Model Driven Engineering

Individual Project:

Meta-Modeling for a Role-Playing Formalism in arKItect

Bojan Arnaudovski

University of Antwerpen, January 2014

Table of Contents

Abstract

ArKItect is a domain-specific meta-modeling tool for designing and representing complex, hierarchical systems. Aside from the meta-modeling feature, arKItect also provides a graphical representation of the model representation and possibility of generating documents and reports such as XML, HTML, Excel and Word. Two user types are supported in arKItect, the Developer (can only modify the project data and its variants) and the Designer (can access meta-modeling, define rules, filters, types and attributes). In this project I will use the arKItect Designer for making some parts of the Role Playing Game. First, I will begin by defining the abstract and concrete syntax of RPG and then do the operational semantics as similar as possible with those done with AtoMPM. Also a detailed comparison between arKItect and AtoMPM will be provided.

Introduction

The main task of this project is to use the knowledge and experience that we've acquired by working on the RPG in AtoMPM, to design the RPG in the tool arKItect. ArKItect is a DSM tool that was developed by Samuel Boutin, Joe Matta and Konstantin Smolin from the French software company Knowledge Inside in 2007. The first version of arKItect was release in 2007 and Renault and Cheuvreux were its first costumers. Since 2008, the number of costumers grew from year to year which made arKItect a crucial software tool for every system engineer/architect.

In the first section of this report, I will discuss the abstract syntax of the RPG. As we remember from defining the abstract syntax in AtoMPM where we use the Class Diagrams as a main entity representation, in arKItect it is a bit different. ArKItect doesn't support Class Diagram representation so in this case, I define arKItect objects and make them appear (visual and operational representation) as close as they can with the Class Diagrams in AtoMPM. Also instead of the Associations in AtoMPM, arKItect uses Flows to represent the connection between the objects.

In the second section, I will give an overview of the concrete syntax of the RPG. Also for this part, arKitect differs from the features provided for designing concrete syntax in AtoMPM. Because arKitect doesn't support this option, for this part of the project, I give a graphical representation of each object with its corresponding image.

The third section of this report represents the operational semantics for the RPG. Like I mentioned for the previous parts and also for this section, arKItect doesn't support operation semantics and transformations. For this part of the project, I used Python scripting which is an additional feature of arKitect. With the use of these Python scripts, I managed to make the operational semantics for Required parts (as mentioned in the MDE assignments) similar with operations in metaDepth where no visual representation is available.

The fourth section will be reserved for my conclusion on this project, where I will also provide some of the key advantages and disadvantages of arKItect in comparison with AtoMPM.

Abstract syntax

In this section I am going to discuss the abstract syntax of the RPG. Because arKItect lacks Class Diagram models, I used the arKItect objects, attributes and flows to represent the abstract syntax. First I begin by creating the objects with its attributes similar to the Classes from the abstract syntax done in AtoMPM. The objects in arKItect consist of attributes of different types such as Integer, Boolean, Enum, String and etc. There is also a special type of attribute called "Program" that is used for writing Python Scripts that I will explain later in this report. In comparison with the abstract syntax done in AtoMPM, arKitect doesn't have constraint (for this part, I wrote Python scripts that act like constraints similar to the constraints written in AtoMPM) and cardinality fields.

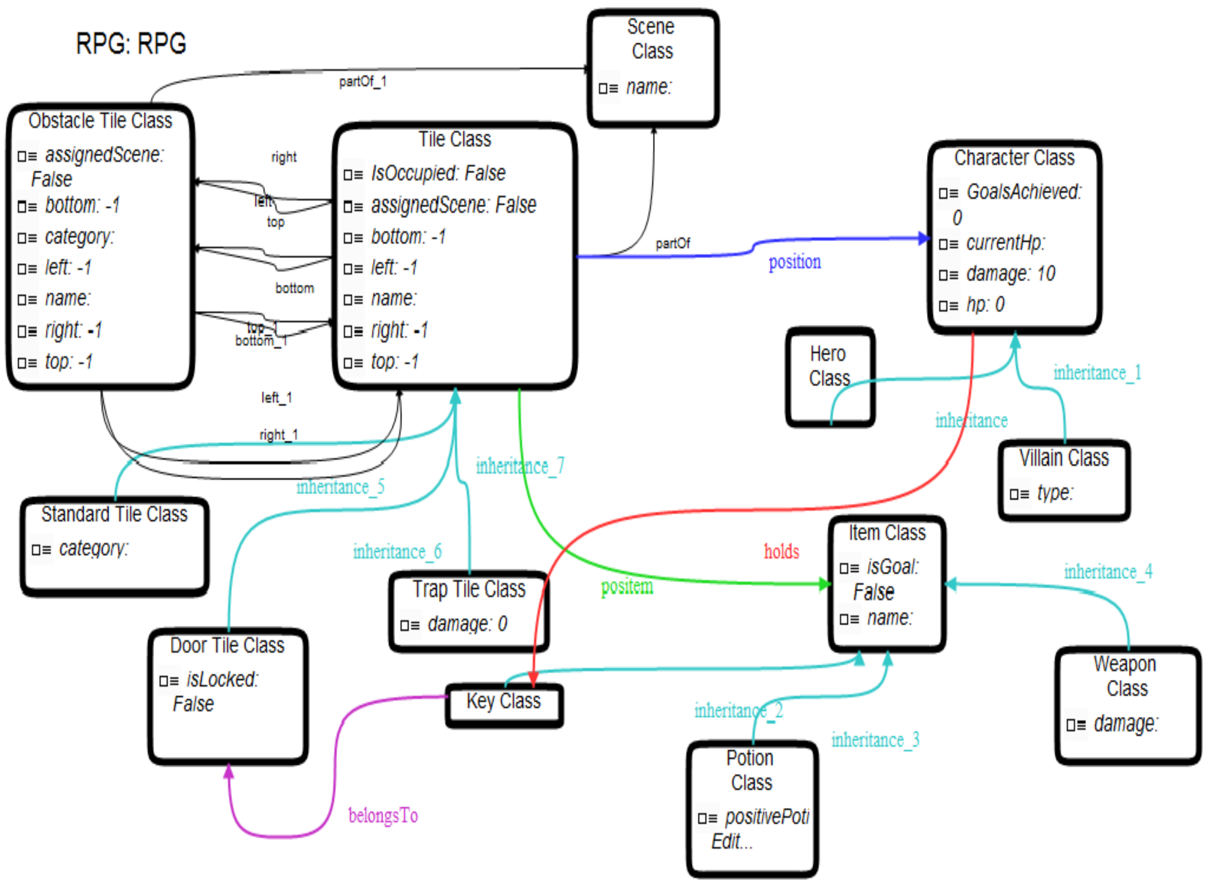


Figure 1 Abstract syntax of the RPG in arKItect

To represent the associations (holds, part of, belongsTo, inheritance and etc.) and the inheritance, I use the arKItect flows which are displayed in different colors. Worth mentioning is that in comparison with AtoMPM, where we could define a class as abstract, this is not possible in arKitect. Also instead of the actions field from the Class Diagram in AtoMPM, in arKItect there is the option of triggering event (Python scripts) which I will explain later in this report.

