CIS 472 Final Report

Connor Phillips
Virginia Military Institute

Introduction

Throughout the first eight weeks of my Independent Research, I have worked with multiple fields of computer science including: database programming implementing SQL statements within Java, enhancing my JavaFX GUI skills, practicing with multiclass programming, multithreading, and an introduction of artificial intelligence. I have been able to use knowledge I am currently learning in CIS-313 to improve the efficiency of the engine. Because of this, many methods implemented have a constant or linear running time. The main focus of the first eight weeks has been the development and testing of a game engine that will be used in the second eight weeks in the implementation of artificial intelligence search algorithms.

Weeks 1-2

The first two weeks of this research were used as a refresher to threading/multithreading and JavaFX GUI design. I read about threading and multithreading to familiarize myself with the main concepts and ideas. This proved instrumental to the work done in later weeks. I also practiced implementing JavaFX GUI using practice questions from the Java textbook by Liang.

After reviewing threading and multithreading, I familiarized myself with basic SQL queries and statements. I downloaded MySQL onto my computer and used the terminal command line to run my personal database. With this, I created a working database called 'javabook'. Soon after, I read chapter 34, "Java Database Programming," and began working on exercise questions (Liang 2019). Many of the exercise programs I worked on involved creating JavaFX GUIs to create, insert, delete, edit, and view tables/data within the 'javabook' database I created.

While I was working on the database programming implementation, I also was reading two research papers to learn about the independent project. First, I read Dr. Lasisi and Robert Dupont's paper, "ALP4AI: Agent-based Learning Platform for Introductory Artificial Intelligence." Next, I read John DeNiro and Dan Klein's paper, "Teaching Introductory Artificial Intelligence with Pac-Man." I analyzed some of the algorithms present in the documents and attempted to get a base understanding of the types of algorithms Dr. Lasisi and I would be using in the semester. While I did not completely understand them at first, it gave me an idea of what type of work Dr. Lasisi and I would be pursuing later in the semester.

Weeks 3-4

During weeks three and four, the workload increased to stay on track with the syllabus as Dr. Lasisi and I began thinking on an idea for the application interface. Once we developed an idea for how we wanted the environment and agent to look, I first began working on developing the agent.

The agent development introduced me to new classes within JavaFX that allows me to connect lines together using a "Path." I connected curved lines with straight lines and fill the space with color to give a shape of the truck agent. Once I had an outline for the truck agent, I worked on making the truck look more realistic by adding some shading of colors. Then, I added more groups of lines and circles to make doors, windows, and tires. After I finished the design of the agent, I moved forward to developing the environment.

The environment proved to be a work in progress for several weeks. However, the underlying set up of the environment class stayed consistent throughout the semester. I used a JavaFX GridPane holding blank StackPanes to give us the design that we were after. After I had accomplished this, I worked at finding the best way to display the agent within the environment. Once I developed a way to display the agent in the environment, I worked on displaying goal cells and unpassable cells in the environment. This gave us a rough draft of the engine for the application.

During this time, I developed a simple webpage using HTML and CSS that could act as a place to learn more about the project. The webpage included several different pages ranging from the algorithms used, basic information about the project, the full research paper, and the team working on the project. This part of the project was interesting and helped me develop my multi-page HTML skills that I will use later in my CIS-331 class.

Weeks 5-6

Once a rough draft of the game engine was produced, it was time to work on efficiency and adding some functionality the final product would require. These weeks were spent finding ways to shorten classes, finding more efficient ways to accomplish goals, and testing.

One big accomplishment I made during this time was figuring out how to check if the agent had made it to the goal in constant time. Previously, the method I made was quadratic. Obviously, this was not efficient enough and needed to be improved. To accomplish this, we implemented new variables and a new Location class that would give every cell within the environment an x and y coordinate location. The agent could then use this new information to check if it was at a goal state in constant time rather than quadratic.

Another skill used was implementing an interface to hold constants and the build method that would return a completed agent. This interface proved to be very helpful in the organization and readability of the code produced. It allowed for constant names to be applied to multiple classes and show what the different values meant.

