adjoining cells and the starting point cell); Von Neumann (incorporates cells toward the North, South, East and West and the inception cell); expanded Von Neumann (a 5x5 rhombus with birthplace in the focal cell); hexagonal topology; and triangular topology. [4] Figure 3 portrays three of these areas.

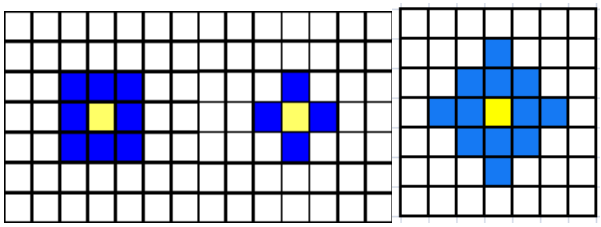

Figure 3. Moore, Von Neumann and Extended Von Neumann neighbourhoods

Local computing function (τ): The capacity used to register the future condition of the cell based upon the contributions from the neighbourhood.

Delay (d): A delay can relate to every cell. It is this delay characterizes the time at which state changes are transmitted to neighboring cells. The delay can be either inertial or transport. Transport delay characterizes state changes that happen upon the close of the delay. Inertial delay presents the idea of pre-emption. The last arrived future occasion can be pre-empted if there is another contribution before the utilization of the inertial delay [4].

Dimensions: The cell space might be two dimensional or incorporate extra layers that give a more prominent level of detail to the model.

Border: A cell space can have a characterized border or no border. At the point when there is no outskirt, this characterizes a model where all cells have a similar conduct (ie. the cells are said to be wrapped, or depicted another route: cells in one outskirt are associated with those in the inverse one utilizing the area relationship). On the off chance that a fringe is characterized then the phone space is not wrapped and the related one of a kind zones should likewise be characterized.

Zones: These are utilized to characterize ranges of the cell space with remarkable conduct. For instance, if the Cell-DEVS model represents to a room, the outskirt would not be wrapped and zones would be determined to characterize the one of a kind conduct of dividers, entryways, spaces, obstacles, etc.

C. CD++ Toolkit

The CD++ Toolkit is a modeling and simulation tool that takes into consideration the development and simulation of models in both DEVS and Cell-DEVS. In the case of DEVS, atomic models are modified in C++ and are joined into an essential class chain of command. Models can then be coupled and built into the simulation engine by utilizing specification language.

At the point when utilizing a Cell-DEVS approach, the toolkit likewise incorporates a specifications language for characterizing models based upon the formal specification. Cell-DEVS models are thought to be an exceptional instance of coupled models. Keeping in mind the end goal to appropriately describe the total cell space, the parameters talked about in area A must be characterized [6]. The behaviour of the neighborhood computing function (τ) is caught in an arrangement of principles that take after a given format:

POSTCONDITION DELAY (PRECONDITION)

This implies when the precondition is fulfilled, the condition of the cell will change to the characterized postcondition taking after the predetermined delay time. The best possible requesting of the standards is significant. On the off chance that the precondition is observed to be False, then the following standard in the rundown will be assessed. On the off chance that no control in the rundown is observed to be True, an error will be produced to recognize that the specification is not finished.

Figure 4 provides a case of the specification languages used to characterize models utilizing the CD++ Toolkit for Cell-DEVS. This specific case was created for the simulation of football ground searching that will be depicted in detail in the following area.

```
[top]
components : isisRobot

[isisRobot]
type : cell
dim : (10,10,2)
delay : transport
defaultDelayTime : 100
border : nowrapped
%Extended Von Neumann neighbourhood.
% Layer 0 - Location of robots and the map of the minefield that they generate (100 = ro
neighbors : isisRobot(-1,0,0) isisRobot(-1,1,0) isisRobot(-2,0,0) isisRobot(0,1,0) isisR
% Layer 1 - Actual disposition of mines (30 = no mine; 40 = mine).
neighbors : isisRobot(-1,0,1) isisRobot(-1,1,1) isisRobot(-2,0,1) isisRobot(0,1,1) isisR
initialvalue : 0
initialCellsValue : IsisFootballGround.val
localtransition : isisRobot10-rule
zone : ULcorner-rule { (0,0,0)..(0,0,1) }
zone : URcorner-rule { (0,9,0)..(0,9,1) }
zone : BLcorner-rule { (9,0,0)..(9,0,1) }
zone : BRcorner-rule { (9,9,0)..(9,9,1) }
zone : top-rule { (0,1,0)..(0,8,0) (0,1,1)..(0,8,1) }
zone : bottom-rule { (9,1,0)..(9,8,0) (9,1,1)..(9,8,1) }
zone : left-rule { (1,0,0)..(8,0,0) (1,0,1)..(8,0,1) }
```

Figure 4. Sample Cell-DEVS specification

Following defined parameters are of from Figure 4
Neighbourhood: Characterized by indicating each of the neighbors in connection to the cell of inception. Every cell (x1,i,...,xn,i) represents a displacement from the middle cell (0,...,0), where - x signifies both up and left, and +x indicates down and ideal, as appeared in Figure 5.

		(-2,0)		
	(-1,-1)	(-1,0)	(-1,1)	
(0,-2)	(0,-1)	(0,0)	(0,1)	(0,2)
	(1,-1)	(1,0)	(1,1)	
		(2,0)		

Figure 5. Sample definition of neighbours

Local computing function: indicated through the meaning of the principles.

Delay: transport.

Dimensions: 10x10 with two layers.

Border: is not wrapped, so exceptional zones have been indicated.

Zones: have been utilized to determine the remarkable conduct for the four outskirts of the stadium and each of its four corners. Every zone represents a cell, or gathering of cells, with its own arrangement of rules.

initialCellsValue: a .val CD++ record is utilized to characterize the begin values for each of the cells in the cell space.

C.1. CD++ Toolkit File and Message Types

CD++ Toolkit records are of 3 types these are: the .ma file; the .val file; and the .log file. The fundamental parameters that characterize the cell space are contained in the .ma record (Figure 4 is a depiction taken from the .ma document from). Next, initial values for the cells may be defined in several ways within the toolkit: using a default initial value (initialvalue); creating a file that contains a list of initial values for the cells (initialCellsValue); or creating a file that contains a map of values that will be used as the initial state for the cellular model. The underlying cell qualities were unequivocally characterized inside a.val record (Figure 6).

```
(7,6,0) = 20
(7,7,0) = 20
(7,8,0) = 20
(7,9,0) = 20

(8,0,0) = 20
(8,1,0) = 20
(8,2,0) = 20
(8,3,0) = 20
(8,4,0) = 20
(8,5,0) = 20
(8,6,0) = 20
(8,7,0) = 20
(8,8,0) = 20
(8,9,0) = 20

(9,0,0) = 120
(9,1,0) = 20
(9,2,0) = 20
(9,3,0) = 20
(9,4,0) = 20
(9,5,0) = 20
(9,6,0) = 20
(9,7,0) = 20
(9,8,0) = 20
(9,9,0) = 120
```

Figure 6. .val file

Finally, the .log file contains the resulting messages passed during the simulation of the model(s). There are several types of these messages, including:

- Message type I – Initialization message
- Message type * – Internal message
- Message type X – External message
- Message type Y – Output message
- Message type D – Done message

