Agents in Anylogic - RPGame

Oriol Pla Roma

University of Antwerp, Antwerp, Belgium

Abstract

Anylogic is one of the most up-to-date tools when creating agent-based models, a new analytical method for the simulation of social behaviours. In this study an RPG (Role Play Game) model is going to be created thanks to the different tools that Anylogic provides in the topic. The design of the model will be explained in detail to show the simplicity and the versatility this program gives to the user.

Keywords: Anylogic, Agent-based modeling, RPG, Statecharts, Agent, Environment, Modeling

1. Introduction

Agent-based modeling allows us to simulate the attitude of some agents on a particular environment. In this work the agents are the characters of the RPG and the environment is defined by the game rules.

The second section goes into the details of the context the paper is built, agent-based modeling, and introduces Anylogic, the tool used, and which role plays. Section 3 explains the main goal of the game and the rules in which it is based. Section 4 provides the description of the model design and how it has been applied in Anylogic. Moreover, goes into more details of each agent and their statechart, which defines the behaviour. Section 5 presents the experience with using the RPGame simulator, explains the user interface and the simulation options provided to the user. Section 6 compares the model with other existing Anylogic models and in section 7 the conclusions of the work are written. Finally, section 8 gives an idea about which functionalities could be extended from the work done in order to improve the model.

2. Agent-based modeling in Anylogic

2.1. Agent-based modeling

Agent-based modeling is a computational method that enables a researcher to create, analyze, and experiment with models composed of agents that interact within an environment (Gilbert, 2007 [1]).
In order to thoroughly understand this definition, a more specific analysis of each of the terms is required.
Firstly, when talking about a model, the actual meaning is a reproduction of a real world domain at a computational level. Obviously, this transformation cannot be completely faithful as the real world is too complex to be brought into a computer. Therefore, the main aim when creating a model is to seek for the highest resemblance with the real world, ensuring that the experimental data extracted from that model will be valid. Nevertheless, some features of the representation might be experimentally superfluous, adding useless complexity, and should be avoided to reach an optimal model.
Secondly, the agents' aim is to act according to the behaviour defined by their model. A wide range of different stimulus, as happens in the real world, can influence this behaviour. Consequently, agents can act collectively, can change or be changed by other agents and even have a learning capability.
On the other hand, the environment is the context within which the agents conduct themselves. This space is the reproduction of what that environment would be in the real world. However, it can also have a more secondary role depending on the model, being able to have a large influence on the agents or none at all. For instance, it is common that the environment provides only spatial boundaries.
Once reached this point, one may ask himself: how can a computational model like that be built and executed for experimentation? And here is where Anylogic comes in.

2.2. Anylogic tools

Anylogic is a java-written modeling program that includes different modeling methods such as system dynamics, discrete event or agent-based, being the last one the topic of this paper. To assist the modeling process, the following items are included: stock and flow diagrams, action charts, process flowcharts and the tool used in this project, statecharts. The remarkable hallmark of Anylogic is to put together all these features, so they can be utterly exploited.

In this project, statecharts have been used to define agents' behaviour, as it is an ideal tool for that purpose. Besides states and transitions, it allows a fairly clear representation of artificial intelligence, explained in section 4.

3. The Game

3.1. Outline

The game consists of a hero, who must collect all the goals that appear on the map, and some villains who have to prevent him from achieving the aim by attacking him (as it happens in pacman game).
The map where they interact has a group of walkable tiles that can contain objects, traps and doors, which can be opened using keys. In addition, the hero can improve his attack skills by picking weapons that can be found on the map.

3.2. Rules

The game is based on the following rules:

- An RPGame consists of a world that is divided into scenes that have a number of connected tiles.
- Tiles can be connected to each other from the left, right, top or bottom. This way, a map is created for each scene.
- In the game, there is one hero, and there can be villains.
- Each of these characters is located on a tile, and only one character can be on a tile at the same time.
- A tile can be an empty tile, a trap, a door, or an obstacle where no one can stand.
- A character can move from one adjacent tile to another if it is not an obstacle or it is not occupied.
- A door is a portal to a door on another scene, so the hero can pass through one door to enter another scene. If the hero goes back through the door, he goes back to the original door at the original scene.
- The hero can attack villains and vice versa, if they stand on adjacent tiles.

