3D Visualization of Predator Prey Cell-DEVs Model

Saima Hidayat / Leili Parishani

Department of Electrical and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON. K1S-5B6 Canada
shida046@uottawa.ca
parishani.leili@gmail.com

ABSTRACT

The project is about the 3D visualization of an existing project Predator Prey, developed by the past students. The idea is to reuse the template developed by the students and create enhanced graphical user experience by manipulating the template as per the requirement and behavior of the Predator Prey project. The project is selected, keeping in view the level of complexity for learning the 3D visualization as beginner. It is developed using the Blender version 2.60 tool with the extension of an embedded Python 3.2. The paper will discuss different aspects of the project including the model and design considerations for 3D rendering of the Cell-DEVS space and the dynamic objects.

1. Introduction

The project is based on the visualization enhancement of the Predator Prey project which is based on a Cell-DEVS model. The DEVS formalism has been used as modeling and simulation technique for different natural and artificial systems. Cell-DEVS is an extension of DEVS that allows for executing cellular automata models with the advantage of evaluating the cells asynchronously with different timing delays. Both techniques have shown success in simulating space-shaped models.

Furthermore, for the 3D visualization of the above, we have used the Blender-Python combination. The Blender is an advanced open-source three-dimensional modeling software, used to create high quality animations. In this project, the Blender is integrated with Python as a scripting language allowing modeling, texturing, rendering, lighting, and post-processing. Using a Blender, one can render the 3D model of the simulation logs captured by CD++ tool. We will be discussing the Blender-Python integration and implementation of 3D visualization of Predator Prey project in this paper.

2. Background

The basic structure of the project uses the template provided by the past students which allows the user to execute a predefined interface to map the Cell-DEVS model, value and log files dynamically. It supports the format of logs generated by the CD++ tool. The design and the simulation is performed in the Blender tool once the files are parsed properly by the python script. The initialization and other modifications can be implemented in the customized python script which executes after the file selection.

The main mapping for the Cell-DEVS model files is performed by the Python script named as file_control.py by the original authors of the file. This file is used to control the simulation and allows the user to select the appropriate CD++ generated and compiled files for the model, initialization and the logs of the simulation as a result. Using this template, the time for the file mapping can be saved since it requires minimal value changes in the scripting environment and an initial one time setup for use.

Moreover, Python is a multi-paradigm language, which allows C, C++, or Cython built-in modules to be implemented, it becomes highly extensible as a scripting language. Because of its nature of readability and ease of use, the Blender Application Programming Interface (API) allows a graphics designer or an animator to quickly implement complex animation routines or simulations through its graphical user interface and relatively simple API calls also improve timing constraints.

Although, Blender 2.5x has a completely different API calling syntax than previous versions. Also, Python similarly has evolved a lot in terms of syntax and libraries. These changes demanded the installation of latest Blender versions. It indirectly impacted the Python scripts, written using the old API. The Blender has a new API for the Python integration.

Unfortunately, since the release of Python version 3.0, Python has broken its own “backward compatibility” rule thus necessitating refactoring of developed scripts for future work and research in the DCD modeling.

In Blender, Python is used primarily for scripting, prototyping, gaming logic, importing/exporting to other rendering formats, task automation and custom tools. The Blender tool itself is composed of obfuscated key board shortcuts that lately have been simplified by the addition of comprehensive contextual panel menus in order to make the tool appear more streamlined and logical.

As an open-source tool, Blender is presently comparable to mid range commercial products, but like most open-source software, it is also criticized for having poor documentation on its API and on the examples for the logical steps to create and deploy rendered animation. [1]

Organization of data by Blender is accomplished through objects, meshes, lamps, scenes, materials where objects are composed of meshes, materials and lamps and can be supported or stored within multiple scenes. Because of this, it would seem very important to understand the API changes to select and modify the data. Through the use of the menus and hotkey selection, the objects can be selected and modified. However, the main panels only work on an activation event such as a mouse click and otherwise are unavailable for direct manipulation through the Python interpreter.

With the introduction of Blender 2.5, a new module bpy.ops was implemented. This implementation provided the programmer the ability to register their operation script and when activate provided a tooltip capability indicating the class, location and the tooltip documentation if provided. [2]

3. Predator Prey Cell-DEVS Model

The objective is to model the behavior of an animal trying to escape a space in the form of a maze, before being trapped by a predator. The labyrinth could represent an environment similar to a forest. The prey, which could be a rabbit or deer, is leaving its scent as a trail. The predator, a wolf or other carnivorous animal, moves faster than the prey. The predator tries to approach a prey by its odor or smell. If the predator reaches the prey, then it eats it up. The above behavior has been modeled using CD++ tool and DEVS formalisms. The space defined for the simulation is like a maze structure. The predator and prey are the dynamic objects around the space, having different values based on their position relative to each other. The prey has a trail of its smell, that attracts the predator towards itself.

Below is the brief description of the objective behind the game:

a. Walls

   Labyrinth of connected walls with one or more exits.

b. Prey

   Movement 1000 msec.