It is the Y messages that are of particular import to the discussion related to visualization, as they provide the time, cell coordinates, and new state values with which to depict the results of the simulation. The complete format for the Y messages is as follows:

Message Y/time/cell/port/new state value

Figure 7 provides a snapshot of the .log file.

```
Mensaje D / 00:06:18:600 / isisrobot(9,6,0)(195) / ... para isisrobot(02)
Mensaje D / 00:06:18:600 / isisrobot(02) / 00:00:00:200 para top(01)
Mensaje D / 00:06:18:600 / top(01) / 00:00:00:200 para Root(00)
Mensaje * / 00:06:18:800 / Root(00) para top(01)
Mensaje * / 00:06:18:800 / top(01) para isisrobot(02)
Mensaje * / 00:06:18:800 / isisrobot(02) para isisrobot(9,8,0)(199)
Mensaje Y / 00:06:18:800 / isisrobot(9,8,0)(199) / out /    304.00000 para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(9,8,0)(199) / ... para isisrobot(02)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(7,8,0)(159)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(8,7,0)(177)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(8,8,0)(179)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(8,9,0)(181)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(9,6,0)(195)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(9,7,0)(197)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(9,8,0)(199)
Mensaje X / 00:06:18:800 / isisrobot(02) / neighborchange /    304.00000 para isisrobot(9,9,0)(201)
Mensaje D / 00:06:18:800 / isisrobot(7,8,0)(159) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(8,7,0)(177) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(8,8,0)(179) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(8,9,0)(181) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(9,6,0)(195) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(9,7,0)(197) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(9,8,0)(199) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(9,9,0)(201) / ... para isisrobot(02)
Mensaje D / 00:06:18:800 / isisrobot(02) / ... para top(01)
Mensaje D / 00:06:18:800 / top(01) / ... para Root(00)
```

Figure 7. .log file

III. CELL-DEVS FOOTBALL GROUND SIMULATION

The cell space displayed in the Cell-DEVS simulation of the simulation represent the football ground. ISIS have planted bombs in different parts of the gound. Every cell speaks to a sensible zone of ground that a robot could be relied upon to check for a bomb. The stadium cell space has two layers. The main (layer 1) contains the real manner of the bombs. The state values for this layer don't change during the simulation. The second (layer 0) contains the guide of the stadium produced by the robots moving about the stadium. The state values for this layer change as the robots move about the stadium and sweep the ground for bombs. Table 1 delineates the state values that are characterized for this simulation and the colour palette that was utilized for the CD++ Cell-DEVS visualization.

In this simulation, robots are compelled to move just to their North, East, South and West. For crash evading, the robots look to the two cells to their North, East, South, and West keeping in mind the end goal to

maintain a strategic distance from crashes with different robots. The neighbourhood may therefore be considered as a variation on the extended Von Neumann neighbourhood (which is meant to be a 5x5 rhombus).

Table 1. State values and colour palette for CD++ Cell-DEVS simulation

Definition	State	Colour
LAYER 1 – FOOTBALL GROUND OUTLOOK		
No bomb	30	Red
Cell with bomb	40	Blue
LAYER 0 – FOOTBALL GROUND MAP		
Cell without mapping	20	Dk Grey
Cell without mapping with robot	120	Grey
Cell after mapping – without bomb	0	Blue
Cell after mapping – with bomb	10	Red
Cell after mapping – without bomb, robot	100	Lt Blue
Cell after mapping – with bomb, robot	110	Pink
Cell after mapping – without bomb, robot moving North	201	Med Blue
Cell after mapping – without bomb, robot moving East	202	Med Blue
Cell after mapping – without bomb, robot moving South	203	Med Blue
Cell after mapping – without bomb, robot moving West	204	Med Blue
Cell after mapping – without bomb, robot moving North	211	Med Red
Cell after mapping – without bomb, robot moving East	212	Med Red
Cell after mapping – without bomb, robot moving South	213	Med Red
Cell after mapping – without bomb, robot moving West	214	Med Red

The variation on the extended Von Neumann neighbourhood is shown in Figure 9.

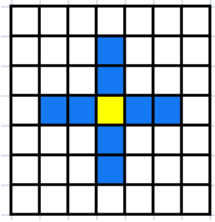

Figure 9. Variation on Extended Von Neumann Neighbourhood

The football ground cell space was intended to have an border that obliges the movement of the robots to that space as it were. In this manner (as appeared in Figure 4), in CD++ the border is characterized as:

```
border : nowrapped
```

Keeping in mind the end goal to show the suitable conduct of robots that are obliged to move just inside the characterized stadium cell space, unique zones were characterized. Zones were characterized for the stadium borders furthermore for each of the four corners. Each zone has its own unique local transition function rules defined in order to properly model the behaviour expected from the cells within those zones. The definition of the zones is depicted in Figure 4.

Keeping in mind the end goal to start the simulation with a characterized arrangement of robots and the real manner of the bombs inside the stadium, a .val record was used. This record was utilized to list the underlying state values for every cell inside the cell space (both layers 0 and 1). A specimen of the .val file for is appeared in Figure 6.

To show that the model appropriately catches the sought of desired behaviour, the accompanying three figures delineate screen captures from the CD++ Modeler at various circumstances all through the simulation. The first figure (Figure 10) demonstrates that the underlying situation of the robots has been completed as per the .val file (ie. that a robot was set in each of the four corners). Layer 0 is appeared at left and layer 1 is appeared at right.

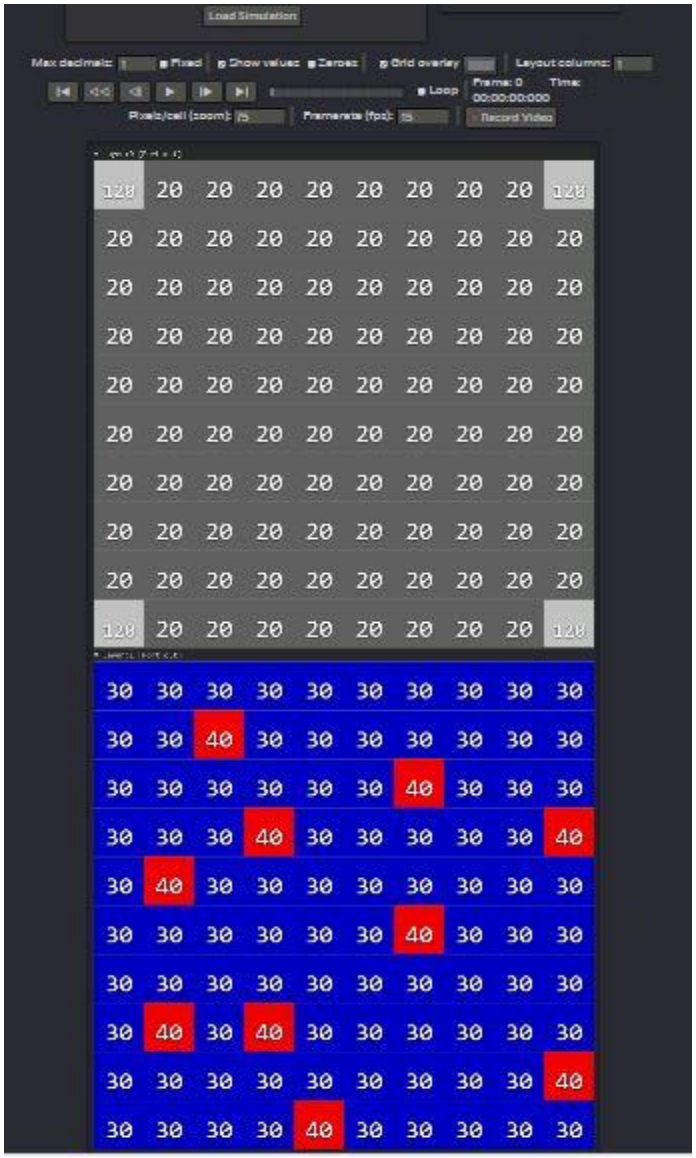

Figure 10. Initial stage of the Football Ground simulation

The second figure (Figure 11) shows that the robots are precisely mapping the stadium and apprising the stadium map (layer 0). Note that the location of the bombs in the stadium (layer 0 at left) accurately reflects their actual location within the stadium (layer 1 at right).

