Final Writeup

By: Ryan Neumann

Abstract

For my semester project, I applied deterministic finite automata to the classic game of PacMan. Specifically, I created my own version, which transitions the state of each ghost, depending on the current placement and action of PacMan. Simply, there are only four states that the ghosts can be. Each of which are accepting states. The ghost's can be wandering, searching for a path, chasing PacMan, or running away from him.

Introduction

Growing up, PacMan has always been one of my favorite games. It was instantly a hit game when it was released in 1980, and continues to be a cornerstone in the gaming community. Although it seems like a simple game, the logic behind it presents a challenging scope of options that were carefully selected to create an AI that feels almost too responsive. However, that is all a part of what makes PacMan so encapsulating. My influence to create this game came from my love of PacMan, gaming, and the knowledge you have provided me with. As you were also my professor for Artificial Intelligence as well as Software Development I, I found this project to be a perfect combination of all the lessons you have taught me over the years. The program was written in Java, including the AI for the ghosts. Contrary to the simplistic user interface, the AI behind PacMan proved to be a challenging concept to code.

Detailed System Description

There are four different states that the ghost assumes, either wandering, chasing PacMan, running from PacMan, or searching for a path. While PacMan is not on the same x or y axis as the ghosts, he wanders around the map. Once he is on the same x or y axis, the ghosts begin searching for a direct path to which they can attack PacMan. Finally, once they have found their path, the ghost begins to chase PacMan, following him at every turn. Initially, each ghost is able to attack PacMan, ending the game. If PacMan manages to get to the candy before being eaten, the ghosts then turn in to blue ghosts, allowing PacMan to eat them. If PacMan is able to successfully eat every power pellet, which are the dots in each space available on the map, the user has won the game.

Requirements

There are not many physical requirements of the system resources. PacMan is a very lightweight applications that basically runs on a system of yes and no, or better yet, 1s and 0s. The biggest consideration is when events are happening simultaneously between PacMan and the ghosts. If PacMan continuously crosses the x or y axis of a ghost, it will quickly transition between states of finding a path and chasing PacMan. This can take a major toll on the frames during the game, however it is a rare occurrence that such a situation would take place. Another thing to consider would be the images that are constantly being pulled to run the game. As a

simple solution, I assigned this images to variables once, which are used over and over again. However, it still has an effect for loading different images constantly. Most noticeably would be PacMan, as he needs to transition between an open and closed mouth throughout the entirety of the game.

Literature Survey

With PacMan being an extremely popular game, there have been many different versions over the years. With constant changes and improvements, almost every aspect of the game has been covered. Although many people talk about the ideology behind the game, not many people public release a project with it. The AI centers around PacMan, however, it is up to the coder to determine when these transitions should take place. Generally, people discuss PacMan as being a series of cause and effect moves. However, that is laymen's terms for us. To a computer scientist, that is a call for deterministic finite automata.

User Manual

When the user begins to play PacMan, each ghost starts at q0, wandering around the maze. This occurs until a ghost is on the same x-axis or y-axis as PacMan. At this point, that individual ghost would transition to q1, finding a path. As the ghost finds it path, it determines what the most direct way to reach PacMan would be, reconfiguring at every move. If the ghost loses sight, they transition back to the wandering state, q0. If the ghost has a clear shot towards

PacMan, they transition to q2, the chasing state. At any moment, if PacMan eats the candy, all ghosts begin to scatter, running away from PacMan.

The directions for the game are fairly simple. The only controls are the arrow keys: up, down, left and right. No other controls are needed to play. The user is in control of the PacMan character, while each ghost is controlled by the AI. The objective is to eat every power pellet without being eaten by a ghost. If successful in this objective, the user has won the game and it immediately begins a new one.

