Simulation-based performance evaluation of PacMan NPC's

Can Babek Ilgaz
University of Antwerp

Abstract

Statecharts give an abstract description of the behavior of a system. We use this behavior with events which are analyzed and represented. These events can occur in one or more possible states. Statecharts allow us to to do more things with less coding and especially in games. Today users expect more realistic behaviour from Non Player Characters (NPC's). If we try to create a realistic NPC with pure coding it will take so much time. Also this technique may lead much more errors which we can not predict before start coding. In this paper, I show how the simulation-based performance evaluation of NPC's in PacMan game.

Keywords: statecharts, modelling, artificial intelligence, game development

1. Introduction

In almost every scenerio, users play the game for wining and somehow system should prevent user to win and for this almost all games have Non-Player Characters (NPC's). They are really important part of a game and it is really hard to make them more realistic. Today in all most all video game genres revolve almost entirely around interactions with non-player characters. In more developed games NPC's and user controlled character can have so much interaction at the end they provide a reference point to the controlled character's relationship with the game. This happens mostly with dialague and true dialogues with NPCs are usually very complex. First there are so many combinations to code. Than there are so many things to solve because many errors that might occure. Lastly maybe the most important they should be smart beacuse usually people does not want to win games so easily so somehow NPC's should predict what user might do and try to push

them to their limits but also it should not be so easy to lose for user as well. Keeping this balance is really hard work.

In Programmed Graph Rewriting with Time for Simulation-Based Design[1] it is shown that how DEVS can be used to dexcribe complex control structures for programmed graph transformation and also can be used as modelling of player behaviour, incorporating data about human players' behaviour and reaction times.

In Programmed Graph Rewriting with Time for Simulation-Based Design[1] it is stated that there are five distinct elements in PacMan game. PacMAN, Ghost, Food, GridNode and Scoreboard. PacMan, Ghost and Food objects can be linked to GridNode objects and finally Scoreboard object holds an integer valued attribute score.

DEVS models can be either atomic or coupled. An atomic model describes the behaviour of a reactive system and a coupled model is the composition of several DEVS sub-models. It means that a coupled DEVS is a composition of atomic or coupled DEVS sub-models. These sub-models should have ports and connect through channels. Ports can be *input* or *output*. Channels go from an output port of some model to an input port of a different model, from an input port of a coupled model to an input port of one of its sub-models, or from an output port of a sub-model to an output port of its parent model. Finally by using ports and channels our model can send and receive signals between other models.

In Programmed Graph Rewriting with Time for Simulation-Based Design[1] User block is declared as coupled DEVS and it is responsible for user interventions to Controller block which is an atomicDEVS. This block encapsulates the coordination logic between the external input and the transformation model. And Controller is connected with User Controlled Rules and Autonomous Rules. These rules make sure that the game is logically true.

In my project I used Statecharts and with statecharts we could model every components by an appropriate model. At the end this will lead us to design game's logic at a higher level and not concern about detail implementations. Maybe the most importantly modeling is visual and it is really easy to describe our behavior flow. The main advantage of modeling is that a designer could go a step back and see the big picture like stopping at any intermediate level and rating how good the transformation is.

My project is designing PacMan game wtih statecharts. Like DEVS statecharts make it really easy to modelling all objects and transactions. In my PacMan game there are apples in almost tile and in some tiles there are ob-

stacles which PacMan can not pass. So PacMan should go some other tiles instead of tiles which constain obstacles. And at last there are two Ghosts which wants to eat PacMan. PacMan's goal is eating all the apples before get eaten by Ghosts and Ghosts goal is eating PacMan. PacMan can be controlled bu user but if in 2 seconds there is no input PacMan starts to move itself randomly. After that PacMan can be controlled as well. In default mode Ghosts move randomly but checks in a 3x3 area if PacMan is around. If one of them can see PacMan it starts to chase it. If Ghost is able to catch PacMan, PacMan gets eaten and Ghosts win the game.

I designed statecharts just for PacMan and Ghosts. All of them have two different states. Untill their states changes for Ghosts this means able to see PacMan in a 3x3 area and for PacMan not receving an input from user they stay on their first state. In every 0.3 second Ghosts goes to a random tile but if they see PacMan in a 3x3 area they switch to another state and start to chase it. Untill PacMan gets eaten or gets disappeared. Than they switch to their first state and move randomly. For Pacman also there is just two states but there are more transactions in these states because every key has its on transaction as well. Buttons to control PacMan are i for going up, j for going left, k for going down and l for going right. If the up key which is i is pressed up conection runs and if left key which is j is pressed left transaction runs. Unless there are no input is given for two seconds than PacMan starts to move randomly and soon or later PacMan gets eaten because Ghosts are smartter than PacMan.