Time to rotate 500 msec.

The odor lasts for 4000 msec.

It dissipates every 1000 msec reaching four levels until disappearing (from strong or recent smell to weak odor).

c. Predator

Movement 800 msec.

Time to rotate 300 msec.

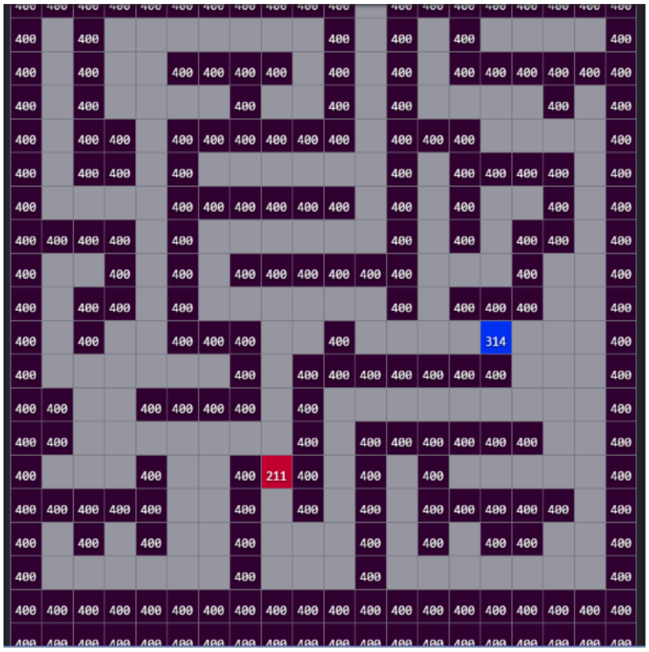


Figure 1. Labyrinth, Predator and Prey

1- Cell Space Configuration:

The model consists the following cell space configurations as given below:

Dimensions: 20 x 20 cells

Delay: inertial

Border: nowrapped

The initial Labyrinth values are defined in the predprey1.val file. The val file is used to initialize the values of the walls which creates a maze.

2- Neighborhood:

Both the prey and the predator cannot move diagonally, but to the cells on the sides, to the front or back. However, because of the motion strategy, it is required to ask for certain opportunities by the cells diagonally to the (0,0). For this reason, the neighborhood considered is that of 9 cells as stated below:

Devlab (-1, -1) devlab (-1,0) devlab (-1,1)

Devlab (0, -1) devlab (0,0) devlab (0,1)

Devlab (1, -1) devlab (1,0) devlab (1,1)

3- Rules:

The model consists of separate rules for the Labyrinth, the Predator and the Prey. The preview of the rules for the above are provided from the original project as under:

a. Labyrinth Rule

The labyrinth cells, determined in the predprey1.val file, take the value 400, with a delay of 100000, since they must not change their initial value.

Rule: {(0,0)} 100000 {(0,0) = 400}

b. Prey Rule

It was decided as a strategy to get out of the Labyrinth, so the animal travel with its right side always in contact with the wall. The rules are divided into Seeking Wall and Following Wall.

For the movement, the following were determined:

- Delay 1000 msec
- Time to rotate 500 msec
- Values of Following wall 201 to 204
- Search Wall Values 211 to 214

By convention, it is considered:

- 1 to move up
- 2 to move to the left
- 3 to move to the right
- 4 to move down

The prey leaves a trace of smell that lasts for 4000 msec. For this, the cell immediately above that was occupied by the prey, takes the value of 104 with a delay of 1000. Then reduces its value in one (from 104 to 101) until finally it disappears.

Rule: {(0,0) - 1} 1000 {(0,0)> 101 and (0,0) <105}

Rule: 0 1000 {(0,0) <= 101}

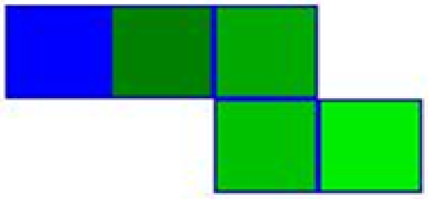


Figure 2. The trail of smell of the prey

The prey advances by trying to ensure that its right side is always in contact with the wall. You cannot advance on: wall, predator or its smell when it is recent (both levels of strong odor). It only rotates when it hits the maze or when the right is empty.

Initially, the prey advances trying to find a wall. When it collides, it turns to the left. Once this is done, it is moved to space following the wall.

Example for searching the wall, we see the case of going up. The other three cases (right, left and down) are similar.

- Go ahead and leave your trace of smell. It cannot pass if there is a wall, predator or smell at high values (104 and 103)
- Rule: 104 1000 {(0,0) = 211 and (-1,0) <103}
- Rule: 211 1000 {(1,0) = 211 and (0,0) <103 and (0,1) <103}
- If you find a wall to your right, it becomes a wall
- Rule: 201 1000 {(1,0) = 211 and (0,0) <103 and (0,1)> = 103}
- If it hits the wall, turn to the left and it turns into following wall
- Rule: 202 500 {(0,0) = 211 and (-1,0)> = 103}

Example for following the wall, we see the case of going up. The other three cases (right, left and down) are similar.