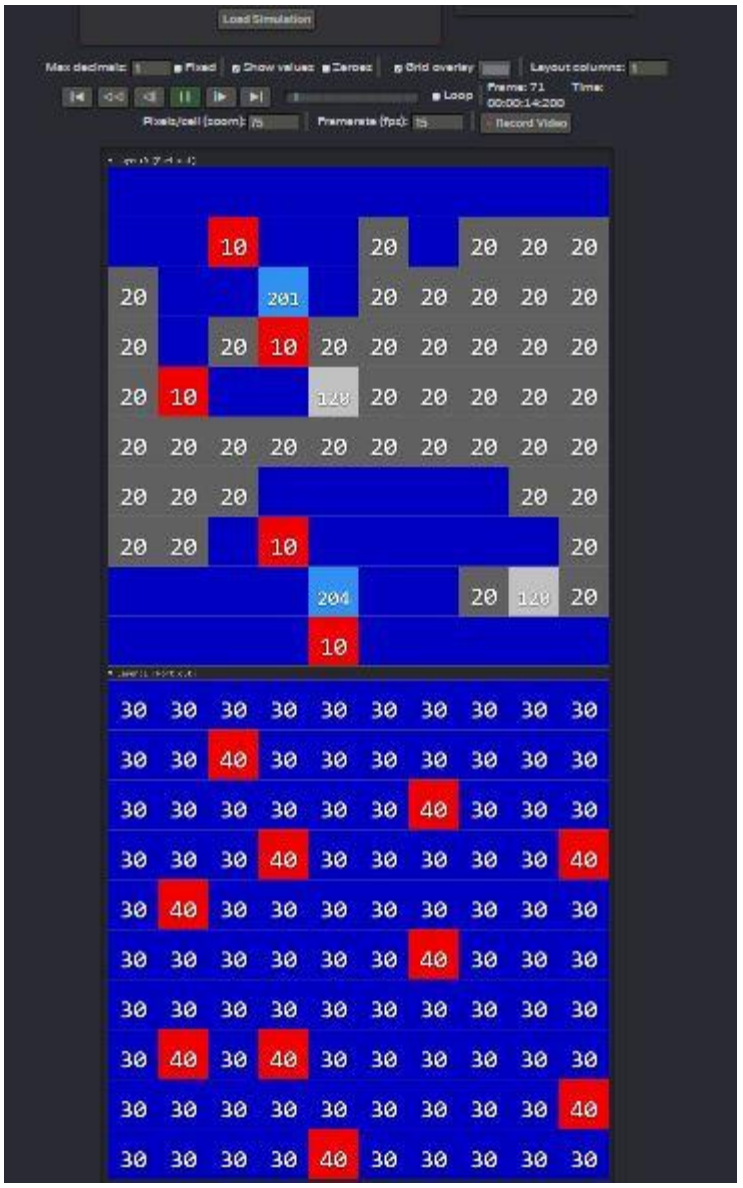

Figure 11. Intermediate stage of Football Ground simulation

The last figure (Figure 12) shows that the robots were effective in completing and accurately mapping the stadium.

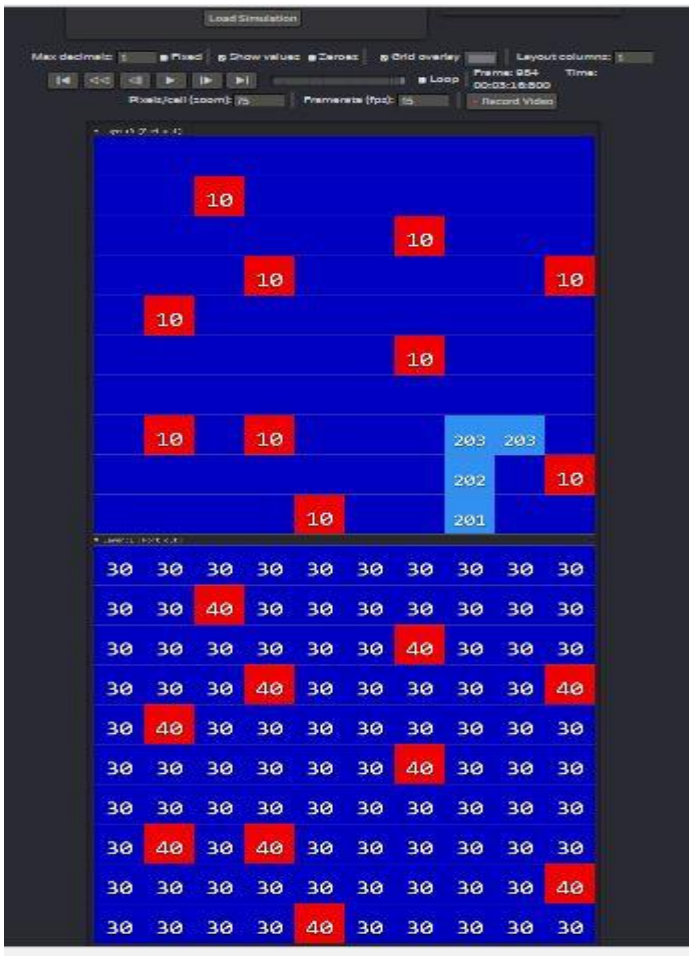

Figure 12. Final stage of Football Ground simulation

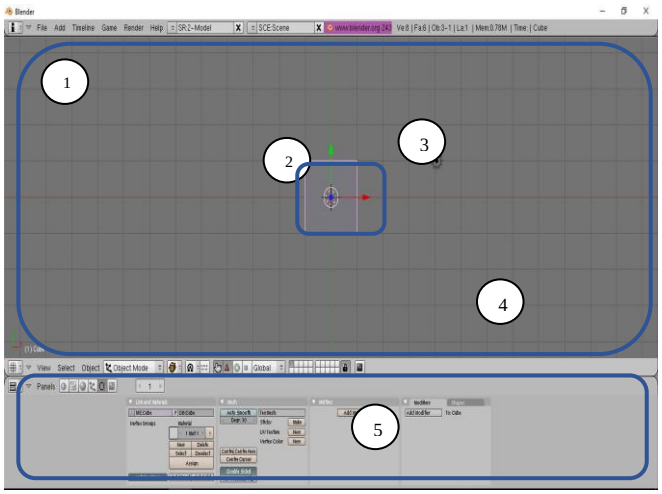

Figure 13. Blender interface

IV. 3D TOOLS – PYTHON AND BLENDERS

Blender is depicted as "an incorporated application that empowers the formation of an expansive scope of 2D and 3D content. It gives an expansive range of displaying, finishing, lighting, liveliness and video post-preparing usefulness in one package." It has a fairly one of a kind User Interface and depends vigorously upon console easy routes connected alone or in conjunction with the mouse. Figure 13 gives a preview of the Blender interface at initial stage. This screencap demonstrates the accompanying key parts of the interface: the 3D View (1); the 3D square that is available at the initial stage (2); the lamp that is available at initial stage (3); the camera that is available at initial stage (4); and the Buttons Window (5).