Section 2 of this paper describes how the modelling of game AI is approached, and gives a concrete implementation by designing the AI for Ghosts and PacMan in the setting mentioned above. Section 3 explains how the Game is developed. Section 3 is about Simulation Experiments done. Finally, I come to a conclusion in section 5.

2. Modelling the statecharts

As mentioned before the behaviour of Ghosts' are modelled according to PacMan. More specifically, in this system there are two Ghosts and 1 PacMan. Ghosts' goal is to eat PacMan. Unless they see PacMan they move a random tile in every 0.3 second. If they see PacMan they start to chase it. PacMan has two states as well. It can be controlled by a user or it can move around randomly like Ghosts.

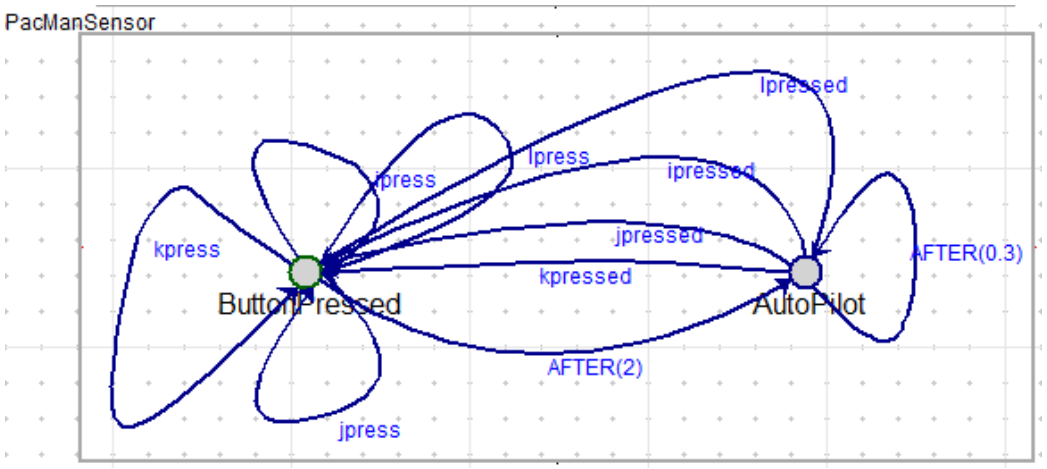

Figure 1: PacMan State

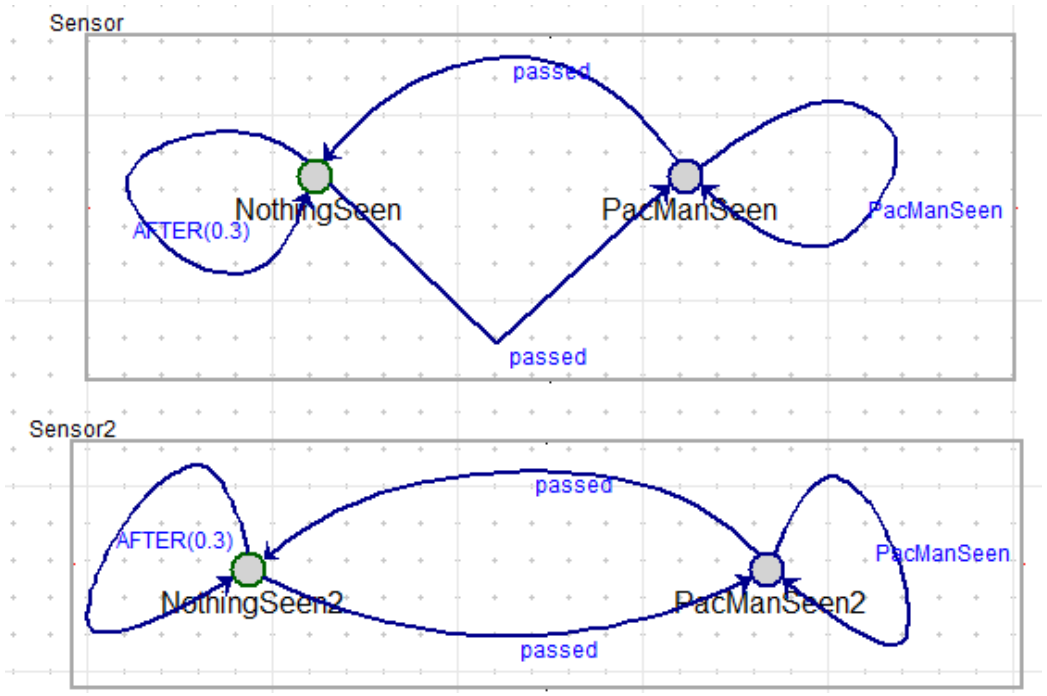

Figure 2: Ghost State

PacMan needs to eat all the apples on tiles to win the game and for Ghosts they should eat PacMan for winning.

3. Developing the game

I used statecharts and python while developing this game and the statecharts themselves are made in atom3.

Statecharts should be updated and all events executed in every 30 milliseconds this means every pass of the game loop in every 30 milliseconds and if it does not happen the game will wait for the statechart to be finished which will give the impression that the game hangs. This was not encountered during the development of this encounter though. While the events are propagated, the necessary code is executed.

First I started to model ghosts. Each ghost should move randomly around the canvas in every 0.3 seconds and in the mean time they should not dissapear from canvas and not be able to pass through Obstacles. By the mean time they should not stuck in a position and wait the code to say them to go another direction to move and look for PacMan and if they see it in a 3x3 area they start to chase and eat it. In my PacMan game there are two ghosts but their models are all same. This gives a advantage modelling Ghosts because we can add Ghosts as many as we want. For Ghost1 its first state is NothingSeen and every 0.3 seconds Ghost1 goes to a random direction and checks if PacMan is around. Untill it detects PacMan it just moves randomly. After seeing PacMan its state changes into PacManSeen. In this state in every 0.3 second it moves into a tile which is in the direction of PacMan. If PacMan is able get escape or gets eaten Ghost1's state changes back to NothingSeen and start to move around randomly. For Ghost2 it is the same scenerio. Ghost2's first state is NothingSeen2 and every 0.3 seconds Ghost2 goes to a random direction and checks if PacMan is around. Untill it detects PacMan it just moves randomly. After seeing PacMan its state changes into PacManSeen2. In this state in every 0.3 second it moves into a tile which is in the direction of PacMan. If PacMan is able get escape or gets eaten Ghost2's state changes back to NothingSeen2 and start to move around randomly.