Another change made was combining methods into single methods to cut down on the test class length. This showed me how different methods did not need to be separate every time a certain method was called, another method was called directly after. Simply, you could combine these methods into one to cut down on the length of classes used to test the application.

Weeks 7-8

During weeks seven and eight, I began to read *Artificial Intelligence: A Modern Approach Third Edition*. This textbook has proved to be very knowledgeable and has taught me most of what I know thus far about AI. It has taught me the basics of what I will need moving forward with Dr. Lasisi in this independent research project.

The beginning of the textbook discussed different approaches to AI and how each approach has been used in past research. Four categories of AI were discussed: thinking humanly, acting humanly, thinking rationally, and acting rationally (Russel and Norvig, 2010). Each approach has a test that is used to set a standard for the AI program to determine if the program is a part of the specific category. For instance, acting humanly uses the Turing Test approach. During this test, a grader must not be able to tell the difference between the computer and real people when asked questions different fields of study. Meaning, the machine would have to be capable of natural language processing, knowledge representation, automated reasoning, machine learning, computer vision, and robotic capabilities. This chapter of the textbook also taught me that perfect rationality is not possible when dealing with complex environments. This is due to the computational demands perfect rationality would require.

Next, I learned about key terms relating to the Agent, which is anything that is capable of perceiving the environment and acting on the environment using actuators (Russel and Norvig, 2010). Actuators are what cause an action to take action or behave within the environment. For example, if an agent had legs, those legs could allow the agent to move. Sensors are what give the agent the ability to perceive inputs from the environment. What is perceived from the environment at a given moment in time is called a percept. This percept reflexes the current state of the environment. Agents are able to store the history of the states it has been in, this data is called a percept sequence. I also learned about what makes a rational agent, rational. A rational agent is one that does the right thing, where the right thing is determined by the consequences of the actions the agent could take.

Information gathering is another subject that was discussed in the reading. There are two main types of information gathering, doing actions to modify future percepts and exploration of the environment. For this project, information gathering must be done by exploration since the environment state is unknown to the agent. In terms of observability, our agent would be unobservable since the agent is unaware of where the goal is within the environment. While our agent is unable to observe the environment, it is known because it knows what its actions outcomes are. If the agent was unknown, the actions of the agent would have unknown outcomes.

Formulating problems is another aspect of AI that is essential to developing a successful agent. The problem must include five parts of information: initial state, possible actions available, description of actions, goal test, and path cost (Russel and Norvig, 2010). In order to create efficient agents, the problems must not include all the small details of the environment. Instead, the problem should have a level of abstraction, looking at the important details and ignoring the unimportant details, that allows the agent to only have the information it needs to complete the problem.

Once a problem is developed, searching algorithms should be used to determine the correct sequence history for an efficient agent. To accomplish this, search trees should be used with the initial state of the agent as the root node. Each node in the tree is a possible new state if a given action is taken. Therefore, each node is described as having four key parts: state, parent, action, and path-cost. Any leaf node in the tree is described as being a frontier, which is a possible expansion that can be taken by the agent.

I learned there are many ways to approach the searching algorithms in my reading. However, it is important to understand how an algorithms performance is measured. There are four main points that are used to determine the performance of an algorithm: completeness, optimality, time complexity, and space complexity (Russel and Norvig, 2010). Completeness is measured as “yes” or “no,” determined by if the algorithm is guaranteed to generate a possible solution to the problem. Optimality is based on if the algorithm finds the optimal solution. Time complexity is the number of nodes the algorithm has to search through, while space complexity is determined by the number of nodes stored in memory. Complexity, which is involved in both time and space, is based on the branching factor, depth of the shallowest goal, and the maximum length of a path.

The first searching algorithm I read about was the breadth-first search. This search involves expanding every node of the tree at the same depth before searching through the next level. For example, the root would be expanded, then the root’s children nodes would be expanded. From there, the children’s children nodes would be expanded but only once all of the children of the root have been expanded. This type of search is complete, given there is a goal node within a given depth (finite) and the branching factor is not infinite. The problem with the breadth-first search is the time and space complexity of the algorithm. The algorithm runs at a time complexity of O(b^d) where b is the branching factor and d is the depth of the tree.