Symbol Table

S - PacMan Spotted

L - PacMan Lost

C - Candy Eaten

X - PacMan on same x-axis

Y - PacMan on same y-axis

Ghost Transitions

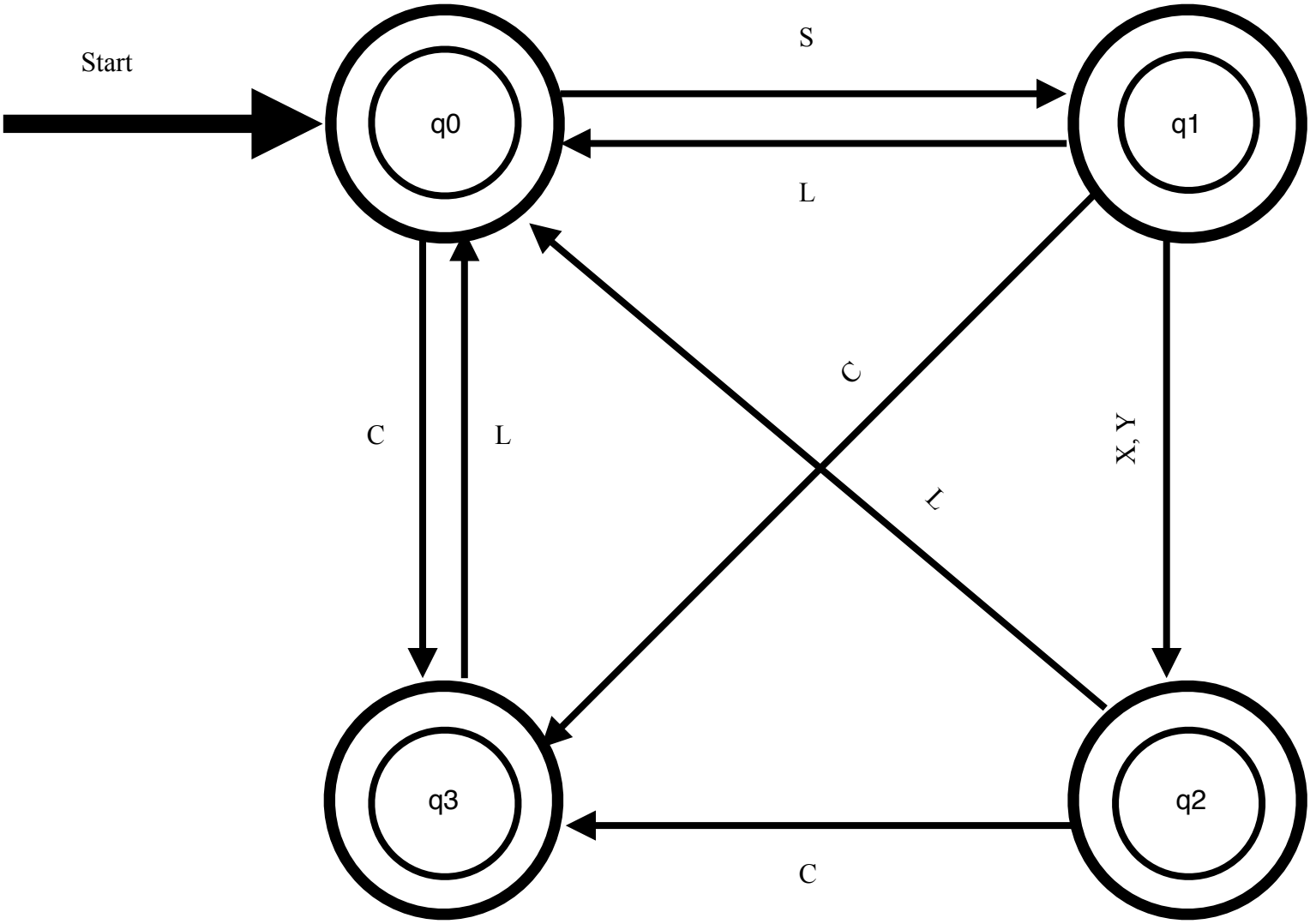
q0 - Wander the maze

q1 - Searching for a path

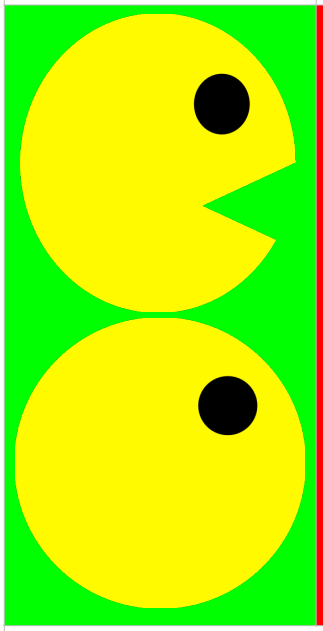
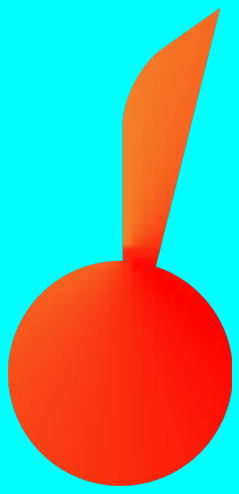
q2 - Chasing after PacMan