For PacMan, first user should be able to move it, also PacMan can not leave canvas, can not go to a tile where there is an obstacle like Ghosts, should be able to eat apples and if it goes to a same tile where one of the ghosts it should lose the game. If user does not give any input PacMan's state changes

and starts to move randomly untill receive another input. PacMan also has two states. First ButtonPressed. ButtonPressed has four transactions which are all connected with itself. For each different button it has different transactions. For up key which is i it runs a different transaction and for left key which is j it runs a different transaction. Unless for two seconds if there is no input from the keyboard PacMan's state changes into AutoPilot and starts to move randomly in every 0.3 seconds untill another key is pressed. PacMan's score can be seen on command prompt. In total PacMan can reach 910 points. Every apple is just 10 points and whenever PacMan eats an apple its score incremented by 10.

3.1. Possible improvements

A lot of possible improvements can be made on this project. Firstly, I did not try to write a code that can be understand clearly and easily so refactoring is needed. Also it does not have an design standards. If extensions are to be made, the code needs to be cleaned first and some repatitive lines must be taken care of. Furthermore there is still too much logic present in code. There are some aspects that can and should be specified in statecharts. The only code present should be a general game engine and the algorithms needed by the specifications made in the statecharts.

One of my main mistake was I started coding immediately not designing the statecharts. This was a problem because subconsciously I was adapting my statecharts to my code and it should be the other way around. It would have prevented bugs, made development time shorter and prevent to code this much line of code.

It is also easily possible to add new behaviour to the game. For example ghosts may develop a strategy to catch PacMan, they might try to surround it from different directions or can be used Different levels of abstraction like Memory, Strategic Decider or Tactical Decider. These additions only need new statecharts.

4. Simulation Experiments

One of the most important thing on games is playability, games would make sure player can not win so easily and also lose so quickly. This means playability is measured by quantity or duration that a game can be played and it is a important part of the quality of gameplay. So the playability of PacMan game depends on the right choice of the response time of the

Ghosts. For getting a decent value I did lots of experiments. If Ghosts move so quickly there is almost no chance to win the game or if they move so slow User will always win the game.