- Villains do not attack each other.
- A trap hurts the hero, but not a villain.
- The hero and the villains have a health value that depicts how much damage they can take. Health always has a strictly positive value.
- The hero, villains and traps have a damage value that depicts how much damage they inflict. The hero and villains inflict damage if they choose to attack. A trap inflicts damage on the hero if the he steps onto it. Damage is always strictly positive.
- On an empty tile, there can be an item. An item can be picked up by the hero by walking on its tile. Every item can only be picked up once. Items can be a key, a weapon, or a goal.
- A door can be locked, and the hero must pick up a particular key to unlock the door.
- A hero can pick up a weapon, which increases 20
- A hero can pick up a goal. The hero wins if he can pick up all of them. There must be at least one goal in the game.
- The game is simulated in time slices: first, the hero gets one chance to move or attack. Then, all the villains in the same scene, get their chance to move or attack. The order in which the villains get their chance is not determined.
- Villains in a different scene from the scene in which the hero currently resides do not do anything.
- When the hero achieves all the goals, or dies, the game is over.

4. The design

4.1. Starting point

When you start creating a new model in Anylogic you are able to generate different types of template that facilitate the starting point.
In figure 1, the simplest skeleton can be seen: a *Main Class*, where the environment will be treated and that will be the domain controller, and a

Figure 1: Basic starting template.

Simulation Class that, as its name shows, is the responsible for the simulation's program part and does a similar function as a model–view controller. Then, it is necessary to add the agents that will interact with the environment, that in this case are the hero and the villains. However, before doing it, it is recommended to build the base where the agents will move: the environment's structure.

4.2. Structure

When reading the game's operations, it is easy to conclude that some kind of structure that uploads the map's information is needed, so both the hero and the villains know at any time their position in the map. As the environment is a 2D discrete space, a matrix is the element that perfectly adapts to the necessities of the RPGame. This matrix contains the information regarding the sort of the tile and the objects placed on it and this information is saved in an integer following a numerical pattern.
On the other hand, it is also necessary to save the position of certain points in a vector, such as goals, keys or doors, in order to make more efficient their search in the map (instead of looking for them through all the matrix).

4.3. The environment

Another important part of the *Main Class* is generating the visual 2D discrete map that will be triggered by the *Simulation Class.* Based on the previously created structure, a graphical environment like the one on the figure 2 has been generated.
As it can be seen in the legend of figure 2, each sort of tile represented in the structural matrix is a different colour in the graphical matrix: white stands for standard tiles, black is used for walls (where obviously agents cannot move through), yellow is the colour of the hero's goals, brown is for the doors (which open with the keys) and red for traps. Note that all traps do not have the same range of red: the higher the damage, the more intense the red.
Furthermore, two kinds of objects can be distinguished: weapons, represented

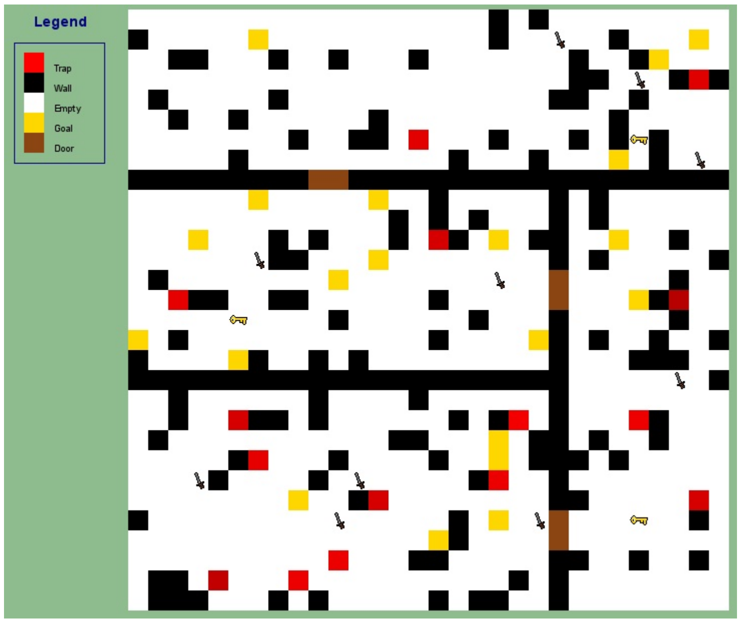


Figure 2: Graphical map representation.

by a sword, and the keys, which are used to open the doors. They disappear when the hero picks them.