As we were new to Blender, we found that there was a somewhat soak expectation to absorb information connected with acquainting our self with how to utilize Blender and how to saddle the greater part of its components. Inside the time limitations connected with this specific venture, I was unquestionably not ready to wind up distinctly a completely useful Blender client. The accompanying segment will talk about a portion of the issues connected with this reality.

The Blender site depicts Python as a universally useful scripting language. A regularly growing Blender Python API [14] exists that gives access to Blender's inward capacities so as to influence them furthermore to broaden Blender's usefulness. The prescribed adaptation of Python is regularly included and introduced with the appropriation of Blender. Since Python was implanted in Blender, access to the Blender Python API modules must be made by running the scripts in Blender. You [cannot] import the Blender module into an outside Python translator. [9][10]

Inside Blender, a scene is made to contain the greater part of the items that will shape part of that scene. A solitary Blender record can contain numerous scenes, since it is sorted out and set up to have the capacity to contain a whole motion picture. [11] Essentially a scene contains numerous articles that may then be sorted out into layers for simplicity of administration [11]. The sorts of articles that can be available inside a scene are compelled just by the creative energy and expertise of the scene's architect. Every object is comprised of various faces, vertices and edges, all of which can be controlled utilizing Blender. The scene will likewise incorporate lamps to give lighting to the scene. Lamps are required when a creator

means to quicken the scene, if no lamp(s) are available then the rendered scene will be dark. Cameras additionally shape part of the scene. These can be set all through the scene to film it from different edges, and it is from a camera's vantage point that a liveliness is shot. The accompanying figure (Figure 14) delineates the Outliner View inside Blender, which gives a rundown of those items that are found inside the scene for the re-enactment from.

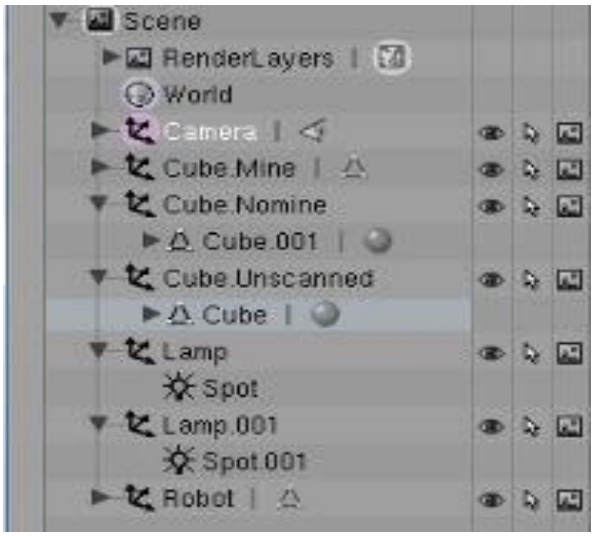

Figure 14. Scene for Football Ground visualization

There are a few modes in which is used to work inside Blender. Edit and Object modes where the important modes that are used here. The faces of the items must be chosen in Edit Mode. So, keeping in mind the end goal to apply diverse materials to a object it was important to be in Edit Mode. In Object Mode, articles could be moved, pivoted or scaled. Edit Mode was essential for performing more nitty gritty changes to the geometry and materials of object.

Just in our acclimation with Blender, we start to peruse about an ability called interpolation (IPO). We never did completely build up a comprehension of the power that insertion could loan to the perception of our CD++ Cell-DEVS recreation. If we had known more about the force of insertion, we think this would have a huge effect upon our theoretical outline. We found that IPO is the way toward assessing a object's position (or different traits) based upon a known begin and end points, and the time between the begin and end [12] and that it is utilized for the liveliness of a scene.

Animation software for the most part uses three techniques to make 3D objects move, these are: key frames; movements and paths. The Blender IPO framework fuses the initial two, and either can be connected to objects keeping in mind the end goal to enliven them. There are a few sorts of addition inside Blender, including constraints, object, sequence, material, texture, world and shape. Inside every IPO there are diverse sorts of channels against which introduction (movements or key frames) can be connected. For instance, inside the Material IPO type, there are channels like R, G, B, and texR, texG, texB. So as to apply insertion, one should first choose a channel and after that apply movements or key frames to it. Insertion can be connected to more than one object (i.e. one IPO can enliven a many objects).

Commands to make new scenes, objects, lights, cameras, and so on is contained in Python scripts, or they can be utilized to just copy objects that as of now exist in a Blender scene. They can likewise contain commands to collaborate IPO motions or key frames with a object or multiple objects.

V. INSPIRATION FROM EARLIER WORK

Blender was picked as the tool to create the 3D representation of the CD++ Cell-DEVS football ground simulation. This decision was overwhelmingly based upon the way that the work that we done to make the assignment 1, made us think that making a stadium and giving it a 3D view would be a marvelous enhancement. So we thought of implementing the model in Cell-DEVS and convert it in to 3D.

With a specific end goal to complete 3D representation of a CD++ simulation, a Python script is required. The first Python script, by Poliakov, empowered Blender to translate a CD++ Cell-DEVS simulation [14]. This script was just intended to peruse and picture the .log document from a CD++ Cell-DEVS reproduction.

The first Python script was essentially enhanced by Castonguay in [13]. His adaptation of the script included two new and critical capacities. The first of these was the capacity to peruse the .ma file and look for the presence of a .val file. The script then read in the underlying cell state values from the .val file and gave a 3D perception of these with a specific end goal to portray the simulation begin state. The second ability that was included was that of an information log file for investigating. The Python script made an information log content document that was utilized to catch key moves that were made, including: the perusing of the .ma file; the perusing and perception of the .val file (if relevant); and the perusing and representation

of the .log file. This capacity was especially useful for troubleshooting the script in our specific case.

Keeping in mind the end goal to acquaint our self with Blender and the current Python script, furthermore to influence the previously mentioned extra usefulness, we depended upon the aircraft evacuation simulation and Python script created by Castonguay [13]. In his work, he made a scene inside Blender that officially contained three objects: a human shape; a 3D cube that is similar to a seat; and a 3D cube similar to a plane exit. The Python script then read the suitable .val and .log files for the visualization of the system. So as to construct the representation, the Python script was organized to make Blender copy the objects that as of now existed in the scene.