4.1. Modelling User Reaction Time

I used the reaction time which is given on Programmed Graph Rewriting with Time for Simulation-Based Design. The reaction time distribution can be described by an asymmetric normal-like distribution. The cumulative distribution function of frequencies for sensorimotor human reaction time is:

$$F(x) = e^{-e^{\frac{b-x}{a}}}$$

where **a** characterizes data scatter relative to the attention stability of the subject: the larger **a** is, the more attentive the subject is; **b** characterizes the reaction speed of the subject. For mimulation purposes, sampling from such a distribution was done by using the Inverse Cumulative Method and i used it as it is.

In Programmed Graph Rewriting with Time for Simulation-Based Design, four types of users were tested: Slow with $\mathbf{a = 33.3}$ and $\mathbf{b = 284}$, Normal with $\mathbf{a = 19.9}$ and $\mathbf{b = 257}$, Fast with $\mathbf{a = 28.4}$ and $\mathbf{b = 237}$, Very Fast with $\mathbf{a = 17.7}$ and $\mathbf{b = 222}$. The parameters used were those of four examples subjects.

4.2. Simulation Results

For the simulations, I changed Ghost's speed netwwen 0.1 second and 0.4 second. As the ghost's become faster it becomes so hard to win the game. But the common thing in all these speed values is the lomnger the game lasts the slower the Ghosts get. This is because the user has more time to move the PacMan away from the Ghosts. An explanation for this behaviour is simply that after a certain point, the Ghosts desicion time is too low and and the user always wins unless user does not control PacMan and PacMan rondomly goes to Ghosts. I decided that the Ghosts speed must be 0.3 seconds because I wanted to make the game for the user more challenging but also able to win.

Figure 3 presents the simulation results of the game. I played the game 10 times in different and this value taken on the x axis. On the y axis you can see the timing results. These results are just time results of the game how long it iasted. At 0.1 second and 0.2 second simulation PacMan could

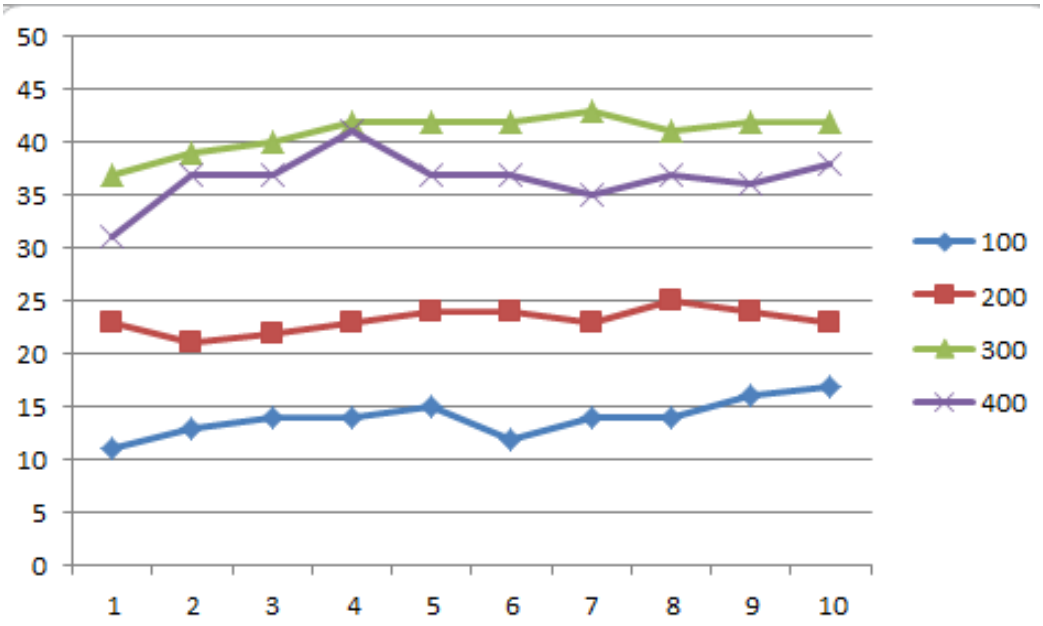

Figure 3: Time Till End

not win any game. Because reaction time ofghosts were too fast but in 0.3 second and 0.4 second PacMan started win the game. I never let PacMan to move around on its own. Avarage time of 0.1 second is around 14 seconds. Avarage time of 0.2 second is around 23 seconds. Avarage time of 0.3 second is around 42 seconds. Avarage time of 0.4 second is around 37 seconds. At 0.1 and 0.2 second Ghosts' response time is so quick it is almost impossible to win. After 0.3 seconds it starts to get easier to win the game. on 0.4 avarage game ending time is 37 seconds. As you can see wining time starts to decrease that is because ghosts does not react to PacMan's moves as they used to in lower response times.