One remarkable thing about the map is that it is divided into 4 different regions using walls and doors. These are the representation of the scenes, which are pre-established from a designed map since it was not the aim of the study to do an algorithm that separated scenes in an equitable way and distributed doors to get into them. So, doors are always in the same place and will disappear when the hero opens them by picking their corresponding key.

Realise that there is one scene which does not have any key. This means that this scene is the last one. In fact, he always starts from the upper scene and changes the scene through the doors.

It is crucial to keep the map updated. A very useful tool to do it is the group, which is not only used to group presentation shapes in order to control them all at once, but also it has dynamic properties that are able to execute a code every time the group is about to be redrawn.

In this case, every time the map is modified, either because a door opens or an object or goal is picked up, the code of the group *mapDrawing* shown in the figure 3 (which updates the visualization of the tiles) is executed.

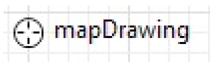


Figure 3: Variable in charge of keeping the map updated.

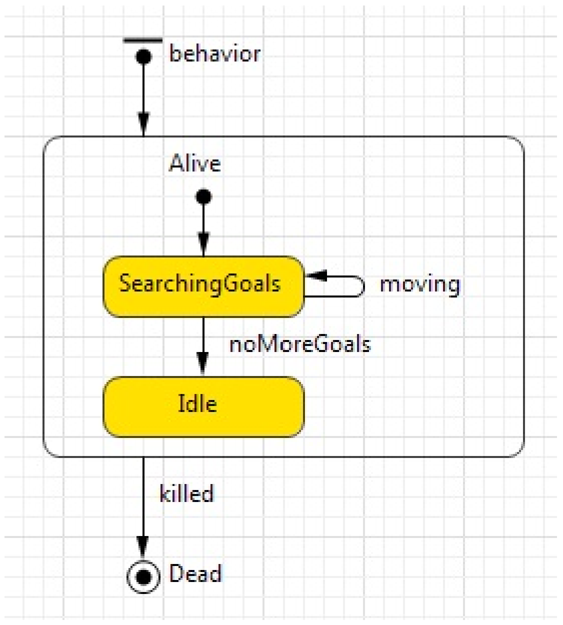


Figure 4: Hero behaviour statechart.

4.4. The hero

Once the environment where the agents move is created, it is easier to support the decisions they have to make reflected in the artificial intelligence. In the hero's situation, his behaviour is clearly defined by one of the Anylogic tools: the statechart.

As it can be seen in the figure 4, there are two possible stats: being alive or being dead. The killed signal is sent when the received damage is higher than the remaining life.

The behaviour when Dead is sending a signal to the *Main Class* in order to confirm that the game is over.

On the other hand, the behaviour when Alive is to move around looking for goals until all of them have been picked. Then, it stops and sends a signal to the *Main Class* to communicate the hero's success. Here appears the structure key for the agent's artificial intelligence: another matrix that counting the steps to make determines the fastest way for the hero to reach