At the start of our perception exertion, we didn't know that the Python script could likewise be utilized to teach Blender to make fresh out of the new scenes and objects. Therefore, the approach that we took unequivocally reflects that of Castonguay [13]. We in the long run made a scene inside Blender that officially contained the model of a football stadium with floodlights and scoreboard. And this was made using the different functions that Blender offers such as cube, plane, tube etc. The initial step four objects in my underlying Blender scene towards this project was a football stadium and it is appeared in the accompanying figure:

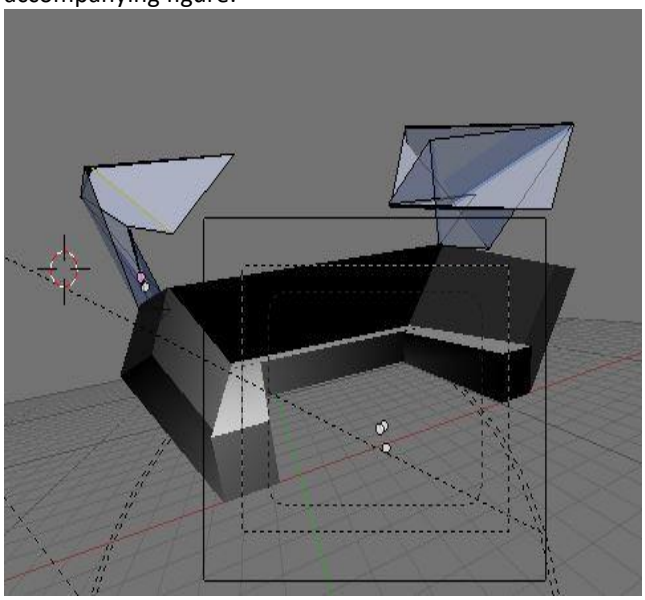

Figure 15. Initial Blender scene for the Football Ground simulation

We knew about the troubles experienced by Castonguay in legitimately scaling an open source-acquired seat object in the scene. Thus, I decided to just utilize the human frame that both he and Poliakov used keeping in mind the end goal to portray my robots. Also the football stadium that we made in the first stage was not looking as original as a football stadium. So we had other ideas in mind. The time imperatives connected with finishing this venture did not allow us to investigate the issues identified with outlining our own particular robot utilizing Blender or appropriately scaling an alternate open source instantiation of a robot inside our scene.

VI. BLENDER AND ITS INFLUENCE ON THE VISUALIZATION CONCEPTUAL DESIGN

As beforehand specified, we confronted a precarious expectation to absorb information in acquainting our self with Blender. If we have been more well known the program, the calculated outline for the perception of the CD++ Cell-DEVS simulation would likely have been exceptional adjusted to amplify Blender's abilities. Rather, we were compelled to all the while figure out how to utilize and saddle Blender's components and to build up our applied outline. It was difficult to totally acclimate our self with the extensive variety of elements and usefulness gave by Blender in the time distributed for this venture.

At the beginning of the project we thought of using the blender functions to make the stadium our own, and we made it as it is seen in the Figure 15. Since it does not look originally as a stadium we made use of open source stadium properties and built our own football ground stadium. This stadium is visualized using the blender functions such as tubes, cubes, spheres etc and connecting those together to look like a stadium with stands.

The accompanying two figures portray the different views of the football stadium in Blender 2.43.

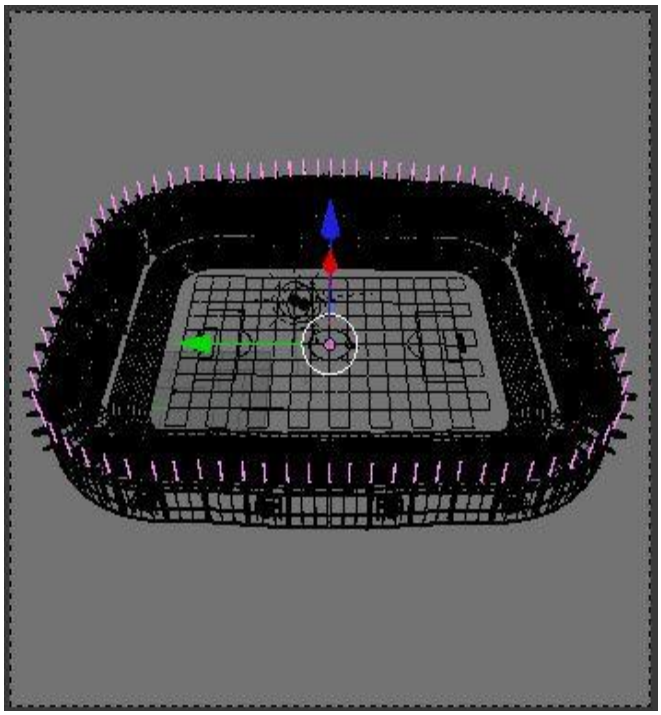

Figure 16. View of our Football Stadium in Blender 2.43

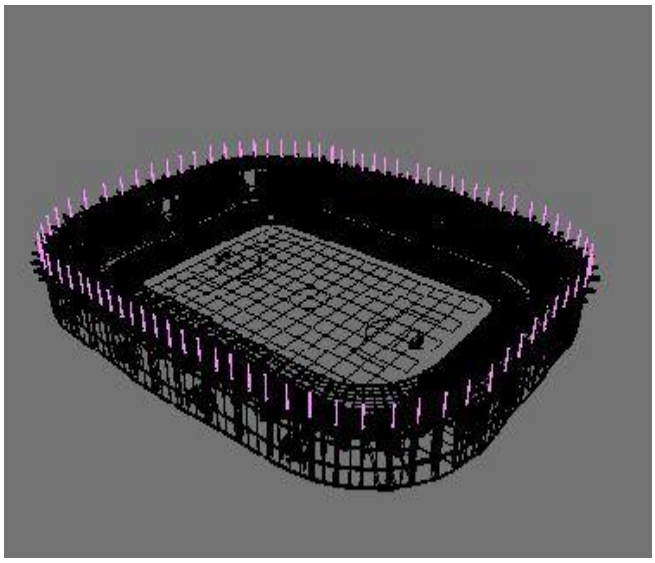

Figure 17. Another view of our Football Stadium in Blender 2.43

Also this football ground can be viewed better in the newer version of Blender such as 2.78 and above, with the greenery of the ground, the stands etc.

VII. 3D VISUALIZATION USING BLENDER

The important structure pieces that is used to accomplish the visualization incorporated: the CD++ Cell-DEVS .ma, .val, .log files from the effective 4 robot simulations; the Python script; and the Blender .blend document with the robot, unscanned cell, no bomb cell, and bomb cell. These will each be examined in this area.

A. CD++ Cell-DEVS Files

The .ma, .val, and .log files from CD++ provided the data that was required to visualize the simulation of four robots scanning a football ground. The .ma file provided the name of the .val file. Initial state values for each of the cells in both layers of the cell space is listed by .val file. The Y messages from which the cell coordinates, state values and time were extracted is provided by the .log file.

B. .blend File

The .blend file was planned to contain the football ground stadium, the ground areas with bomb, and the ground areas without bomb. The observation was created by duplicating these objects according to the cell coordinates, state values and times isolated from the CD++ records using the Python script.

C. .py file

Our Python script work principally utilized that of Castonguay [13]. Note that his script utilized the first work of Poliakov [14]. Python script takes after the structure of Castonguay's work with significant alterations to represent the way of my specific simulation. The script contains the accompanying key functions:

i. read_val, read_ma, and read_log
ii. apply_log
iii. import_maFile and import_logFile
iv. gui
v. buttonHandler

The first three functions are relatively self-explanatory. The read_ma function searches the .ma file for either a default initial value to assign to all of the cells, or for a .val file that contains a list of initial values by cell. In the case of the football ground simulation, this function would locate the IsisFootballGround.val file. The read_val function takes the cell coordinates and initial state values (time = 00:00:00:000) from the .val file in order to visualize

the initial state of the simulation. The read_log function then takes the cell coordinates, state values and time values from the Y messages in the .log file in order to visualize the simulation from start to finish.