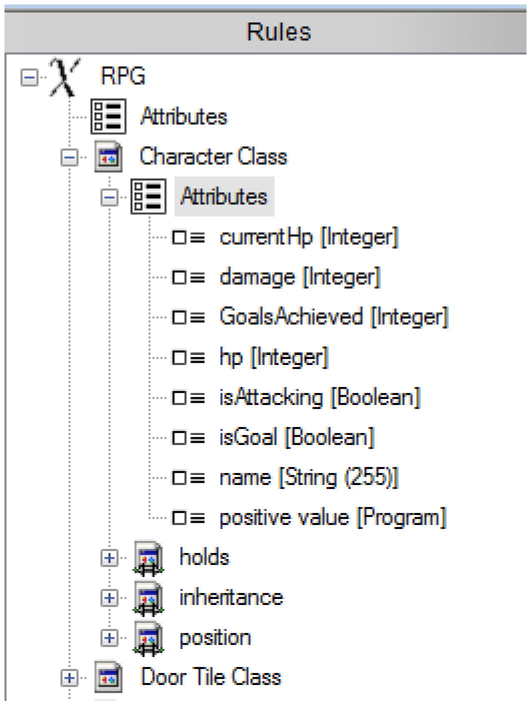
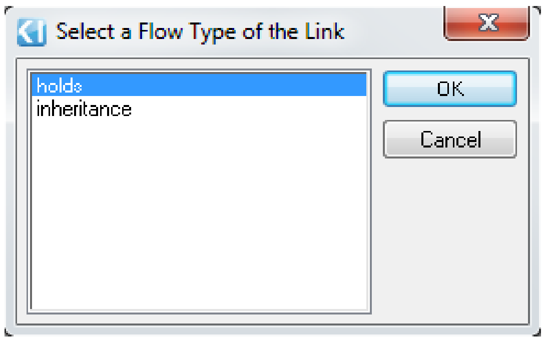

```
Rules
⊟ RPG
    Attributes
  ⊟ Character Class
    ⊟ Attributes
        currentHp [Integer]
        damage [Integer]
        GoalsAchieved [Integer]
        hp [Integer]
        isAttacking [Boolean]
        isGoal [Boolean]
        name [String (255)]
        positive value [Program]
    ⊞ holds
    ⊞ inheritance
    ⊞ position
  ⊞ Door Tile Class
```

Figure 2 Overview of the rule "CharacterClass" for the RPG in arKItect

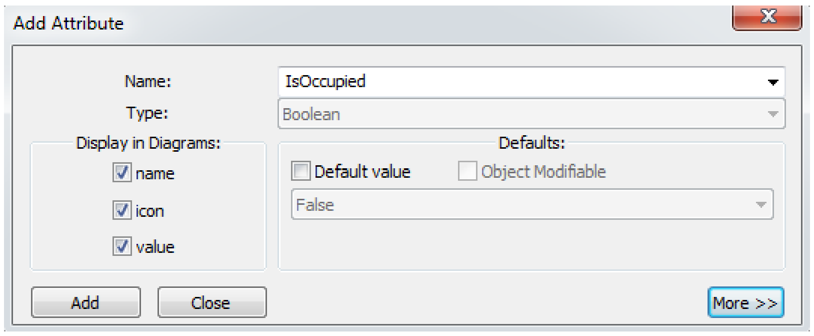
From the picture above, we can see the rule (object) "CharacterClass" with its attributes and flows/associations that he can have. Later we deploy these objects in the Internal Block Diagram and we create the connections between the objects. Interesting about arKItect is that we can connect two objects only if they both have the same data flows/associations defined in the meta-model (like in Figure2).


Select a Flow Type of the Link

holds
inheritance

Figure 3 Selecting type of a flow between objects

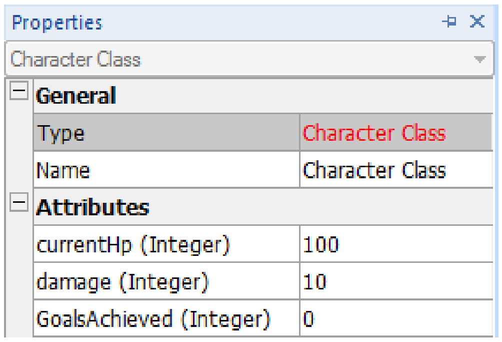
When we want to add an attribute to an object, we can either select from the list of available attributes that we made or we can define a new one, together with the type and the default value.



Name: IsOccupied
Type: Boolean
[✓] name
[✓] icon
[✓] value
[ ] Default value
[ ] Object Modifiable
False

Figure 4 Adding an attribute to an object in arKItect

If we want to modify the objects attributes, we can check the properties panel and set the values of the given attributes.



General	
Type	Character Class
Name	Character Class
Attributes	
currentHp (Integer)	100
damage (Integer)	10
GoalsAchieved (Integer)	0

Figure 5 Properties of the selected object

The RPG made in arKItect (in accordance with the Class Diagram representation shown in Figure1.) consists of one Scene "Azar's Foreset", one Character: Hero "Sepiroth" and one Item: Weapon, which is the goal of the game. When the hero picks up the item the game finishes. The Scenes (size: 3x2) is made of 5 Standard Tiles and 1 Obstacle Tile.

Concrete Syntax

Making the concrete syntax in arKItect is different from making it in AtoMPM. Because arKItect doesn't have a predefined model toolbar for creating the concrete syntax (like the icon instance in AtoMPM), I created new rules/objects (different from the ones used in the Class Diagram representation) and along with it a new hierarchical model. Inside the Scene rule, I defined the other rules: Tile (Standard Tile), Obstacle Tile, Item (Key, Weapon and Potion) and Character (Hero or Villain). This scenario also adds restrictions of creation objects that are not in that level of hierarchy. For example, we cannot add another Scene object in the current Scene. Also if we want to add a Hero or A Villain object, it will allow us to add it inside the Character object. So for giving a visual representation of the objects, I uploaded their corresponding picture as a foreground image. Later when we create the RPG model, we select the object with its visual representation from the palette panel in arKItect, similar to the default Icons toolbar in AtoMPM.