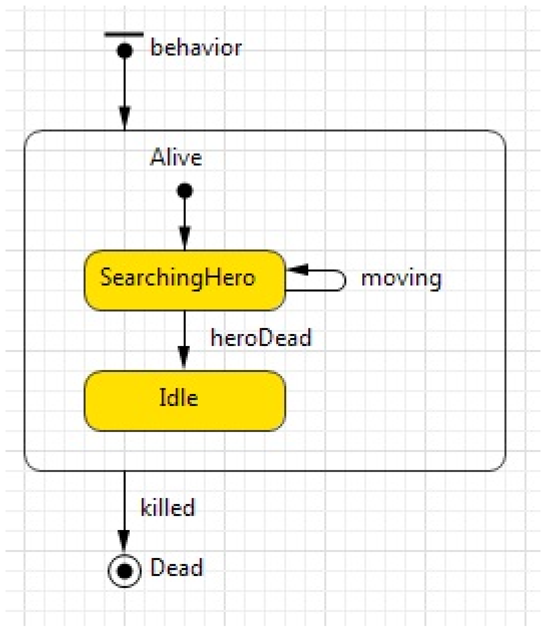


Figure 5: Villains behaviour statechart.

the target (goals or keys). However, if a villain is placed near him, the hero will attack. The algorithm chosen to fill this matrix is the BFS (breadth-first search), where the map is filled progressively from the position of the several points of interest.
Once a movement is made, the hero observes if the tile where he is standing has any special element (a trap, a goal, a weapon or a key) with which he can interact.
Obviously, the statechart is supported by some functions that perform the structural modifications and update the parameters of the hero.

4.5. The villains

As for the villains, an statechart is also used to describe their behaviour. This statechart is similar to the hero's one.
Two states, which are Alive or Dead, can also be seen and they have similar functions than the ones the hero has.
In this case, villains do not send any signal to the *Main Class* because if they are killed the game continues running, and if the hero dies he is the one that sends the signal when he realizes his health value is zero.
On the other hand, their only aim is killing the hero, so when he is dead,

they will stop moving. In order to simulate the hero's search, villains also have a matrix that shows them the way to reach him. This matrix uses the same algorithm as the hero, a BFS, but in this case the starting point is only one, the hero's position.
Moreover, unlike the hero, villains only interact with the hero and not with other elements, so they do not have to worry about traps or objects.
Also in villains case, there are some functions behind the statechart that modify parameters of the system.
It is worth saying that all villains follow the same behaviour, but if having more artificial intelligences was important, it could be done in two different ways: creating a new agent with his own statechart or creating a new states branch in the existing statechart. In this domain it would has more sense to do it in the second way, because although the behaviour would be different, in essence they would be the same agent with the same variables.

5. User's guide

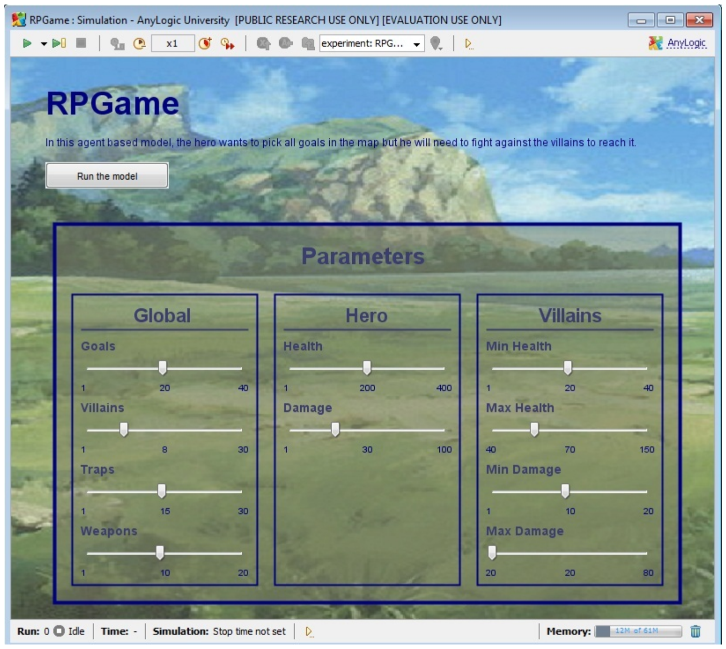
In order to make easier the use of this simulator it is worth to make an explanation about how to use it. The simulator has only two screens.