Both the read_val and read_log functions call the apply_log function keeping in mind the end goal to do the visualisation. An apply_log function call is made every time another state esteem is perused in. The apply_log function does numerous things. To start with, it pulls the estimation of the hours, minutes, seconds, milliseconds from the time esteem keeping in mind the end goal to ascertain the aggregate time in milliseconds. This value is utilized to set the present frame in Blender and partner the time, state and cell values with that frame. Besides, it pulls the x, y, and z coordinates out of the cell value. It also calculate the state value (the Y message design for the state value has three decimal places, these are truncated here). Next, it makes two strings based upon the coordinates, one for the football ground cell name and one for the robot's name. Finally, the main part of the apply_log function manages the treatment of each of the state values when they are connected with a cell. In a nutshell, the accompanying moves are made for the following state values:

i. 20 (unmapped cell)
    a. Get the Cube.Unscanned
    b. Assign it a pointer (isisFootballGround)
    c. Link it to the scene, if it is not already
    d. Select it
    e. Duplicate it
    f. Make this duplicate the active object
    g. Name it according to its intended coordinates
    h. Place it according to its coordinates
    i. Deselect it
ii. 120 and time = 00:00:00:000 (robot on an unscanned cell at initial stage)
    a. Get the Robot
    b. Assign it a pointer (robot)
    c. Link it to the scene, if it is not already
    d. Select it
    e. Duplicate it
    f. Make this duplicate the active object
    g. Name it according to its intended coordinates
    h. Place it according to its coordinates
    i. Deselect it
    j. Add an unscanned cell as for state = 20
iii. 120 and time != 00:00:00:000 (robot on an unscanned cell during the simulation)
    a. Only add a new robot as per 120 and time = 00:00:00:000
    b. Do not add an unscanned cell as one already exists
iv. 0 and 10 (no bomb in cell OR bomb in cell: need to remove robot, cell already scanned and coloured accordingly)
    a. Look for robot in the scene with coordinates matching the cell
    b. Select it
    c. Make it the active object
    d. Rename it
    e. Unlink it
v. 100 and 110 (no bomb and a robot OR bomb and a robot: need to add a robot, cell already scanned and coloured accordingly)
    a. Only add a new robot as per 120 and time = 00:00:00:000
    b. Do not add a scanned cell as one already exists
vi. 301-304 (cell just scanned, no bomb: change the football ground cell to no bomb)
    a. Look for football ground cell in the scene with coordinates matching the cell
    b. Select it
    c. Make it the active object
    d. Rename it
    e. Unlink it
    f. Get Cube.Nobomb
    g. Link it to the scene, if it is not already
    h. Select it
    i. Duplicate it
    j. Make this duplicate the active object
    k. Name it according to its intended coordinates
    l. Place it according to its coordinates
    m. Deselect it
vii. 311-314 (cell just scanned, bomb: change the football ground cell to bomb)
    a. Look for football ground cell in the scene with coordinates matching the cell
    b. Select it
    c. Make it the active object
    d. Rename it

e. Unlink it
f. Get Cube.Bomb
g. Link it to the scene, if it is not already
h. Select it
i. Duplicate it
j. Make this duplicate the active object
k. Name it according to its intended coordinates
l. Place it according to its coordinates
m. Deselect it

The roles of the import_maFile and import_logFile functions are self explanatory. The gui function takes care of the display that is generated when the script entitle CD++ Simulations is selected from the Scripts menu under Misc (Figure 19) . The gui itself is shown in Figure 20.

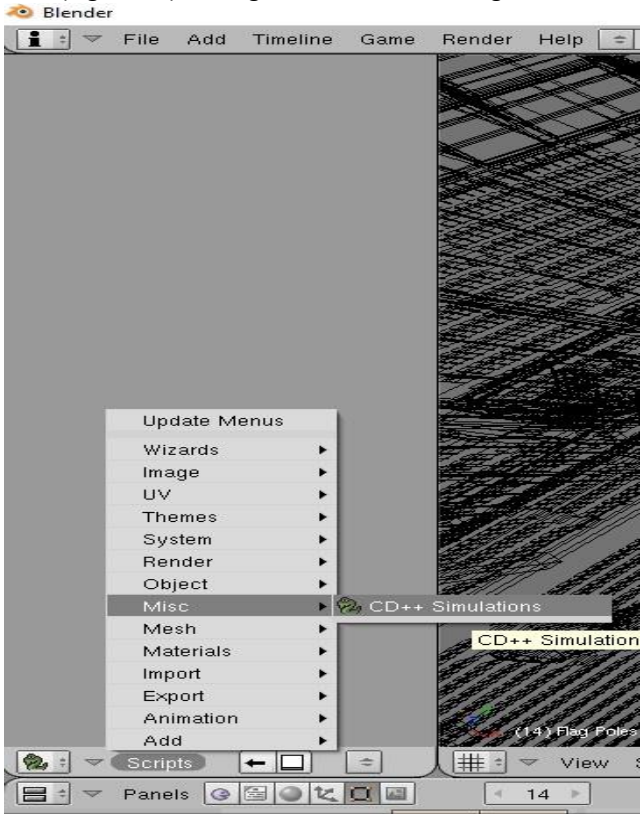

Figure 18. Selecting the CD++ Simulations script

Figure 19. CD++ Simulations script GUI

D. Setting up the visualization

The following set up procedure should be followed to properly to carry out the visualization of the C++ Cell-DEVS football ground simulation:

- Install Blender
- Install Python version 2.4
- Place **IsisFootballGround.blend** in C:\Program Files\Blender Foundation\Blender\.blender
- Place **IsisFootballGround.py** in C:\Program Files\BlenderFoundation\Blender\.blender\scripts
- Place **IsisFootballGround.ma**, **IsisFootballGround.val**, and **IsisFootballGround.log** in C:\Program Files\Blender Foundation\Blender\.blender
- Open IsisFootballGround.blend
- Load the CD++ Simulations script (see Figure 19)

- In the script GUI, select **IsisFootballGround.ma** and **IsisFootballGround.log** by browsing the .blender folder, if they are not already selected
- Hit the Execute button in the script GUI
- Watch the 3D visualization

The following figure (Figure 21) depicts the final frame of the football ground simulation visualization.

The accompanying (Figure 22) delineates the rendering of the final frame of the football ground visualisation. So as to vivify the simulation, progressive frames would need to be rendered. Since the applied outline required the duplication of objects, this keeps animation from being possible (ie. there are not genuinely four robots, but instead various duplicates of the underlying robot that are put in the scene and afterward unlinked). Interpolation is required for animation, and it is connected with particular objects. For our situation, the robots are appeared to "move" by unlinking (erasing them) from their past cell and copying them for the new cell. Thus, it is impractical to apply interpolation to a specific object accordingly in order to animate its movement. To do so would require changing the conceptual design, and as a result, the Python script.