- If the cell on the right is occupied (wall, predator, strong odor), move in the direction it came and leave its smell trail.
- Rule: 104 1000 {(0,0) = 201 and (0,1)> = 103 and (-1,0) <103}
- Rule: 201 1000 {(1,0) = 201 and (0,0) <103 and (1,1)> = 103}
- If the right is empty, it rotates.
- Rule: 213 500 {(0,0) = 201 and (0,1) <103}
- If the cell on the right is occupied and also the one on the top, turn left.
- Rule: 202 500 {(0,0) = 201 and (-1,0)> = 103 and (0,1)> = 103}
- If the cells on the right, up and left are occupied, then turn down.

- Rule: 204 500 {(0,0) = 201 and (-1,0)> = 103 and (0,1)> = 103 and (0, -1)> = 103}

c. Predator Rules

The predator tries to leave the labyrinth, but if it detects the smell of prey in the next cells (up, down, right and left), it follows the prey. And if he attains it, he eats it.

For the movement, the following meanings were determined:

- Delay 800 msec
- Time to rotate 300 msec
- Values of Following Wall 301 to 304
- Search Wall Values 311 to 314

By convention, it is considered:

- 1 to move up
- 2 to move to the left
- 3 to move to the right
- 4 to move down

The predator can advance over any cell, except the wall. Similar to the prey, the predator tries to get out of the labyrinth.

- If, in the next cells (up, down, right and left), it detects the odor and follow the trail.
- Rule: 311 300 (0,0)> 300 and (0,0) <350 and (0,0)! = 311 and (-1,0)> = 100 and (-1,0)
- Rule: 300 and (0,0) <350 and (0,0)! = 313 and (0,1)> = 100 and (0,1)
- Rule: 314 300 ((0,0)> 300 and (0,0) <350 and (0,0)! = 314 and (1,0)> = 100 and (1,0)
- Rule: 312 300 ((0,0)> 300 and (0,0) <350 and (0,0)! = 312 and (0, -1)> = 100 and (0, -1)
- If predator and prey confront, the predator advances to the prey cell and makes it disappear.
- Rule: 0 800 {(-1,0)> 100 and (-1,0) <250 and ((0,0) = 311 or (0,0) = 301)}
- Rule: 311 800 {(0,0)> 100 and (0,0) <250 and ((1,0) = 311 or (1,0) = 301)}
- Rule: 0 800 {(0,1)> 100 and (0,1) <250 and ((0,0) = 313 or (0,0) = 303)}
- Rule: 313 800 {(0,0)> 100 and (0,0) <250 and ((0, -1) = 311 or (0, -1) = 301)}
- Rule: 0 800 {(1,0)> 100 and (1,0) <250 and ((0,0) = 314 or (0,0) = 304)}
- Rule: 311 800 {(0,0)> 100 and (0,0) <250 and ((-1,0) = 314 or (-1,0) = 304)}
- Rule: 0 800 {(0, -1)> 100 and (0, -1) <250 and ((0,0) = 312 or (0,0) = 302)}
- Rule: 311 800 {(0,0)> 100 and (0,0) <250 and ((0,1) = 312 or (0,1) = 302)}

The above description of the model gives a background knowledge of the project which is necessary for an individual to understand the working of the Cell-DEVS model so that the python script required for the 3D visualization of the simulation can be coded. The predpreycdpp.py is the python script file for the logic of the rules and initial values implemented in the model above.

4. File Control Interface

We have reused the interface provided by the students as a project. The overview of the script to select and map the Cell-DEVS files is given below as per the original paper [3]:

The File Control I/O module provides the following capabilities:

d. File I/O control separation from the DCD rendering requirement;

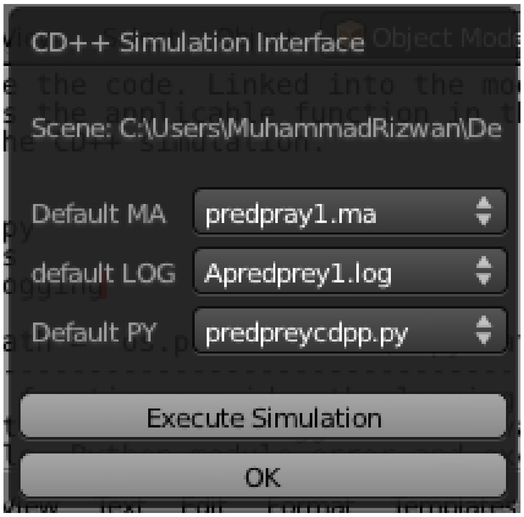
e. Menu Generation at the location of the mouse;

f. File enumeration and selection capabilities of the current blend file directory;

g. Logging and exception handling of the I/O;

h. Execution control of the DCD Python module; and

i. User simulation and execution display notification. [3]