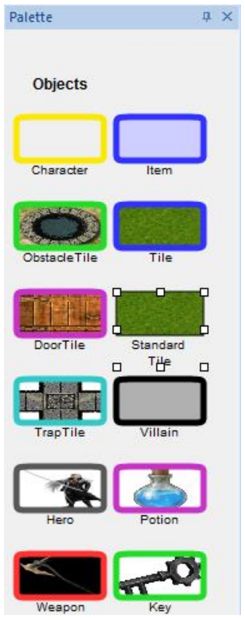


Figure 6 Palette of the objects in arKItect

Important to mention is that, arKItect in comparison with AtoMPM, doesn't have mapper and parser fields so like in the case with choosing the category of the Standard Tile (where in AtoMPM we could write code in the mapper to select the required picture with the selected category), in arKItect that can only be done with Python script in a Program attribute.

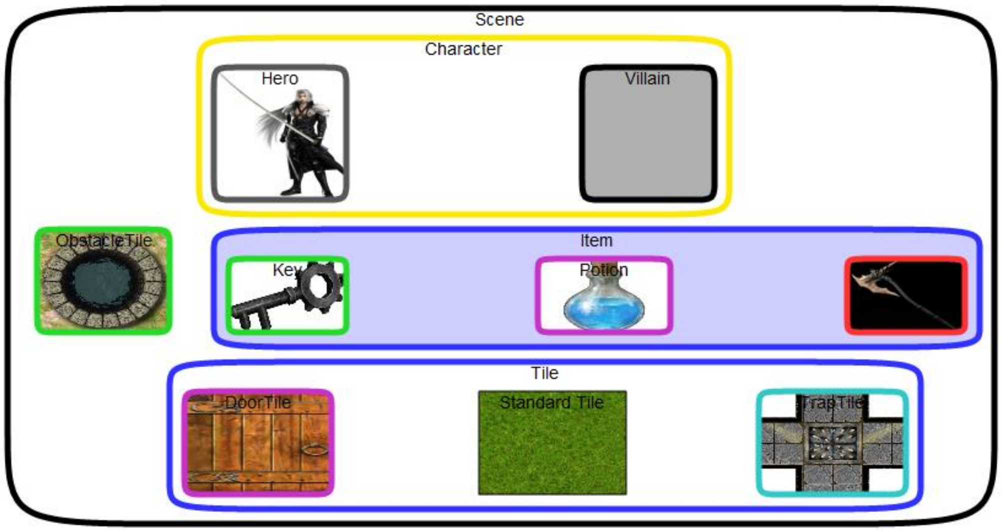
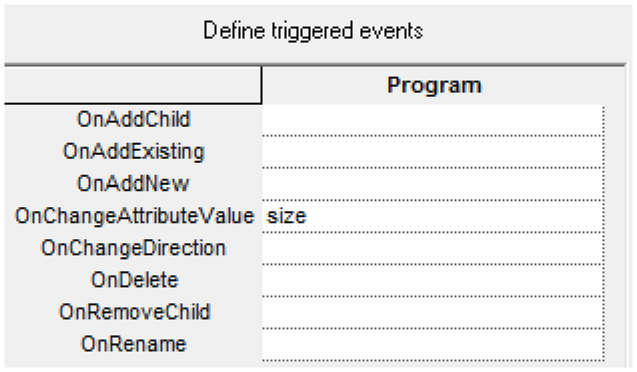


Figure 7 RPG concrete syntax in arKItect

Operational Semantics and Constraints

As I mentioned in the Introduction section, operational semantics and transformation and not supported in arKitect. However I managed to do required parts (as mentioned in the assignment for the operational semantics) of the RPG through Python scripts. From what I've specified in the above sections, arKitect has the option of adding an attribute of type ”Program”, where we write the Python script which can be executed either by only clicking on the selected attribute or as an event triggered by some actions that we perform on the object. Because arKitect limits the selection of a triggering event to only one, some of the scripts have to be run manually (by clicking on the selected attribute).


Define triggered events

	Program
OnAddChild	
OnAddExisting	
OnAddNew	
OnChangeAttributeValue	size
OnChangeDirection	
OnDelete	
OnRemoveChild	
OnRename	

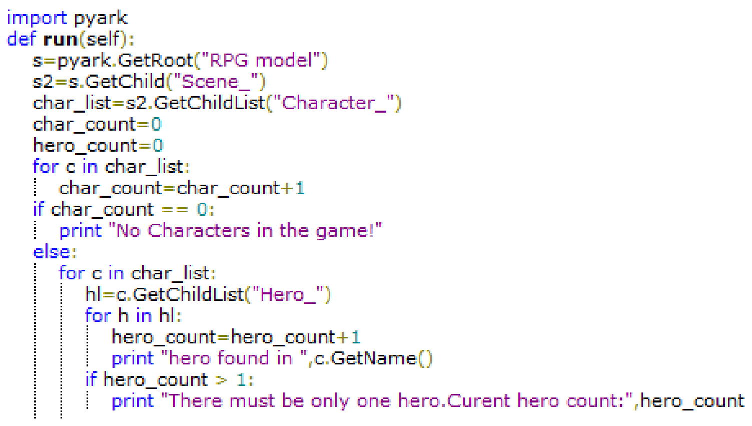
Figure 8 Example of a triggered events panel in the object Scene

Important to mention is that the output of the operational semantics cannot be shown visually in arKitect. The final output is represented in textual form containing of steps similar with the output from the tool “metaDepth”, which I will discuss further in this part.

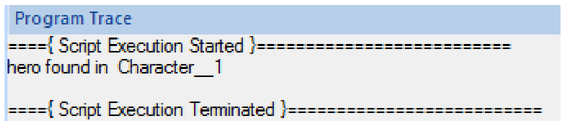
When it comes to the constraints because arKItect lacks this option, all of the constraints are written with Python scripts and executed either manually or triggered by event. Also I like to point out that unlike in AtoMPM, where the abstract and concrete syntax are connected with the final model and the operational semantics, in arKItect they are not connected. That's why I had to make some changes in the “RPG model” (adding some additional attributes, so that I can make the model operational) in comparison with the abstract and concrete syntax that I've discussed earlier in this report.

First, I am going to begin by explaining each of the constraints for the RGP, made in arKitect, and then I am going to jump to the operational part of this project.