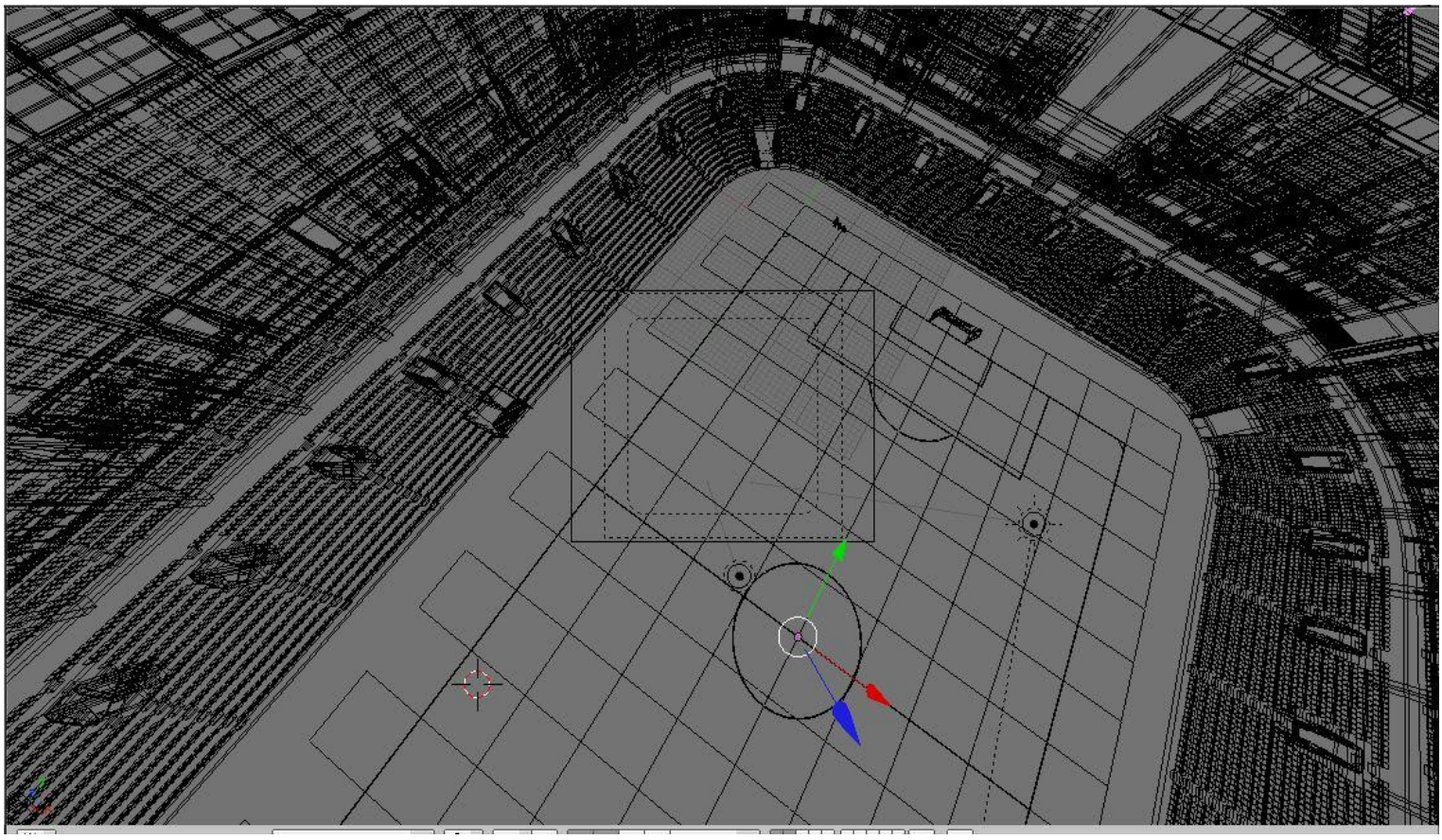

Figure 20. Final frame of the visualization of the Football Ground simulation

VIII. COMPLICATIONS CAME ACROSS

The Python .write summon was utilized to compose key data to the information log .txt document to investigate issues with the script and its conduct. I had at first expected to utilize it overwhelmingly to comprehend the conduct of the apply_log fucntion, considering the way this was the capacity that was completing the representation of the re-enactment. Shockingly, the more .write orders I embedded in the apply_log function the less the capacity would work. I at long last recognized that the .write order arrange utilized by Castonguay [13] was bringing on my Python script to make exemptions. The underlying organization utilized was as per the following:

```
datalogfile.write("Processing:"+robotName+"forlogValue:"+logVal
ue+"\n")
```

This problem was resolved by using the following format instead:

```
datalogfile.write("Processing: %s for %d\n"
%(isisFootballGroundCell,logValue))
```

Underlying absence of Python information had a negative effect upon our capacity to adjust Castonguay's code [13] to address our particular re-enactment's issues. Every time we erased areas of the script or adjusted it somehow, we would render that specific segment pointless. We then realized that this was connected with the way that our IDE was blending tabs with spaces and rendering the code indiscernible. This is on the grounds that in Python, space is utilized to gathering articulations into squares [18].

By blending tabs and spaces we were contrarily influencing the gathering of the announcements in the script.

- The determination for this issue was very straightforward: legitimately arrange the IDE that is utilized. The IDE ought to be arranged to either:Add spaces when pressing the tab key; OR
- Add tabs when pressing the tab key. The programmer should then also only use tabs for indentation. [18]