Scene: C:\Users\MuhammadRizwan\De

Default MA: predpray1.ma

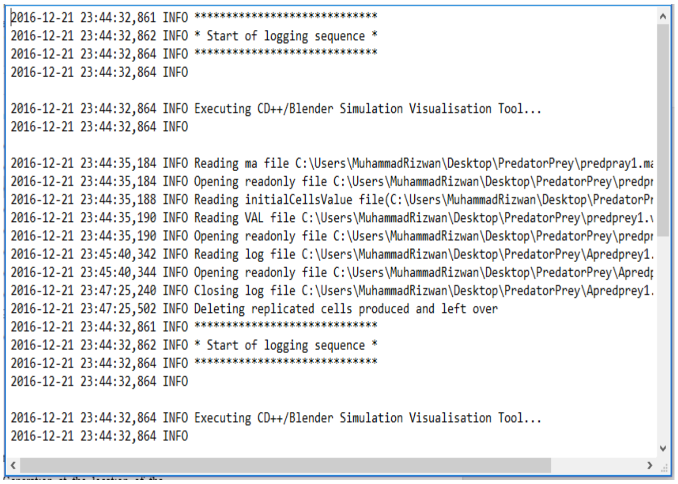
default LOG: Apredprey1.log

Default PY: predpreycdpp.py

Figure 3. File Control I/O

The Python scripts for file IO is based on the C language open file function that required a file pointer to the file to be opened. Failure of the function either through the file being open already or a non-existent file would generate an exception and was not handled by the Python interface. It is believed that exception handling should be handled at all times, regardless of the exception handling expense, and especially for non-standardized applications or scripts. [3]

It also creates the logs for the instantiation of the simulation through File Control python script.
The preview for the same is given under:



```
2016-12-21 23:44:32,861 INFO *****************************
2016-12-21 23:44:32,862 INFO * Start of logging sequence *
2016-12-21 23:44:32,864 INFO *****************************
2016-12-21 23:44:32,864 INFO

2016-12-21 23:44:32,864 INFO Executing CD++/Blender Simulation Visualisation Tool...
2016-12-21 23:44:32,864 INFO

2016-12-21 23:44:35,184 INFO Reading ma file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\predpray1.m
2016-12-21 23:44:35,184 INFO Opening readonly file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\predpr
2016-12-21 23:44:35,188 INFO Reading initialCellsValue file(C:\Users\MuhammadRizwan\Desktop\PredatorPr
2016-12-21 23:44:35,190 INFO Reading VAL file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\predprey1.
2016-12-21 23:44:35,190 INFO Opening readonly file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\predpr
2016-12-21 23:45:40,342 INFO Reading log file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\Apredprey1.
2016-12-21 23:45:40,344 INFO Opening readonly file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\Apred
2016-12-21 23:47:25,240 INFO Closing log file C:\Users\MuhammadRizwan\Desktop\PredatorPrey\Apredprey1.
2016-12-21 23:47:25,502 INFO Deleting replicated cells produced and left over
2016-12-21 23:44:32,861 INFO *****************************
2016-12-21 23:44:32,862 INFO * Start of logging sequence *
2016-12-21 23:44:32,864 INFO *****************************
2016-12-21 23:44:32,864 INFO

2016-12-21 23:44:32,864 INFO Executing CD++/Blender Simulation Visualisation Tool...
2016-12-21 23:44:32,864 INFO
```

Figure 4. Logging sequence for the File Control module

5. Predator Prey - Python Module

The script for game initialization and simulation is selected in the File Control module. The File Control script automatically takes the first encountered python file in the same folder and directory path. It auto-selects the file in the file browser. Although if any other file is present, it will show them in the dropdown as well if any change is required before the execution of the script. The customized script for the Predator Prey project has been added to the same path as File Control script. This file helps the Blender to parse the logs generated from the CD++ tool. It also takes into account the parsing of the model and val files.

Furthermore, it has all the basic functions required for the linking and delinking of the object. This script is responsible for the mapping of the Blender objects to the logs of the simulation.

Below is a brief description of all the functions in the predpreycdpp.py file:

a. class MyClass(),
    1. __init__(self, cell="", myTime="", logValueWord=""),
    2. convertStr(self, myString),
    3. createGrp(self, grpName),
    4. deleteGrp(self, grpName),
    5. linkGrpObjs(self, name, grpName),
    6. unlinkGrpObjs(self, name, grpName),
    7. deleteObj(self, name),
    8. deleteReplication(self, name),
    9. apply_log(self, cell="", myTime"", logValueWorld="", destModel="", port="")

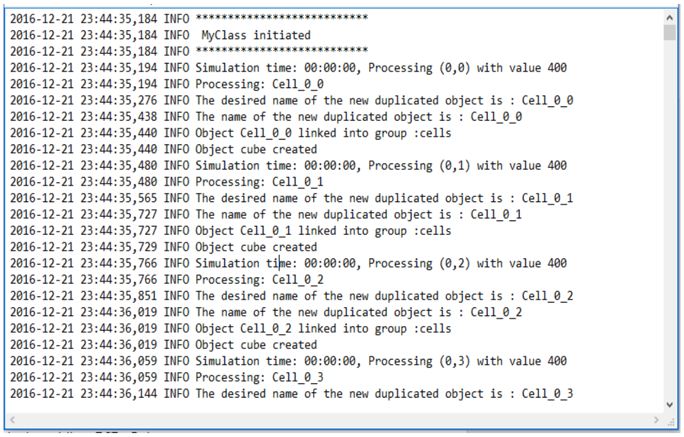
b. CreateLogger(),

c. returnObjectByName(passedName = ""), and

d. SelectAndDuplicate(cell, name).