Similar to the breadth-first search, the uniform-cost search expands the nodes with the lowest path cost. This is helpful when not all step-costs are equal. This type of search is useful when the step-cost is not equal of all nodes because it will search the lowest cost first, possibly limiting the search time and space complexity. If there is a goal node, it will have the lowest step-cost of any goal node as a result of the algorithm.

Another search algorithm I read about was the depth-first search. In this approach the deepest/highest depth node will be expanded completely before the next node will be expanded. When illustrated, it will appear as if the algorithm is searching the left-side branches of the tree completely before moving on to the right-side branches. This type of search will have a similiter run time to the breadth-first search with O(b^m) where b is the branching factor and m is the maximum depth of a node. Given that these values are finite, the search will be a complete search.

Very similar to the depth-first search, the depth-limited search involves expanding the nodes in increasing depth first, however there is a limit to the depth of a node that will be expanded. Therefore, if there is a limit set as depth of 5, and a node at the depth of 5 expands to have children nodes at a depth of 6, they will not be expanded. This leads the depth-limited search to be an incomplete search algorithm, since it does not view all possible nodes when there are nodes with a depth > maximum depth searched.

Iterative deepening depth-first search is a search algorithm that will expand each node at a given depth until the goal node is found. The part of the algorithm that makes this iterative is the limit of depth searched is incremented by one every time an iteration runs without finding the goal node. This search algorithm is complete given that the depth of the tree is finite since the depth limit will continue to be incremented until the goal node is found, at which point the algorithm will be finished running.

The final search discussed in my reading was the bidirectional search. The algorithm in this search will run one search algorithm from the initial state while simultaneously running an algorithm from the goal node. The goal of this algorithm is that the two search algorithms will meet in the middle to find the correct path. This will cut the time and space complexity of this algorithm will be O(b/2) where the b is the branching factor.

Weeks 9-10

During weeks nine and ten I mostly worked on revising my SURI Proposal so that it was the best it could be before it was due. However, I also worked on the early parts of implementing the Node and Successor Function classes. The Node class is implemented nearly identical to how DeNero and Klein discussed in their textbook. However, it is altered to fit the specific Agent we created. Within the Node class it will have a Cell as the state, a parent Node, an action the parent took, the path cost, heuristic estimate value, and a f value.

The Node was the start to implementing the searching algorithms. However, the working class will be the Successor Function which will be expanding Nodes by using the actions possible by the truck. The Successor Function class has the expand method, the successorFunction method, and the actions a truck can take. When first implementing this I was mostly focused on how to store the actions, I came the finalization that storing the action as a String was the most efficient way since there are only three possible actions for a given agent.

Weeks 11-12

After the initial implementation of the Node and Successor Function it was time to begin implementation of the Breadth First Search as discussed in weeks seven to eight. I had a template to use for implementing the Breadth First Search given to me by Dr. Lasisi, however, because of the project's complexity with Objects a few of the methods had to be changed and/or overridden. First was the equals method for the Cell, Node, and Agent classes. Because these are all objects and different instances of them are being created the equals method needed to test all of the attributes of one compared to another to see if they are equal. This would then help in fixing the duplication errors I was having. In the method of checking for duplicates it was giving me an error where it was not detecting duplicates, however, once these methods were overridden it solved the problem.

Once the duplication error was solved the code began to run, however, the agent was being moved by the Successor and expand functions. This was causing a problem because those functions should not move the truck. Rather they should just find the goal and return that goal Node. Therefore, I had to move the action methods (goForward, turnRight, and turnLeft) into the Successor Function class and alter them so that they do not directly move the agent, but rather adjust the location of the agent in that node.

Weeks 13-16

During the last few weeks of this semester I spent some time fixing known outstanding bugs within the code. One of these was the code running infinitely. This was fixed by altering the method in the Breadth First Search when checking nodes in the Node Database which stores all previously used Nodes. Next, I worked on creating a two-dimensional array that stores percepts of the cells to the north, west, east, and south of a given cell. This could then be used to check what the percept was of a cell when a truck was facing a given direction. Once this was working it now allows me to utilize it within the Successor Function class when testing the go Forward method.