5.1. Starting screen

Figure 6 is the starting screen, where it is possible to start the simulation but also to change some parameters of it by using the sliders.
There are three kinds of parameters:

1. Global: Related to the map features, it is possible to define the number of goals, villains, traps and weapons of it.
2. Hero: These affect to the hero skills, specifically the health and the damage can be defined.
3. Villains: These affect to each villain skills. Unlike the hero, the health and the damage of each villain are determined by an initial random. The boundaries of these random are the values established on the sliders.

Once the parameters have been selected, the simulation is able to start by clicking the button which says "Run the model".

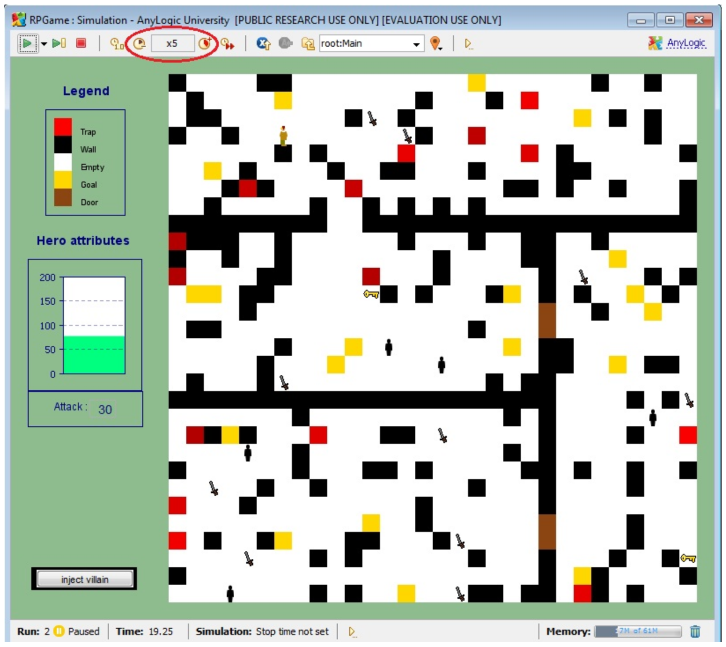

RPGame : Simulation - AnyLogic University [PUBLIC RESEARCH USE ONLY] [EVALUATION USE ONLY]

RPGame

In this agent based model, the hero wants to pick all goals in the map but he will need to fight against the villains to reach it.

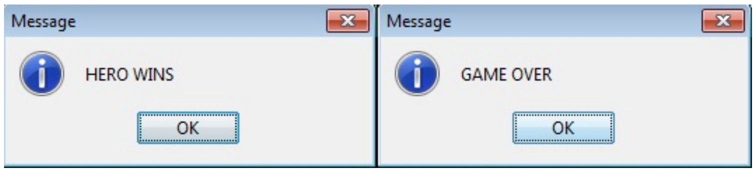
Run: 0 Idle | Time: - | Simulation: Stop time not set | Memory:

Figure 6: Starting screen of RPGame.


RPGame : Simulation - AnyLogic University [PUBLIC RESEARCH USE ONLY] [EVALUATION USE ONLY]

Run: 2 Paused | Time: 19.25 | Simulation: Stop time not set | Memory:

Figure 7: Main screen of RPGame.


Message

HERO WINS

Message

GAME OVER

Figure 8: Game over messages.

5.2. Main Screen

Figure 7 shows the RPGame main screen, where the experiment simulation can be followed.
Five different statements can be seen:

1. The map. As it can be seen, it is composed of the tiles, explained in other sections. Also it is shown two different kinds of agents: the hero, represented in light brown color, and the villains, represented in black.
2. The legend, explaining the meaning of every color on the map.
3. Hero atributes. Life is shown on a green chart that empties itself when the hero is attacked. The damage he takes is written below the chart.
4. "Inject villain" button. Adds a new villain on the map. He will appear in a random position.
5. On the top of the screen the simulation controls bar can be seen. Play, pause and stop buttons are placed on the left. Circled in red, the controls of the simulation speed: the left button is used to reduce the speed and the right one, to increase it. The other elements of the bar are not relevant.

When the game is over two different types of messages can appear: one that says "HERO WINS", or another one where the text "GAME OVER" is taped (figure 8). Depending on if the hero has picked all goals or if he has been killed before reaching them, one or the other will show up.