The constraint “hero count” (executed manually) appears in the Scene object, which goes into all of the Characters objects and checks if their child (inside the Character object we add a child, either a Hero or a Villain) is Hero and counts its appearance. If the counter for the Hero is bigger the one, it gives a warning message that we should have only one Hero in the entire game. The option of deleting the other Hero or types of objects or not allowing the user to add a Hero character cannot be done as arKitect doesn't have methods for such operations.



```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    char_list=s2.GetChildList("Character_")
    char_count=0
    hero_count=0
    for c in char_list:
        char_count=char_count+1
    if char_count == 0:
        print "No Characters in the game!"
    else:
        for c in char_list:
            hl=c.GetChildList("Hero_")
            for h in hl:
                hero_count=hero_count+1
                print "hero found in ",c.GetName()
            if hero_count > 1:
                print "There must be only one hero.Curent hero count:",hero_count
```

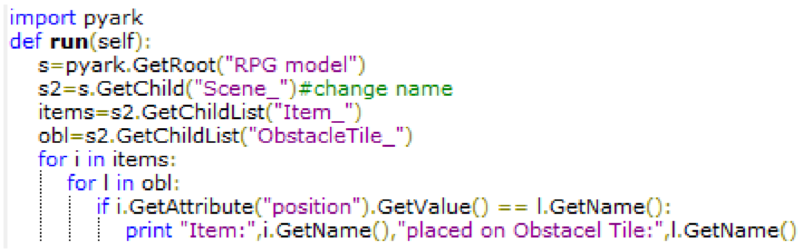


Program Trace

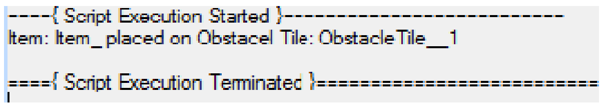
```
====[ Script Execution Started ]=========================
hero found in  Character__1

====[ Script Execution Terminated ]=========================
```

Figure 9 Constraint "hero count" and the output from its execution

The constraint "item_tile_check" is part of the Scene object (executed manually), which checks is the item placed on a normal tile or on an Obstacle tile. It checks the "position" attribute of the Items when performing the check. If there are no violations of the constraint no output message is printed.



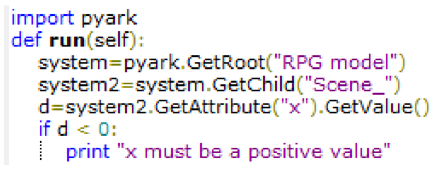
```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")#change name
    items=s2.GetChildList("Item_")
    obl=s2.GetChildList("ObstacleTile_")
    for i in items:
        for l in obl:
            if i.GetAttribute("position").GetValue() == l.GetName():
                print "Item:",i.GetName(),"placed on Obstacel Tile:",l.GetName()
```



```
----[ Script Execution Started ]----------------------------
Item: Item_ placed on Obstacel Tile: ObstacleTile__1

====[ Script Execution Terminated ]=========================
```

Figure 10 Constraint "item_tile_check" and its output

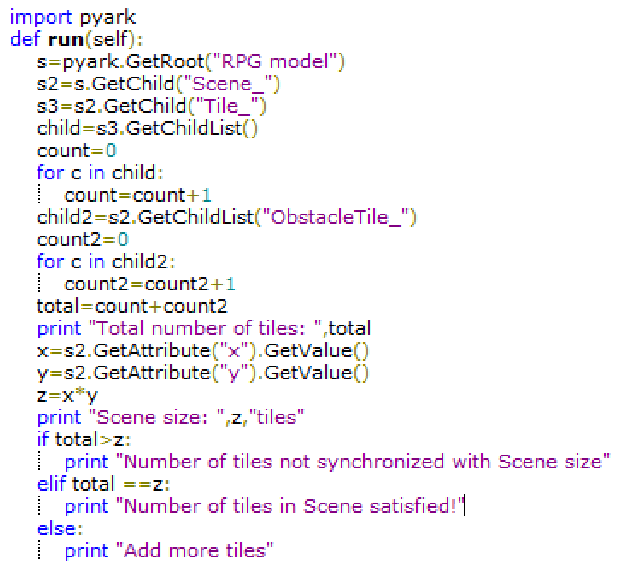
Next we have the "positiveX" and "positiveY" constraints (manually executed), which are part of the Scene object and check if the values for the attributes x and y are positive values. The object Scene has this two attributes x and y that we set when we set the size of the Scene.



```
import pyark
def run(self):
    system=pyark.GetRoot("RPG model")
    system2=system.GetChild("Scene_")
    d=system2.GetAttribute("x").GetValue()
    if d < 0:
        print "x must be a positive value"
```

Figure 11 Constraint "PositiveX"

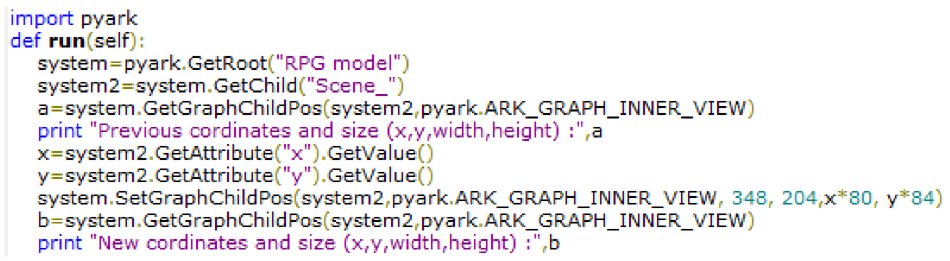
The constraint “right_num_tiles” (executed manually), which is part of the Scene object, once executed will count all the tiles (standard and obstacle tile) and check that number with the size of the current Scene (x and y attributes). Depending on the result number it will output either the number with the size is satisfied or add more tiles or not synchronized with the size of the Scene.



```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    s3=s2.GetChild("Tile_")
    child=s3.GetChildList()
    count=0
    for c in child:
        count=count+1
    child2=s2.GetChildList("ObstacleTile_")
    count2=0
    for c in child2:
        count2=count2+1
    total=count+count2
    print "Total number of tiles: ",total
    x=s2.GetAttribute("x").GetValue()
    y=s2.GetAttribute("y").GetValue()
    z=x*y
    print "Scene size: ",z,"tiles"
    if total>z:
        print "Number of tiles not synchronized with Scene size"
    elif total ==z:
        print "Number of tiles in Scene satisfied!"
    else:
        print "Add more tiles"
```

Figure 12 Constraint "right_num_tiles"

The program attribute “size” (executed on event “change attribute”), which is part of the Scene object redraws the Scene object in the Internal Block Diagram depending on the attributes x and y so it can fit all the tiles (similar with resizing the Scene in AtoMPM).