During this time, I also got the Breadth First Search to recognize goal nodes when the goal node is on the same row as the agent is on originally. However, a major issue arose during these weeks. After a lot of testing I found that the agent was moving forward, however, the direction was not changing due to how Objects are passed in Java. Because the same agent is being altered whenever the agent is turned right, and then left, the direction is not actually changing. In other words, the direction is being cancelled out by the direction changes.

This issue is why the agent has only been able to find goal nodes on the same row as the agent is originally. There is only side-to-side movement and not vertical (up-and-down) direction. During these weeks I spent many hour doing research on Object manipulation in Java and possible ways to get around the Object passing issue. Because the agent must remain the same agent throughout the successor function, I can not create new agent representing the same attributes as the original agent. This would most likely be the easiest solution to the problem, however, it would break the requirements of the project.

Another possible solution that I thought of was adding a turning action to every second and third Node being tested in cycles of three. However, this was not successful in working. I think this is because the agent had already moved forward in the environment and thereby not exploring Cells in the vertical direction except for after the agent had already moved.

I also tried updating the equals methods in the Cell and Node classes to not testing the agent within the cell since that is also an Object and would require an equals method. However, after testing I found that the equals method in all of the three classes were working without error.

Through the research and not finding a solution to the object issue with the agent, I now figure the solution that will be most likely to give a desirable outcome will be adding an extra data structure that will store the actions done by all previous parent nodes. Then, when that node is called upon to test if it is a goal node, it will call all the parent's actions stored in this list. This will keep the most up to date direction and location that the node should be at. However, actually implementing this has been very difficult. I believe it is mostly due to the various amounts of testing I have done on the code changing lines. During week fourteen especially I tried to clean up the code and get it back into running shape without unnecessary clutter. This will allow me to see the clean code and maybe help me with implementing the list system I just discussed.

Moving Forward

My plan for moving forward is to continue working on the implementation of this list idea as I believe that is the best chance at getting the search to work. Ideally, I will be able to continue working on this through the finals week since I will have more time with most of the work for other classes complete and have everything figured out before moving into the summer. This will allow Nick Hartnett and I to work on more implementation with a guideline to go off of.

I also plan on updating the website draft I made earlier in the semester to JavaScript for added functionality and efficiency. This should allow for the website to be faster, more optimal, and be more updatable as more things are added to it.

What I Learned

This semester has helped me as a programmer and computer science student not only because of the multiple fields I have learned about (database programming, java servlets, and AI) but because of it forcing me to do research. I have done more research for this project than any other class in my college experience. Specifically, in the last few weeks, I have not made as much progress on the code as I would have liked, however, I have done the majority of the research necessary to hopefully be able to complete the project. Another thing I have learned is that a lot of projects require more research than initially thought or expected. When I was reading about some people's struggles with various projects, they explained that for every hour of actual code writing they did they did double or triple that amount in research time. I found that very relatable in the latter part of the semester. Another big thing I think I took for granted was having someone to work directly with throughout the duration of the project. That is one reason I am excited to continue working on this project through the summer with Nick as I believe we will be much more productive having twice the amount of eyes and thoughts as we come closer to the end of this project.

Another huge lesson I learned was that testing and bug fixing is probably the biggest part of programming. Had I not learned how to properly test different methods I would not have made it this far and would still be stuck with a simple graphical interface. Also, there have been many times a bug or bugs have derailed my plans and forced me to spend much time on resolving a small issue.

References

DeNero, J. and Klein, D., 2010, July. Teaching introductory artificial intelligence with pac-man. In *First AAAI Symposium on Educational Advances in Artificial Intelligence*.

Lasisi, R. and Dupont, R., 2019. ALP4AI: Agent-based Platform for Introductory Artificial Intelligence.

Liang, Y. Daniel. *Introduction to Java Programming and Data Structures: Comprehensive Version*. 11th ed., Pearson, 2019.

M. Ryczkowska, M. Nowicki and P. Bała, "Level-Synchronous BFS Algorithm Implemented in Java Using PCJ Library," *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 596-601, doi: 10.1109/CSCI.2016.0118.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach Third Edition*. Prentice Hall.