6. Comparison with related work

Anylogic offers a huge range of learning mechanisms or tutorials regarding the program. One of these learning tools is a range of examples mostly created by Anylogic's developers.
It would be worth to compare models that want to represent the same scope as the RPGame, but in this case, they do not exist. Two of the given

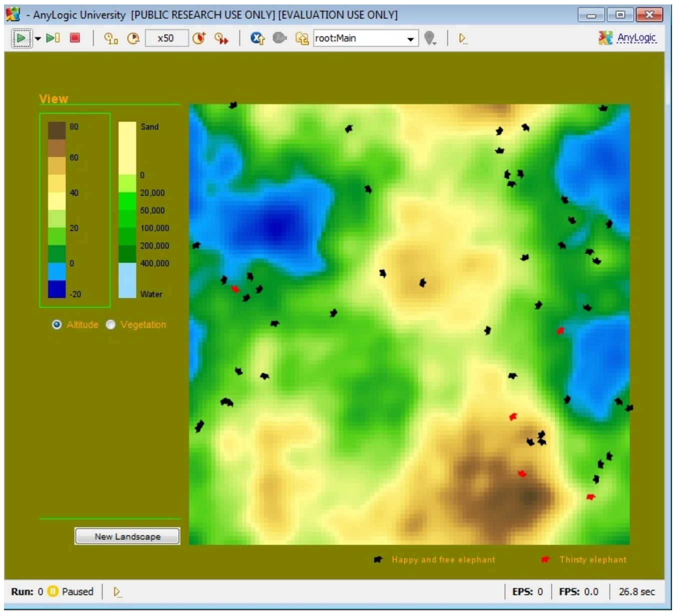


Figure 9: Main screen of Wandering Elephants model.

examples can be used as a reference, as they have some similarities with this work. However, it is complicated to know which one is better or worse, since each model represents something different.

6.1. Wandering elephants

This model is based in a 100 by 100 kilometers area where a number of elephants live, as it is shown in figure 9.
There are two layers of geographical information: altitude and vegetation. Elephants randomly wander in the area, but every once in a while they get thirsty and go to the nearest lake to drink water. As elephants walk, they demolish vegetation with a certain rate. Vegetation regenerates, but it can never exceed a natural maximum that depends on the altitude. A new landscape is generated each time the model is restarted.

Regarding the similarities, it is worth to mention that the environment in which is developed, although it is a 2D continuous, uses a 2D discrete structure to represent the map's information, as it happens in the RPGame model.
Furthermore, the map is modified because of the agents. In this case they are the elephants, who eat the vegetation, and in this study the agent is the

