# Week 1 (02/10/2023 – 09/10/2023)

Got introduced to the specification and started thinking of the way to approach the project

Had the first meeting with supervisor.

# Week 2 (09/10/2023 – 16/10/2023)

Wrote the specification and got a few suggestions from supervisor and implemented them. Added a few references that can be found in the specification.

# Week 3 (16/10/2023 – 23/10/2023)

End of first sprint. Made a sample automata and implemented it in python, using graphviz to get a picture output. Met with supervisor and thought about goals for the second sprint, those goals being:

1. Adding a visualisation aspect to the project
   1. A slider for the user to decide how “good” or “bad” the NPC should be which the program takes as input
   2. Put the DFA picture and the path it took in the visualisation(?)
2. Expand the DFA, maybe add a couple of new pictures
3. Do the randomisation outside the DFA, similar to a “seed” where this seed will decide on the input but regardless of it the input is always constant to keep the determinism of the DFA.

Read a few papers:

1. Mentions that finite state machines are widely used in the gaming industry
   1. Fathoni , K., Nurhadi, H.A.T. and Hakkun , R.Y. (2019) *Finite state machines for building believable non-playable ... - iopscience*, *IOPscience*. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1577/1/012018 (Accessed: 23 October 2023).
   2. Syahputra, M.F. *et al.* (2019) *Historical Theme Game Using Finite State Machine for Actor Behaviour*, *Historical Theme Game Using Finite State Machine for Actor Behaviour - IOPscience*. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1235/1/012122 (Accessed: 23 October 2023).
2. Talks about the FSM AI method and how it helps makes NPCs more believable.
   1. Jagdale, D., 2021. Finite state machine in game development. *algorithms*, *10*(1).

# Week 4 (23/10/2023 – 30/10/2023)

Added python visualisation and a “goodness” slider. This slider will be a deciding factor on the traits the NPCs will get.

A randomisation heuristic was made to instil the sense of liveliness and differences between people and how no two similar ones are the same.

Added a couple extra traits to increase the diversity of the NPCs with the goal being having possibly 10 traits to increase number of possible characters to 1024

References

1. Talk about how NPCs are built in a world and how our approach is trying to mimic that and how it is doing it
   1. NPCs! | Running The Game (2016). 5 October. Available at: https://www.youtube.com/watch?v=NwJxM1ABLJM (Accessed: 02 November 2023).

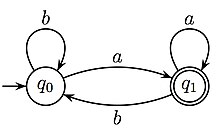
# Week 0 (01/01/2023 – 08/01/2023)

Got report mark (65%). Need to make the project more complicated to reflect a third year module. The recommended method was by using historical deterministic automatons. When looking into at I found several papers that talk about historical deterministic timed automatons in games. Essentially History deterministic finite automatons can solve any non-determinism in them by looking at the run thus far, which could be a way of us making the automatons more non deterministic (in turn, powerful with less states to use) the longer we run it as we will have more information. Two questions are 1. How we can do this? 2. Is the python module we use support that? My answer is No and that we will have to implement it manually which may involve me making my own module with the previous module used as a reference or abstracting it so even if we make HDTA (Historical deterministic timed automatons) we make the actual DFA in background which doesn’t increase the computing speed of the program at all. More time will have to be put understanding **how** those automatons work and how they can be implemented.

**ω-automaton:** A variation of finite automata that runs on infinite, rather than finite, strings as input. Examples are **Buchi Automaton, Muller automaton and others.** All instances of it can represent ω-regular languages except Deterministic Bucho Automaton (CoBuchi) which is strictly weaker.

**ω-regular languages:** a class of ω-languages that generalize the definition regular languages to infinite words.

**Buchi Automaton:** It is a theoretical deterministic or non-deterministic machine which either accepts or rejects **infinite** states. Such a machine has a set of states and transition function, for the deterministic version there is only one path for each transition function, while for the non-deterministic one there are multiple outputs leading to many possible paths for the same input. The deterministic machine accepts an input if and only if it will pass through an accepting state **infinitely** many times. For example the figure below, as it reads the input where is the number of repetitions it accepts it, while if it only reads it will not accept it as is the accepting state and it only passes through it a finite number of times. While for non-deterministic versions it accepts an infinite output if and only if some possible path is accepting.