The logs are captured for the python script errors or instances created in the Blender scene through a log file generation method. All the activities of the simulation are logged in the MyClass.log file.

A preview of the log file is given below:

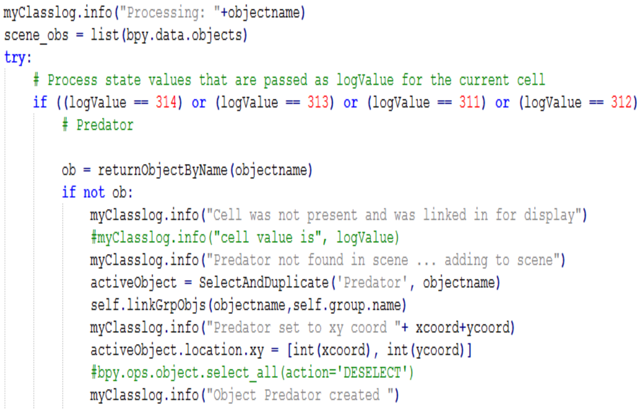


```
2016-12-21 23:44:35,184 INFO ***************************
2016-12-21 23:44:35,184 INFO  MyClass initiated
2016-12-21 23:44:35,184 INFO ***************************
2016-12-21 23:44:35,194 INFO Simulation time: 00:00:00, Processing (0,0) with value 400
2016-12-21 23:44:35,194 INFO Processing: Cell_0_0
2016-12-21 23:44:35,276 INFO The desired name of the new duplicated object is : Cell_0_0
2016-12-21 23:44:35,438 INFO The name of the new duplicated object is : Cell_0_0
2016-12-21 23:44:35,440 INFO Object Cell_0_0 linked into group :cells
2016-12-21 23:44:35,440 INFO Object cube created
2016-12-21 23:44:35,480 INFO Simulation time: 00:00:00, Processing (0,1) with value 400
2016-12-21 23:44:35,480 INFO Processing: Cell_0_1
2016-12-21 23:44:35,565 INFO The desired name of the new duplicated object is : Cell_0_1
2016-12-21 23:44:35,727 INFO The name of the new duplicated object is : Cell_0_1
2016-12-21 23:44:35,727 INFO Object Cell_0_1 linked into group :cells
2016-12-21 23:44:35,729 INFO Object cube created
2016-12-21 23:44:35,766 INFO Simulation time: 00:00:00, Processing (0,2) with value 400
2016-12-21 23:44:35,766 INFO Processing: Cell_0_2
2016-12-21 23:44:35,851 INFO The desired name of the new duplicated object is : Cell_0_2
2016-12-21 23:44:36,019 INFO The name of the new duplicated object is : Cell_0_2
2016-12-21 23:44:36,019 INFO Object Cell_0_2 linked into group :cells
2016-12-21 23:44:36,019 INFO Object cube created
2016-12-21 23:44:36,059 INFO Simulation time: 00:00:00, Processing (0,3) with value 400
2016-12-21 23:44:36,059 INFO Processing: Cell_0_3
2016-12-21 23:44:36,144 INFO The desired name of the new duplicated object is : Cell_0_3
```

Figure 5. Logging sequence for the predpreycdpp.py file

The apply_log() function is the main method for the mapping of the simulation logs. The objects are mapped as under:

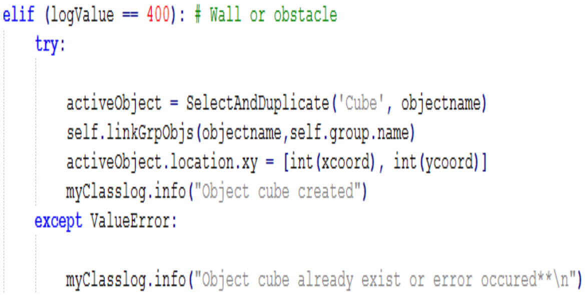
1- Predator



```
myClasslog.info("Processing: "+objectname)
scene_obs = list(bpy.data.objects)
try:
    # Process state values that are passed as logValue for the current cell
    if ((logValue == 314) or (logValue == 313) or (logValue == 311) or (logValue == 312)
        # Predator

        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("Predator not found in scene ... adding to scene")
            activeObject = SelectAndDuplicate('Predator', objectname)
            self.linkGrpObjs(objectname,self.group.name)
            myClasslog.info("Predator set to xy coord "+ xcoord+ycoord)
            activeObject.location.xy = [int(xcoord), int(ycoord)]
            #bpy.ops.object.select_all(action='DESELECT')
            myClasslog.info("Object Predator created ")
```