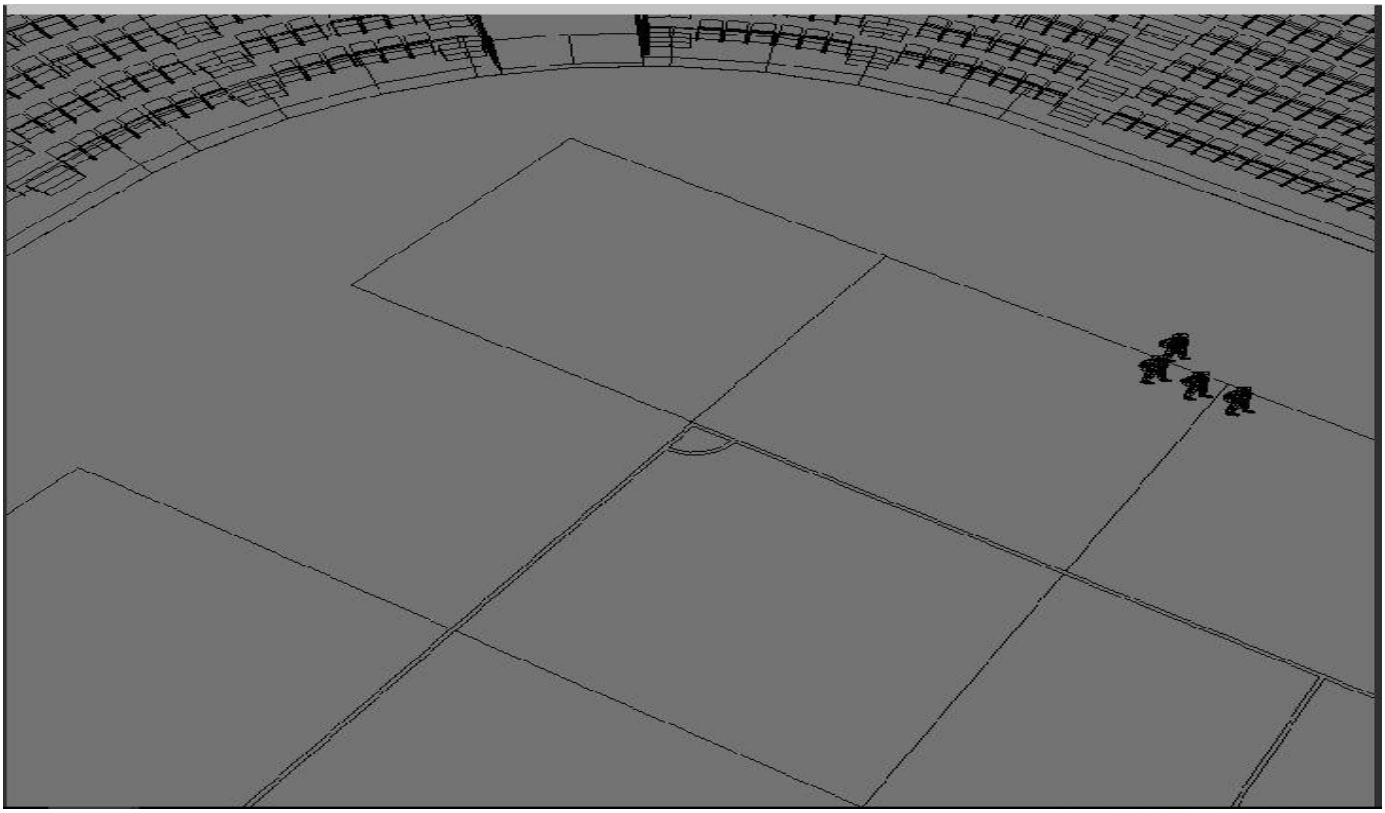

Figure 21. Another view of final frame of the Football Ground simulation

IX. FUTURE WORK

While this visualization effort was an extremely valuable experience for the us, it could serve as a springboard for future work. As we can see that simulations of the robots moving are being done in the corner of the football ground, it is not being fully covering the stadium. So keeping this in mind we can enhance the project as a future work.

The limited exposure to interpolation during this work did not enable the us to fully utilize the functionality that interpolation could provide. Two points were made clear from reading the Blender documentation:

- Interpolation is required for the animation of a scene; and
- Interpolation animates an object, or objects, by estimating [their] position based upon a known start end value, and the time between the start and end.

Primary focus could instead be upon how to apply interpolation to a scene. As a result, our recommendations are that future work should:

- Overcome the design hurdles early on;
- Shift focus to understanding and applying interpolation to the scene. Key things such as IPO types, channels, IPO curves, key frames, etc. should be understood;
- Investigate the use of IPO types, and their associated channels, in order to animate the scene;

Also this football ground stadium can be viewed more lively in the Blender 2.78. Since we have done this in Blender 2.43 it is not that too eye catching. We also tried to implement this in newer versions but the CD++ simulations was not being carried out properly. So as a future work, implementing this in latest version of Blender also can be done. Figure 22 depicts the view of the stadium in Blender 2.78.

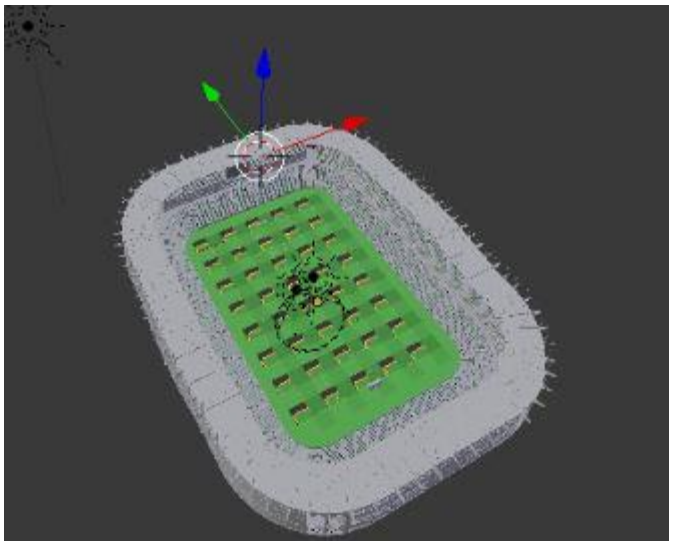

Figure 22. View in Blender 2.78

X. CONCLUON

This paper has introduced the foundation, plan and execution steps, and issues experienced and settled, as for a push to envision a CD++ Cell-DEVS re-enactment utilizing both Blender and a Python script. It has likewise given suggestions to future work. While the creator has no other 3D programming against which to analyze usefulness and usability, no doubt Blender could be a fitting representation device for CD++ Cell-DEVS simulations. The individuals who seek future work here can influence the current Python script and the lessons gained from this work to build up their own particular remarkable situations, or they can develop the present Cell-DEVS football stadium situation. The conceivable outcomes are numerous and the potential methodologies are just constrained by the creative energy and time of future designers.

XI. References

[1] Blender, "Blender," 2016, http://www.blender.org/.

[2] G. Wainer, "Introduction to the DEVS Modeling and Simulation Formalism," in *Discrete-Event Modeling and Simulation: A Practitioner's Approach (Draft)*, pp. 1-22.

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

[4] G. Wainer, "The Cell-DEVS Formalism," in *Discrete-Event Modeling and Simulation: A Practitioner's Approach (Draft)*, pp. 1-22.

[5] G. Wainer, "CD++: A Toolkit to Develop DEVS Models," *Software - Practice and Experience,* vol. 32, No. 13, pp. 1261-1306, November 2002.

[6] G. Wainer, "Introduction to the CD++ Toolkit," in *Discrete-Event Modeling and Simulation: A Practitioner's Approach (Draft)*, pp. 1-32.

[7] A. C. Morris, S. P. N. Singh, and S. M. Thayer, "Development of an Immunology-Based Multi-Robot Coordination Algorithm for Exploration and Mapping," Conference on Intelligent Robots and Systems, Presentation, October 2, 2002.

[8] S. P. N. Singh and S. M. Thayer, "ARMS: Autonomous Robots for Military Systems - A Survey of Collaborative Robotics Core Technologies and Their Military Applications," Carnegie Mellon University Robotics Institute, CMU-RI-TR-01-16, 2001.

[9] Blender, "Manual/Introduction - BlenderWiki," 2016, http://wiki.blender.org/index.php/Manual/Introduction.

[10] The Blender Python Team, "Module API_Intro: The Blender Python API Reference: Built Thu May 10 20:32:02 EST 2007," 2007, http://www.blender.org/documentation/245PythonDoc/.

[11] Blender, "Scene Management Structure," 2016, http://wiki.blender.org/index.php/Manual/Scene_Management.

[12] Blender, "Manual/Ipo Types," 2016, http://wiki.blender.org/index.php/Manual/Ipo_Types.

[13] P. Castonguay, "CD++ Simulations version 1.0.2," 2008.

[14] E. Poliakov, "CD++ Simulations version 1.0.1," 2007.

[15] Blender, "Open Material Repository," 2016, http://www.blender-materials.org/index.php.

[16] Blender, "Blender 2.43 Download," http://download.blender.org/release/Blender2.43/blender-2.43-windows.exe.

[17] Python, "Python version 2.4 - Download," http://www.python.org/ftp/python/2.4.4/python-2.4.4.msi.

[18] Python, "How to Edit Python Code," 2016, http://wiki.python.org/moin/HowToEditPythonCode.