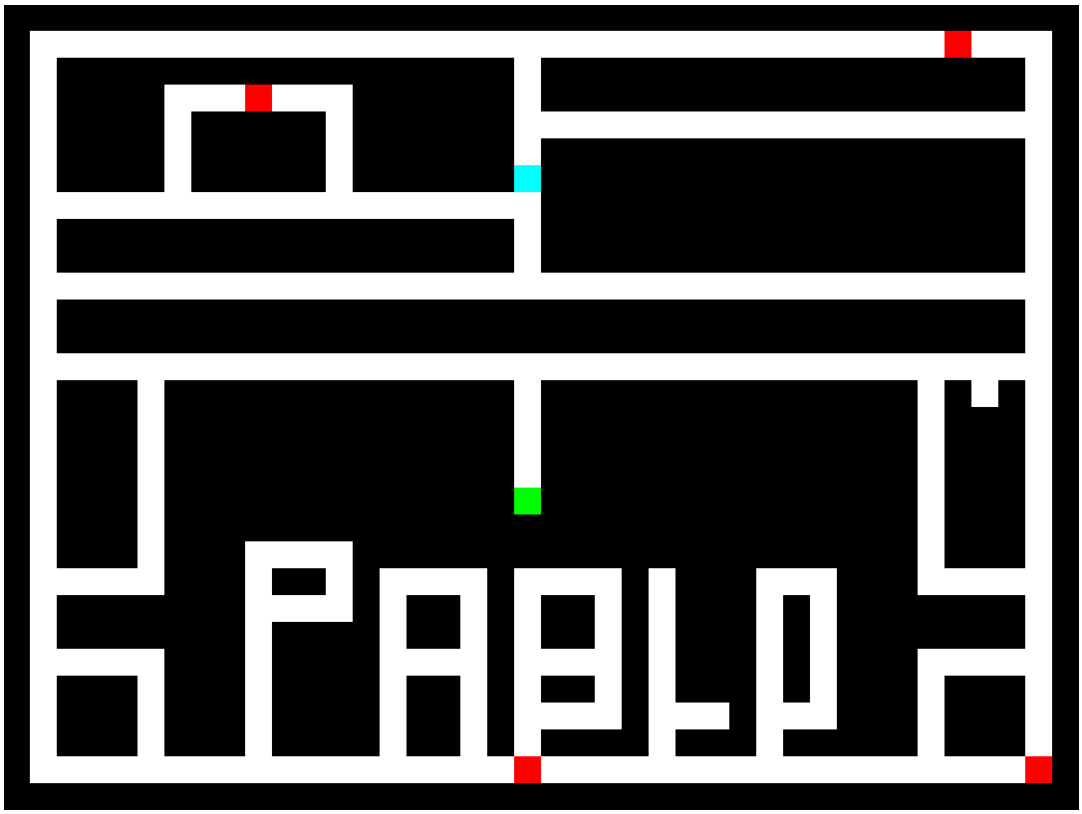
q3 - Running away from PacMan

DFA



Design

PacMan (Open/Closed)	Ghost (Attack mode/ Run Mode)	Candy	Power Pellets
			



Conclusion

Essentially, it is a big loop that creates the game of PacMan. With only four states, PacMan provides a surprisingly rich experience, that has near infinite playability. Not only that, but PacMan has also been an influence to other popular games, such as Donkey Kong, Super Mario Bros, and Sonic the Hedgehog. For such a simple game, it had such a significant impact on many generations. I found the challenge of coding the game enjoyable, as well as the knowledge I have learned from it. PacMan being one of my favorite classic games, it gives me great pleasure in knowing I have skillset to recreate such a masterpiece.

References/Bibliography

http://web.cs.ucdavis.edu/~rogaway/classes/120/spring13/eric-dfa.pdf