Figure 6. The mapping for the Predator values

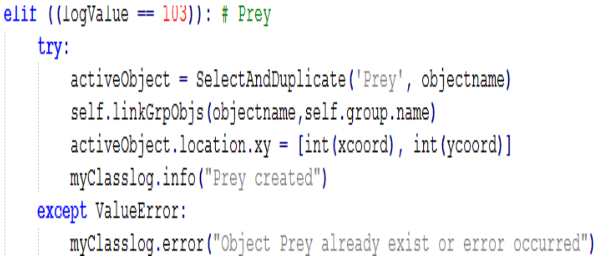
2- Wall



```
elif (logValue == 400): # 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")
```

Figure 7. The mapping of the Wall values

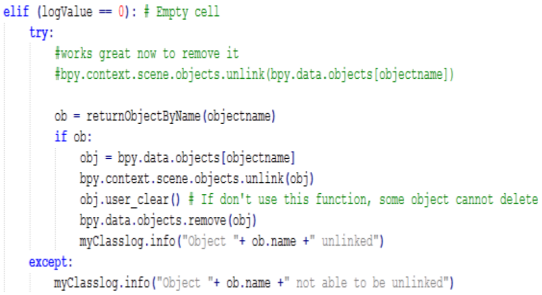
3- Prey



```
elif ((logValue == 103)): # Prey
    try:
        activeObject = SelectAndDuplicate('Prey', objectname)
        self.linkGrpObjs(objectname,self.group.name)
        activeObject.location.xy = [int(xcoord), int(ycoord)]
        myClasslog.info("Prey created")
    except ValueError:
        myClasslog.error("Object Prey already exist or error occurred")
```

Figure 8. The mapping for Prey values

4- Delinking Objects



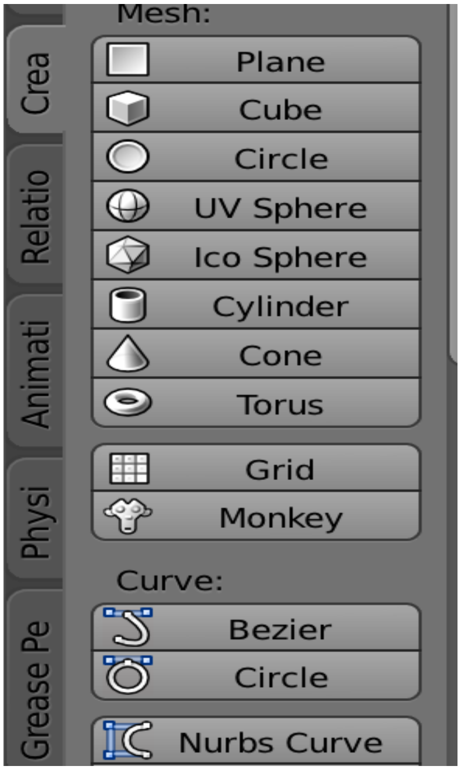
```
elif (logValue == 0): # Empty cell
    try:
        #works great now to remove it
        #bpy.context.scene.objects.unlink(bpy.data.objects[objectname])

        ob = returnObjectByName(objectname)
        if ob:
            obj = bpy.data.objects[objectname]
            bpy.context.scene.objects.unlink(obj)
            obj.user_clear() # If don't use this function, some object cannot delete
            bpy.data.objects.remove(obj)
            myClasslog.info("Object "+ ob.name +" unlinked")
    except:
        myClasslog.info("Object "+ ob.name +" not able to be unlinked")
```

Figure 9. The delinking of the objects from the scene

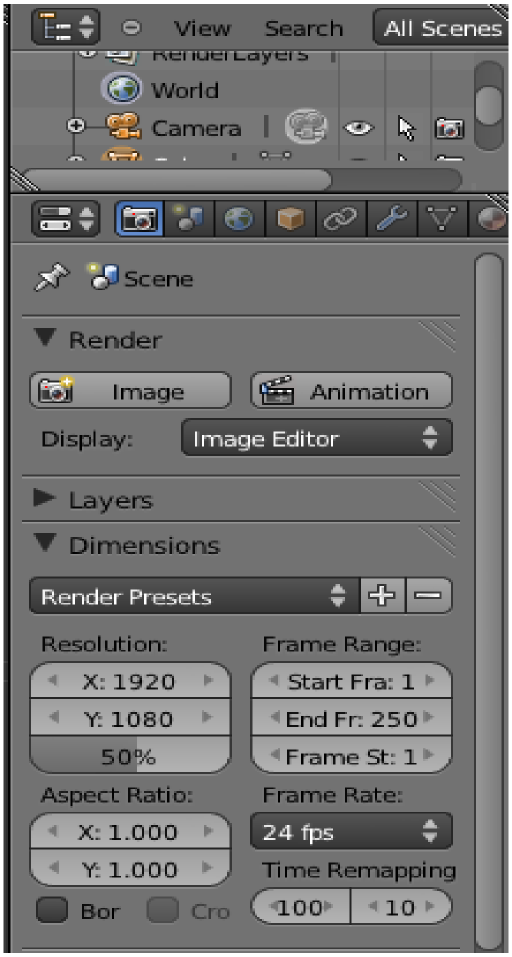
6. Design and Graphics of the Objects in Blender 2.60

The objects can be designed in the Blender tool by the variety of properties and built in meshes, circles, cones and many structures which can be edited and modified as per the design required.