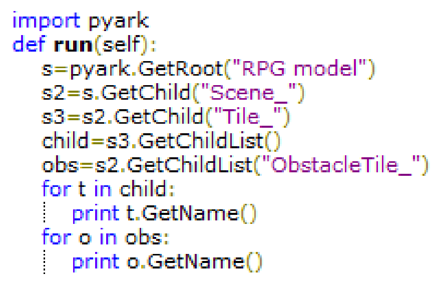


```
import pyark
def run(self):
    system=pyark.GetRoot("RPG model")
    system2=system.GetChild("Scene_")
    a=system.GetGraphChildPos(system2,pyark.ARK_GRAPH_INNER_VIEW)
    print "Previous cordinates and size (x,y,width,height) :",a
    x=system2.GetAttribute("x").GetValue()
    y=system2.GetAttribute("y").GetValue()
    system.SetGraphChildPos(system2,pyark.ARK_GRAPH_INNER_VIEW, 348, 204,x*80, y*84)
    b=system.GetGraphChildPos(system2,pyark.ARK_GRAPH_INNER_VIEW)
    print "New cordinates and size (x,y,width,height) :",b
```

Figure 13 Program attribute "size"

The constraint “one item” (executed on event “add child”), which is part of the Item object, checks if the Item object has more than one child (Weapon, Key or Potion). When adding a child item object it immediately displays a warring message that only one child per Item Object is allowed. The option of removing the newly added child Item object cannot be done in arKitect so instead the user is warned in advance.

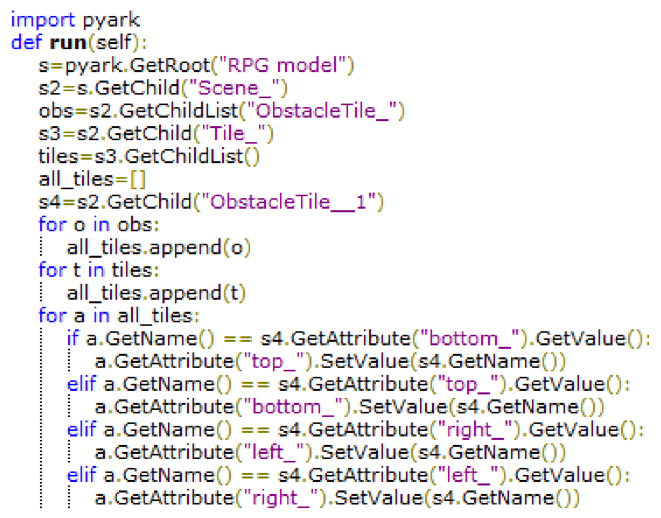
The “get all Tiles” program attribute (executed manually), which is part of Character, Standard Tile, Obstacle Tile and Item object, returns the instances of all the Tile (Standard and Obstacle) objects so later we can set the update the “position” attribute when either we connect tiles, place characters or items on tiles.



```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    s3=s2.GetChild("Tile_")
    child=s3.GetChildList()
    obs=s2.GetChildList("ObstacleTile_")
    for t in child:
        print t.GetName()
    for o in obs:
        print o.GetName()
```

Figure 14 Program attribute "get all Tiles"

The program attribute “update connections” (executed manually), which can be found in Tile Objects, connects a give tile with its neighboring tiles. The script begins by collecting the values for the attributes (top_, bottom_, left_ and right_ , where we write the name of the tile that we want the current tile to connect to) then finds the stated tiles and it sets the name of the current tile as its neighbor depending on the given side. For example, “StandardTile__1” has “StandardTile__2” as right neighbor and when we execute the script, the attribute “left_” in “StandardTile__2” will be set to “StandardTile__1”.

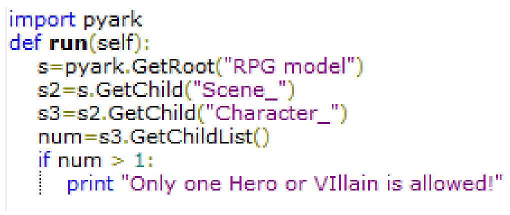


```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    obs=s2.GetChildList("ObstacleTile_")
    s3=s2.GetChild("Tile_")
    tiles=s3.GetChildList()
    all_tiles=[]
    s4=s2.GetChild("ObstacleTile__1")
    for o in obs:
        all_tiles.append(o)
    for t in tiles:
        all_tiles.append(t)
    for a in all_tiles:
        if a.GetName() == s4.GetAttribute("bottom_").GetValue():
            a.GetAttribute("top_").SetValue(s4.GetName())
        elif a.GetName() == s4.GetAttribute("top_").GetValue():
            a.GetAttribute("bottom_").SetValue(s4.GetName())
        elif a.GetName() == s4.GetAttribute("right_").GetValue():
            a.GetAttribute("left_").SetValue(s4.GetName())
        elif a.GetName() == s4.GetAttribute("left_").GetValue():
            a.GetAttribute("right_").SetValue(s4.GetName())
```

Figure 15 Program attribute "update connections"

The constraint “same type” (executed on event add new object), which is part of the Tile object, checks if we add another object of type “Tile”. This constraint was made so that only one type of the Tile object will exist per Scene. As we know till now, the Tile object has Standard tile as child which can be Road, Grass and Concrete that we choose when we design the Scene.

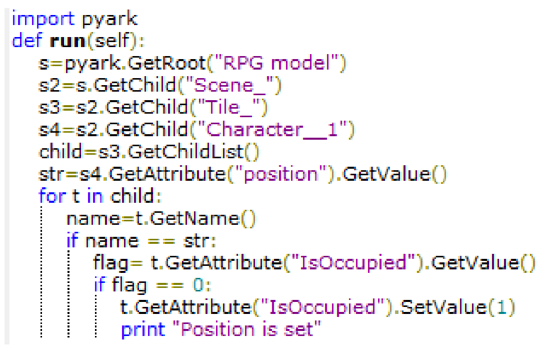
The constraint “hero or villain” (executed on event add child) is part of the Character object, and give us a warning message that the Character object can only have one child object, a Hero or a Villain. The newly added object cannot be deleted so the user has to perform that action.



```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    s3=s2.GetChild("Character_")
    num=s3.GetChildList()
    if num > 1:
        print "Only one Hero or VIllain is allowed!"
```

Figure 16 Constraint "hero or villain"