**Difference between timed automata and finite automata**: In a finite automaton, at some point of the execution, the state is entirely described by the number of letter read and by a finite number of possible values, which are actually called "states". That means that, given a state and a suffix of the word to read, the remaining of the run is totally determined. Thus, the word "finite" in the name "finite automata". However, as it is explained in the section "run" below, in order to resume clocks are used to determine which transitions can be taken. Thus, in order to know the state of the automaton, you must both know in which location you are, and the clock valuation.

**IDEA: Why not make every character has its own infinitely running finite automata (Buchi) and its state well change depending on the environment + some randomness**

**Question: How am I going to use the fact that history-deterministic timed automatons can have non-determinism while still knowing exactly the next step using its previous states? (prime multiplication?)**

References

1. Historical deterministic Timed automatons, how to use them and the math behind them
   1. Thomas A Henzinger, Karoliina Lehtinen, Patrick Totzke. History-Deterministic Timed Automata. CONCUR, 2022, Warsaw, Poland. ff10.4230/LIPIcs.CONCUR.2022.14ff. ffhal-03849404f
2. EXPtime
   1. Papadimitriou, Christos (1994). Computational Complexity. Addison-Wesley. ISBN 0-201-53082-1. Section 20.1, page 491.
3. Timed automaton
   1. Rajeev Alur, David L. Dill. 1994 A Theory of Timed Automata. In Theoretical Computer Science, vol. 126, 183–235, pp. 194–1955
4. Buchi automaton
   1. Büchi, J.R. (1990). On a Decision Method in Restricted Second Order Arithmetic. In: Mac Lane, S., Siefkes, D. (eds) The Collected Works of J. Richard Büchi. Springer, New York, NY. https://doi.org/10.1007/978-1-4613-8928-6\_23
5. Muller Automaton
   1. D. E. Muller, "Infinite sequences and finite machines," Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design (swct 1963), Chicago, IL, USA, 1963, pp. 3-16, doi: 10.1109/SWCT.1963.8.
6. Omega automaton
   1. S. Safra, "On the complexity of omega -automata," [Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, USA, 1988, pp. 319-327, doi: 10.1109/SFCS.1988.21948.
7. Omega regular lanaguages
   1. Wolfgang Thomas, "Automata on infinite objects." In Jan van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, pages 133-192. Elsevier Science Publishers, Amsterdam, 1990.
8. Hybrid Automata
   1. T. A. Henzinger, "The theory of hybrid automata," Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, NJ, USA, 1996, pp. 278-292, doi: 10.1109/LICS.1996.561342.
9. Reachability Problems
   1. Giorgio Delzanno, Igor Potapov (Eds.): Reachability Problems - 5th International Workshop, RP 2011, Genoa, Italy, September 28–30, 2011. Proceedings. Lecture Notes in Computer Science 6945, Springer 2011, ISBN 978-3-642-24287-8

# Week 1 (08/01/2024 – 15/01/2024)

Trying to implement a more general History Deterministic Timed Automaton. Managed to implement a generic automaton where I don’t have to write every state out/do it manually and instead I gave a list of tuples with each tuple having two elements being opposing traits. The following is the shape obtained, with the loop added to represent the fact that the plan is to make it run forever or until something happens to the character (or the machine can no longer handle it)

A screenshot of a computer

Description automatically generated

# Week 2 (15/01/2024 – 22/01/2024)

Made the handling of the traits more generic and made it dependent on environment variables to fix the issue of the states having to be declared in the class and not the \_\_init\_\_ method. Next step is 1. Generative AI to decide on the examples of traits as I can’t possibly have examples for all possible traits 2. Implement multi processes i.e. run multiple automatons at the same time after redirecting them to loop, with the loop changing their traits according to the environments