Below is the preview of these objects palette:



The Blender has many wide range of properties and toolbox for each type of mesh or any parent object type. The windows for tools and properties can be enabled under settings and different properties can be set or altered to meet the desired output as shown below:



Display: Image Editor

Resolution:
X: 1920
Y: 1080
50%

Frame Range:
Start Fra: 1
End Fr: 250
Frame St: 1

Aspect Ratio:
X: 1.000
Y: 1.000

Frame Rate:
24 fps

Time Remapping
100
10

Bor
Cro

For the design of the Predator and the Prey, we have used the toolbox for surface subdivision which creates the sub surfaces for the cells modified or selected and thus we can highlight or change the color of the cells with the material window as shown in the figures below:

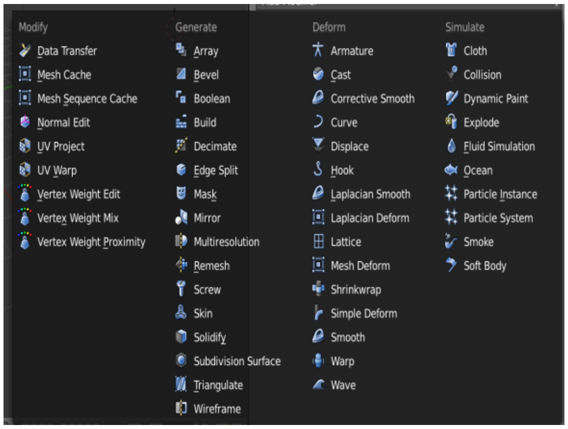


Figure 10. The palette for adding a modifier to the object in the Blender 2.60

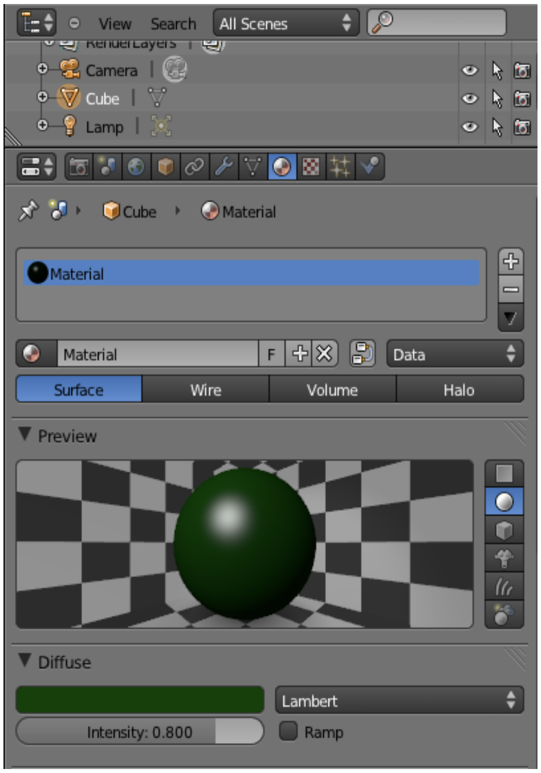


Figure 11. The material change palette

We have used these properties and created the curves for the eyes, nose and the mouth of the Predator and the Prey. The two objects are also differentiated based on their expressions to show their role in the game.

Below are the objects used in the game for the 3D visualization of the game:

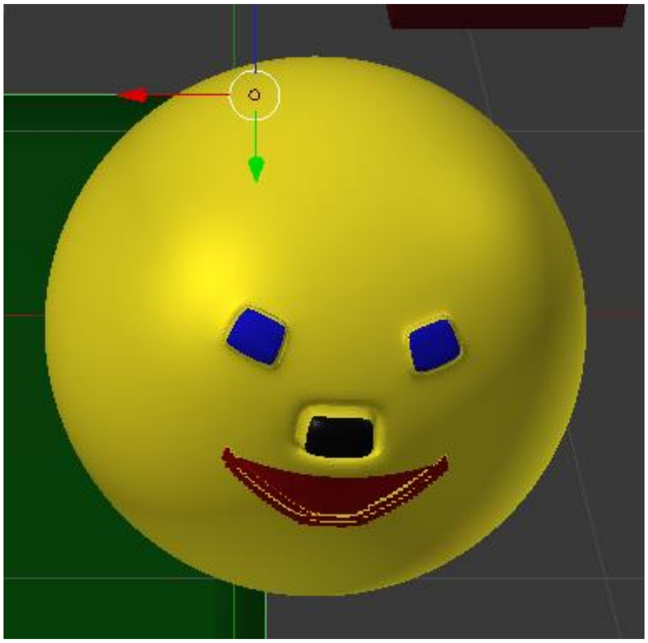


Figure 12. The Prey Design

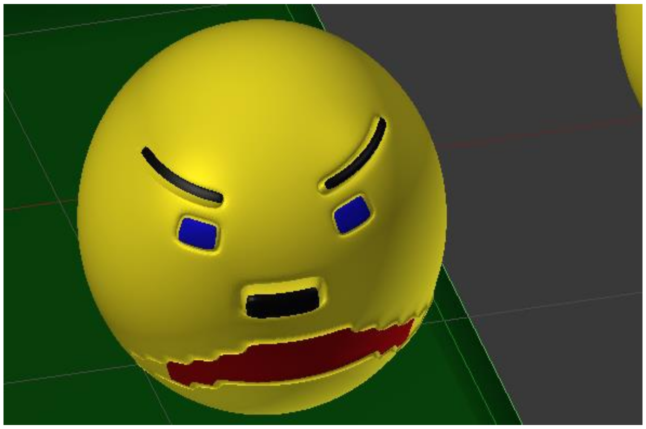


Figure 13. The Predator Design

Each object is carefully designed by the edges and curves properties. It is basic design which allowed us to learn how to use the properties and create more high quality graphics for the simulations or games in future. [4]

The game setup and cell space is designed by the combination of cubes, which is a built-in structure provided by the Blender toolbox as shown below:

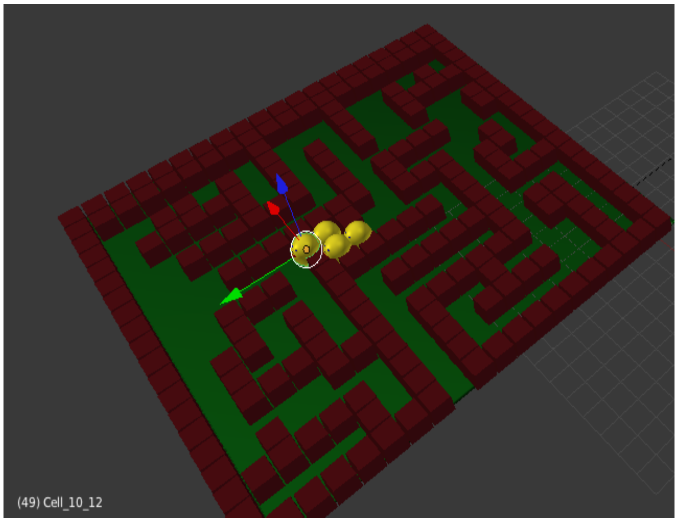


Figure 14. The scene for the game

The blender file attached to the simulation is made generic so it requires just to change the model and the log files to simulate different kinds of initializations and the same objects can be used for the animation as well. [5]

The simulation continues with the different log values and the objects moved along the cell space with the speed and time mentioned in the model file as shown below:

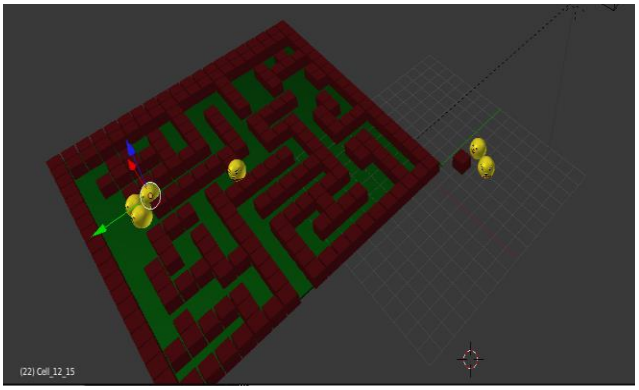


Figure 15. The simulation results

7. Conclusion

The project implementation was a great learning experience and it will definitely help us to implement some great graphics and animation in the world of gaming and simulation. The Blender 2.7 is already here with a vast toolbox of new elements, properties and API for the Python versions. Even the Python has been evolving along with the Blender API changes. The graphics can be further enhanced by using the latest versions of both.

The game is designed for the basic animation and the case in the project Predator Prey. This can be extended to the next levels with minimal changes to the same python script and the graphics designed. The Blender tool is a plus when you need the quick high quality user interfaces and can be animated in the animations window as well. Due to the logs of the simulation in CD++ format, we have to follow the python way of doing the animation of the simulation but it can always be animated using the Blender animation toolbox for the objects. The future enhancements can be implemented in terms of both graphics and cases for this game.

8. References

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

[2] 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)

[3] 3D Visualization of DEVS/Cell-DEVS (DCD) Models by Colin Timmons

[4] how to make a pacman in blender 3d v 2.76: spoken tutorial (beginner)

https://www.youtube.com/watch?v=UVlc6U3O0Bg

[5] Blender Tutorial for Beginners – How to make a mushroom

https://www.raywenderlich.com/49955/blender-tutorial-for-beginners-how-to-make-a-mushroom