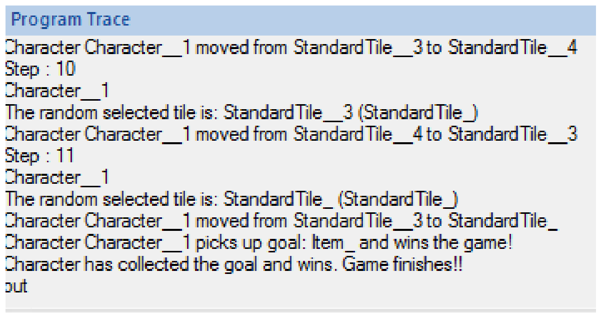
The Program attribute "set position" (executed manually) is part of the Character object and I used it for setting the position of the character. Before we execute this script we decide on which tile we are going to place the character (only on standard and not occupied tile) by running the script "get tiles" and then we set that tile as the attribute "position" of the character. After this we run the "set position" attribute and it sets the tile where the hero is placed to is occupied (Boolean type).



```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    s3=s2.GetChild("Tile_")
    s4=s2.GetChild("Character__1")
    child=s3.GetChildList()
    str=s4.GetAttribute("position").GetValue()
    for t in child:
        name=t.GetName()
        if name == str:
            flag= t.GetAttribute("IsOccupied").GetValue()
            if flag == 0:
                t.GetAttribute("IsOccupied").SetValue(1)
                print "Position is set"
```

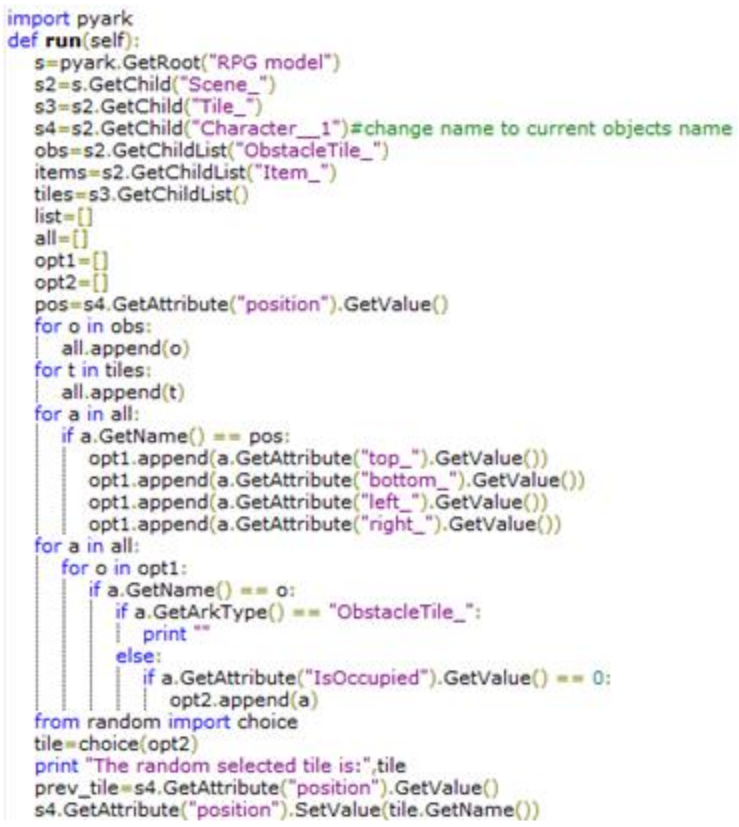
Figure 17 Program attribute "set position"

The "move" program attribute (executed manually) is part of the Character object and represents the move and collect items operation for the character. Depending on the tile where the Character is positioned, he can only move to its neighboring tile that are not occupied by other Character or are not Obstacle tiles. This condition limit the choice of the character and by random selection a tile is selected. After performing the move to the other tile, the value of the attribute "IsOccupied" in the new tile is set to True and in the previous tile to False. Also when moving to a new tile, an item check is performed so if the Character picks up an Item (the attribute "has item" is set to the name of the item) and If the Item is a goal, the "collected goal" is set to True.



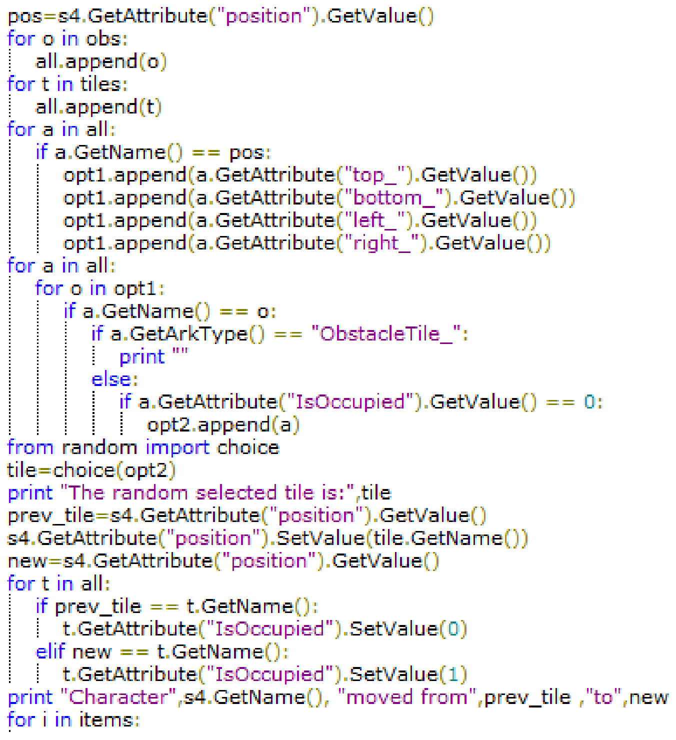
```
Program Trace
Character Character__1 moved from StandardTile__3 to StandardTile__4
Step : 10
Character__1
The random selected tile is: StandardTile__3 (StandardTile_)
Character Character__1 moved from StandardTile__4 to StandardTile__3
Step : 11
Character__1
The random selected tile is: StandardTile_ (StandardTile_)
Character Character__1 moved from StandardTile__3 to StandardTile_
Character Character__1 picks up goal: Item_ and wins the game!
Character has collected the goal and wins. Game finishes!!
out
```