4.3. Simulation Results

For winning scale it was as it showned in Figure 4. X axis is victory frequency of wining game and y axis is ghosts' reaction time. As we can see when the response time of ghosts get slower winning becomes easier. In this simulations I never let PacMan to move by itself as well.

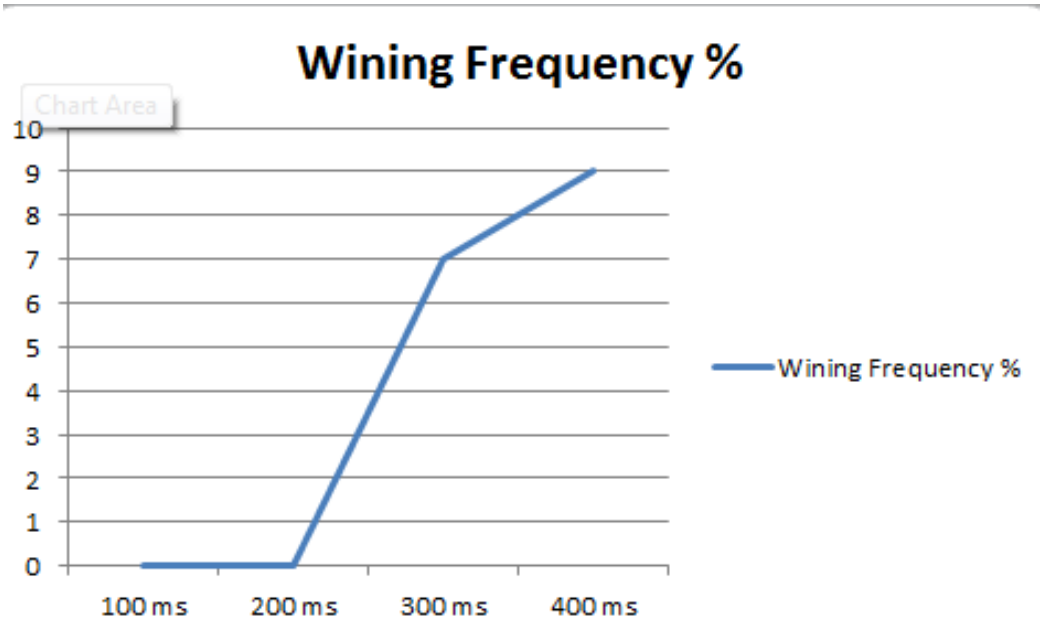

Figure 4: Time Till End

5. Conclusion

In this article, I described the use of Statecharts Simulation-based performance evaluation of PacMan NPC's. Statecharts definetely helps to design and code NPC's. I used statechart to generate various behaviors. I started by defining all components as simple statecharts. Then using model transformation, I combined those statecharts to difierent orthogonal components in a composite state. Of course the transformation could be done in a various ways. In my case PacMan and the Ghosts are almost the same, except for the controlling the PacMan. Also we can add as much as ghost we want without any other extra work.

Using statecharts to specify the behaviour of NPC's works very well. As we can see Ghosts have the same statecharts so it is obvious that components are very reusable. In my case, Ghosts and PacMan are the same, except for PacMan can be controlled. This means that every part can be left alone, except for direction key transactions.

For a programmer it is just needed to implement specific algorithms and because of this some behaviour is actually simpler to implement using statecharts than code. By using statecharts, we can, on a high level describe what

actions need to be taken to achieve a certain goal. All in all statecharts prevents bugs when certain components need to work together, greatly reduces the complexity of the needed structures in code and non-programmers can use this technique. A game designer may not know the algorithm and still can decide which state when should be used.

Eventhough Programmed Graph Rewriting with Time for Simulation-Based Design was about modelling PacMan with DEVS it really helped me to design PacMan with statecharts. I had the same positive experience using statecharts and it seems they are really helpful in this kind of context.

References

[1] Eugene Syriani and Hans Vangheluwe. Programmed graph rewriting with time for simulation-based design.