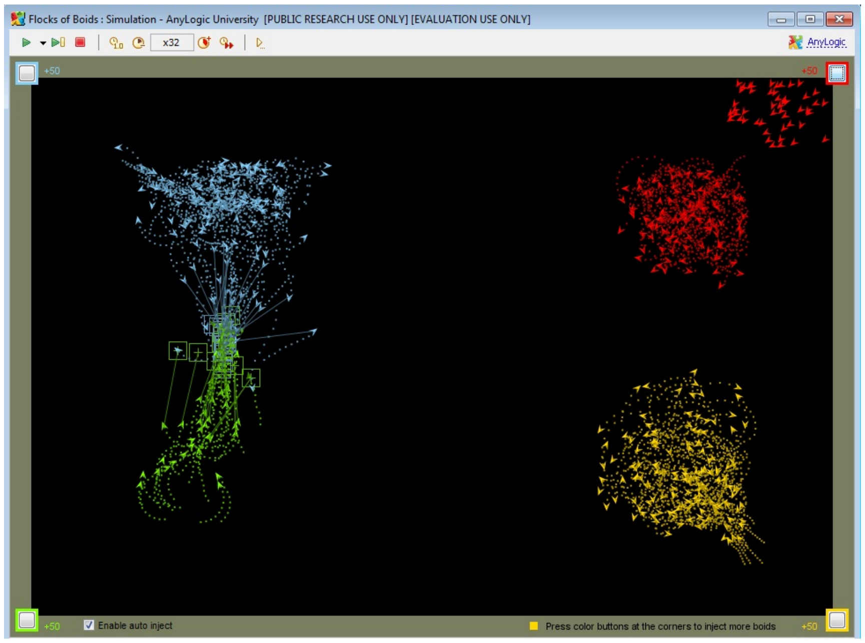

Flocks of Boids : Simulation - AnyLogic University [PUBLIC RESEARCH USE ONLY] [EVALUATION USE ONLY]
x32
AnyLogic
+50
+50
+50
+50
Enable auto inject
Press color buttons at the corners to inject more boids

Figure 10: Main screen of Flock of Boids model.

hero, who collects objects and opens doors.
On the other hand, the agents are distributed randomly, as it happens in an RPGame. Needless to say, artificial intelligence has nothing to do with it. Although the elephants' behaviour is similar, the behaviour is random because there is not a structure that tells them what to do.

6.2. Flocks of boids

In this agent-based model (figure 10) boids follow the flocking rules (move towards the centre of mass of the flock, align individual speeds, avoid collisions), and also fight and kill boids of the other flocks (of different colors).
This model has a more sophisticated environment, since it is a 2D continuous environment in all aspects. However, the agents are not influenced by the environment and they do not modify it either. On the other hand, agents have a minimal artificial intelligence but it is not determined by a statechart. These agents also attack themselves and it is represented graphically, as it is shown in the squares on the image. Furthermore, they can eliminate themselves like the hero and villains.
This model also has an interface that allows users to change some parameters

of the simulation, as it happens in an RPGame.

7. Conclusions

In this study a an RPGame model has been developed and, as it can be seen, Anylogic is the perfect tool to create agent-based modelling. So, a model like this one can be created easily from any domain. On the other hand, Anylogic allows the user to use statecharts, which has many advantages when generating the agents' behaviour.
Moreover, Anylogic has a really useful help tool, a great amount of examples and written or recorded tutorials. So, if someone with no previous Anylogic knowledge can create his own model easily.

8. Future work

This project's extension has been limited by a temporal boundary. However, in modeling, the limits are to be established by the developer as the model can always be improved. Regarding the RPGame, the following ideas could improve some of its features:

- It could be switched to 3D but the environment would have to be continuous, and therefore, the tiles would need to be relatively delimited. For instance, the damage that traps inflict would have to be time-based as there would not be a step-by-step temporal notion.

- The AI of the villains could be improved, so they would be able, for example to surround the hero being that one the best way to terminate him.

- An algorism could be developed in order to dynamically generate scenes instead of the current pre-designed walls. It would have to be taken into account that the number of keys and doors has to be the same, and that they have to be related in a judgmental way. Besides, the location of the keys would have to follow a distribution such that the hero can open them in a specific order.

- Add more editable parameters such as the number of scenes, the size of the map, the damage inflicted by the traps and so on.

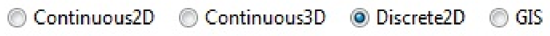

Continuous2D Continuous3D [✓ Discrete2D] GIS

Figure 11: Possible kind of environments in Anylogic.

- Add a graphic representation of the fights between the hero and the villains.
- Include control buttons so the gamer can command the hero, instead of having AI-controlled movement. This improvement would turn the experiment into an actual game.
- Add more complexity to the RPG, despite the fact that this addition would change the represented domain.

Besides all these modifications to the RPG, an aspect from Anylogic that could be improved is the lack of ability to create a discrete 3D environment, as it can be seen in the figure 11.

Switching from continuous to discrete should not mean a great deal, as it only implies a restriction of liberties and, therefore, it aborts conflictive situations. Implementing this option would improve the choice of environment-defining alternatives that Anylogic offers.

9. References and Bibliography

[1] Gilbert, N. (2007) Agent-Based Models. SAGE Publications, num 153, p.1-20

[2] Scholarpedia,*AnylogicWebsite*, http://www.anylogic.com/

[3] Anylogic, *ScholarpediaWebsite*, http://www.scholarpedia.org/article/Agent-based-modeling