Figure 18 Output from the "move" attribute



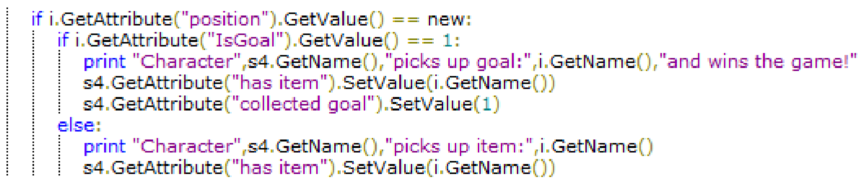
```
import pyark
def run(self):
    s=pyark.GetRoot("RPG model")
    s2=s.GetChild("Scene_")
    s3=s2.GetChild("Tile_")
    s4=s2.GetChild("Character__1")#change name to current objects name
    obs=s2.GetChildList("ObstacleTile_")
    items=s2.GetChildList("Item_")
    tiles=s3.GetChildList()
    list=[]
    all=[]
    opt1=[]
    opt2=[]
    pos=s4.GetAttribute("position").GetValue()
    for o in obs:
        all.append(o)
    for t in tiles:
        all.append(t)
    for a in all:
        if a.GetName() == pos:
            opt1.append(a.GetAttribute("top_").GetValue())
            opt1.append(a.GetAttribute("bottom_").GetValue())
            opt1.append(a.GetAttribute("left_").GetValue())
            opt1.append(a.GetAttribute("right_").GetValue())
    for a in all:
        for o in opt1:
            if a.GetName() == o:
                if a.GetArkType() == "ObstacleTile_":
                    print ""
                else:
                    if a.GetAttribute("IsOccupied").GetValue() == 0:
                        opt2.append(a)
    from random import choice
    tile=choice(opt2)
    print "The random selected tile is:",tile
    prev_tile=s4.GetAttribute("position").GetValue()
    s4.GetAttribute("position").SetValue(tile.GetName())
```

Figure 19 Program attribute "move" part 1



```
pos=s4.GetAttribute("position").GetValue()
for o in obs:
    all.append(o)
for t in tiles:
    all.append(t)
for a in all:
    if a.GetName() == pos:
        opt1.append(a.GetAttribute("top_").GetValue())
        opt1.append(a.GetAttribute("bottom_").GetValue())
        opt1.append(a.GetAttribute("left_").GetValue())
        opt1.append(a.GetAttribute("right_").GetValue())
for a in all:
    for o in opt1:
        if a.GetName() == o:
            if a.GetArkType() == "ObstacleTile_":
                print ""
            else:
                if a.GetAttribute("IsOccupied").GetValue() == 0:
                    opt2.append(a)
from random import choice
tile=choice(opt2)
print "The random selected tile is:",tile
prev_tile=s4.GetAttribute("position").GetValue()
s4.GetAttribute("position").SetValue(tile.GetName())
new=s4.GetAttribute("position").GetValue()
for t in all:
    if prev_tile == t.GetName():
        t.GetAttribute("IsOccupied").SetValue(0)
    elif new == t.GetName():
        t.GetAttribute("IsOccupied").SetValue(1)
print "Character",s4.GetName(), "moved from",prev_tile ,"to",new
for i in items:
```

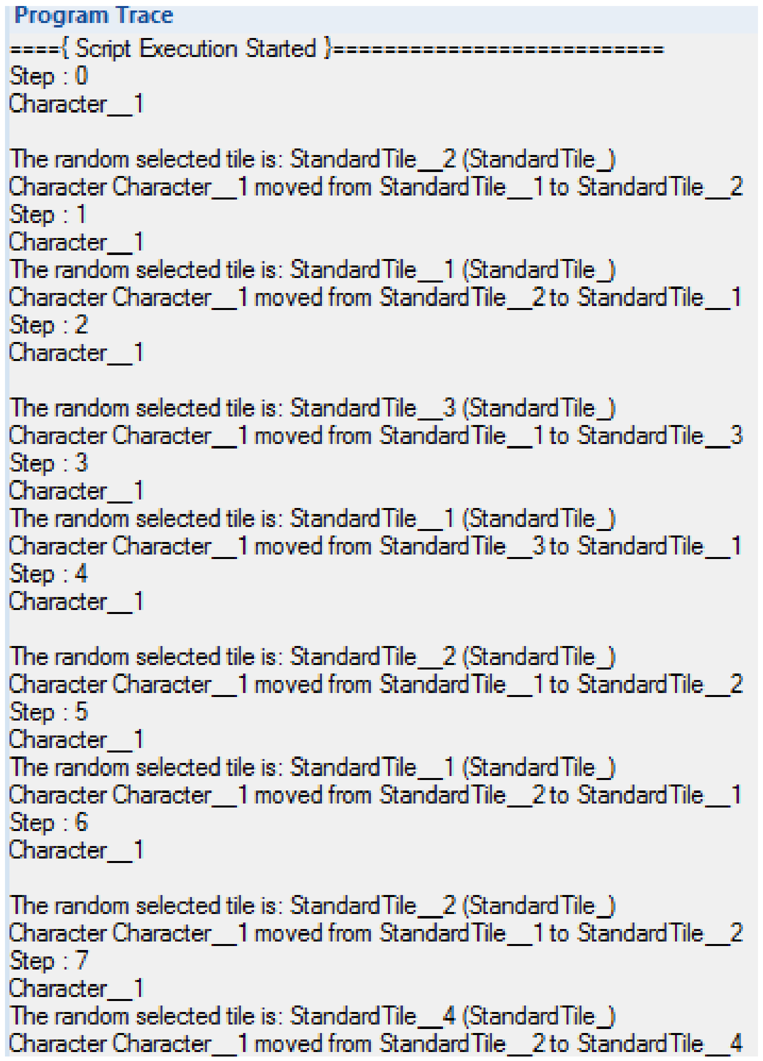
Figure 20 Program attribute "move" part 2



```
if i.GetAttribute("position").GetValue() == new:
    if i.GetAttribute("IsGoal").GetValue() == 1:
        print "Character",s4.GetName(),"picks up goal:",i.GetName(),"and wins the game!"
        s4.GetAttribute("has item").SetValue(i.GetName())
        s4.GetAttribute("collected goal").SetValue(1)
    else:
        print "Character",s4.GetName(),"picks up item:",i.GetName()
        s4.GetAttribute("has item").SetValue(i.GetName())
```

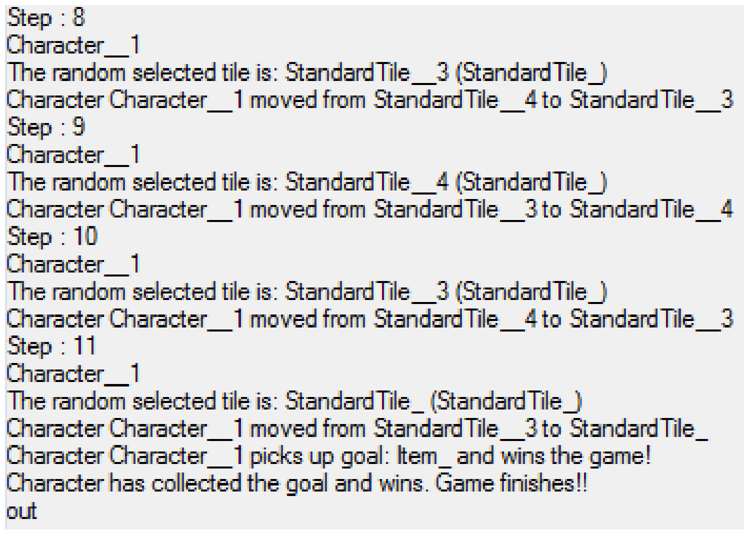
Figure 21 Program attribute "move" part 3

Finally the program attribute “simulate” (executed manually), part of the Scene object is used form running the simulation. This script executes the move operation for every character and checks in each step weather some of the Character has collected the goal (collected goal = True). If so it finishes the game.



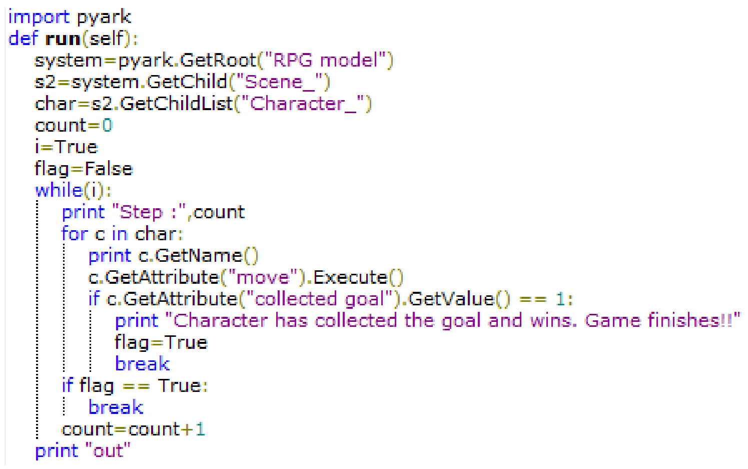
```
===={ Script Execution Started }==========================
Step : 0
Character__1

The random selected tile is: StandardTile__2 (StandardTile_)
Character Character__1 moved from StandardTile__1 to StandardTile__2
Step : 1
Character__1
The random selected tile is: StandardTile__1 (StandardTile_)
Character Character__1 moved from StandardTile__2 to StandardTile__1
Step : 2
Character__1

The random selected tile is: StandardTile__3 (StandardTile_)
Character Character__1 moved from StandardTile__1 to StandardTile__3
Step : 3
Character__1
The random selected tile is: StandardTile__1 (StandardTile_)
Character Character__1 moved from StandardTile__3 to StandardTile__1
Step : 4
Character__1

The random selected tile is: StandardTile__2 (StandardTile_)
Character Character__1 moved from StandardTile__1 to StandardTile__2
Step : 5
Character__1
The random selected tile is: StandardTile__1 (StandardTile_)
Character Character__1 moved from StandardTile__2 to StandardTile__1
Step : 6
Character__1

The random selected tile is: StandardTile__2 (StandardTile_)
Character Character__1 moved from StandardTile__1 to StandardTile__2
Step : 7
Character__1
The random selected tile is: StandardTile__4 (StandardTile_)
Character Character__1 moved from StandardTile__2 to StandardTile__4
```

Figure 22 Output from running the “simulate” attribute part 1



```
Step : 8
Character__1
The random selected tile is: StandardTile__3 (StandardTile_)
Character Character__1 moved from StandardTile__4 to StandardTile__3
Step : 9
Character__1
The random selected tile is: StandardTile__4 (StandardTile_)
Character Character__1 moved from StandardTile__3 to StandardTile__4
Step : 10
Character__1
The random selected tile is: StandardTile__3 (StandardTile_)
Character Character__1 moved from StandardTile__4 to StandardTile__3
Step : 11
Character__1
The random selected tile is: StandardTile_ (StandardTile_)
Character Character__1 moved from StandardTile__3 to StandardTile_
Character Character__1 picks up goal: Item_ and wins the game!
Character has collected the goal and wins. Game finishes!!
out
```

Figure 23 Output from running the "simulate" attribute part 2



```
import pyark
def run(self):
    system=pyark.GetRoot("RPG model")
    s2=system.GetChild("Scene_")
    char=s2.GetChildList("Character_")
    count=0
    i=True
    flag=False
    while(i):
        print "Step :",count
        for c in char:
            print c.GetName()
            c.GetAttribute("move").Execute()
            if c.GetAttribute("collected goal").GetValue() == 1:
                print "Character has collected the goal and wins. Game finishes!!"
                flag=True
                break
        if flag == True:
            break
        count=count+1
    print "out"
```

Figure 24 Program attribute "simulate"

Comparison and Conclusion

The last part of this report is reserved for comparison between AtoMPM and arKitect and summarization of the entire project. In the previous sections I also stated and discussed some of the differences (encountered while working on this project) so in this part I am also going to add e few more.

First thing that I noticed in arKItect is that its performance is very slow. It takes couple of seconds or sometimes even more to load the entire project or while doing some modifications when working on the project. This can be clearly seen when executing the "simulate" Program attribute, where arKitect takes too much time to compute the output of the script (displaying Not Responding on the icon).This is probably because every change made in project is immediately uploaded to the server. Another thing that arKitect lacks is the support for ".png" files unlike

AtoMPM where this is possible. If we want to attach image to an object, first we have to upload the image to the server and that set it either as foreground or as a background image. As I mentioned previously in this report, arKItect doesn't have support for constraints, operational semantics and transformation and even the documentation [2] for the Python API doesn't have support methods. Other thing that needs to be mentioned is that when we open arKItect, we can see all of our current projects and if a collaboration with another user was made, it will display if the user is currently working on it and what changes were made in the project. This feature is not available in AtoMPM. Finally, I also want to add that I've used the export to Word feature of arKItect were a detailed representation of the objects was created.

It can be concluded from reading this project report and modeling the RPG that arKItect lacks important characteristics that are necessary in the field of Model Driven Engineering where AtoMPM is a very powerful tool in this field. Despite the disadvantages of arKItect, at the end, the project achieved its goal. ArKItect remains an important software tool for designing large scale complex systems that interact with the physical world. The power of arKItect comes from its graphical and visual representation of hierarchical complex systems and also from the feature of exporting complex system architecture into readable textual document.

References

1. http://www.k-inside.com/web/produits-et-services/produits/arkitect-designer/
   arKItect home page
2. https://support.k-inside.com/display/ARKI22/arKItect+2.2.x+documentation+home
   arKItect documentation