ALP4AI: Agent-based Learning Platform for Introductory Artificial Intelligence

Ramoni O. Lasisi, Connor Philips and Nicholas Hartnett
Department of Computer and Information Sciences, Virginia Military Institute, U.S.A.

Keywords: Artificial intelligence, Agent, Search, Algorithm, Visualization, Experiments.

Abstract: We develop *ALP4AI*, an Agent-based Learning Platform for Introductory Artificial Intelligence. *ALP4AI* is a graphical-based tool that is suitable for teaching introductory AI, places emphasis on hands-on learning, and provides for visualization of results. The tool we have developed is suitable for solving problems in the state space search problem domain. It provides for different environments modeling, including, environments that contain obstacles or are obstacle-free, single or multi-agent, and contains single or multi goals. Students can also conduct and report results of experiments using *ALP4AI*. This project is expected to provide a new frontier of a simple, yet theoretically grounded hands-on learning tool with visualization to aid in AI education and provision of vast resources that benefit the academic community.

1 INTRODUCTION

There has been much emphasis on hands-on and experiential learning in undergraduate computer science education including computer programming (von Hausswolff, 2017) and artificial intelligence (Parsons and Skiar, 2004), (Bryce, 2011), (Perhinschi, 2017). We designed and implemented a learning tool in artificial intelligence (AI) tagged, *ALP4AI: Agent-based Learning Platform for Introductory Artificial Intelligence.* The idea to develop this tool was conceived by one of the authors after teaching several sections of the introductory AI course to undergraduates at a four-year college in the United States. Students in the course typically have completed two semesters of introductory programming and have exposure to a data structures course. Topics for the course were selected from the widely used AI textbook: *Artificial Intelligence - A Modern Approach, Third Edition* (Russell and Norvig, 2010).

Our goals in developing this tool on one hand is to enhance experiential learning for students, and on the other, aid in course delivery for instructors in teaching the introductory AI course. While our tool would be applicable to a wide range of domains and topics in AI education, as a case study we demonstrate its application to *problem solving as search* in this paper. By design, the developed tool is reconfigurable, extensible, and adaptable; thus, providing opportunities for faculty and students to research, design, build, and test pluggable modules for intelligent agents' strategies, single and multi-agent search, and conduct appropriate experimentations and evaluations.

To develop the tool, we design and implement an agent class. Agents in our work are modeled to provide internal representation of the world state-space to find paths to goals. We also implement a GUI-based two-dimensional representation of a problem solving environment that allows for visualization of agents' movements, interactions, and results. The tool showcases a grid-styled environment consisting of cells that can hold an agent, a goal, nothing, or an obstacle. The cells that hold nothing are free spaces for an agent to move through as it searches for goals. All cells are passable except the cells with obstacles. *ALP4AI* is built to aid in teaching, thus allowing students to add new features and functionalities to the environment. The adaptability of the tool allows for it to be used such that the teacher controls what the students have access to and what they need to make for themselves.

The following is our working hypothesis:

Students are likely to have intuitive recognition and be enthusiastic to use and implement artificial intelligence algorithms presented using a graphical-based tool that provides for visualization of results, and places emphasis on hands-on learning.

Lasisi, R., Philips, C. and Hartnett, N.
ALP4AI: Agent-based Learning Platform for Introductory Artificial Intelligence.
DOI: 10.5220/0010916100003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 3*, pages 842-850
ISBN: 978-989-758-547-0; ISSN: 2184-433X

Many traditional methods of teaching computer science topics including AI algorithms in most part provide description of the methods and use some data to illustrate the functionalities of these ideas. These descriptions may in some cases be followed with limited visualization of the steps involved before students are asked to provide implementations of the ideas. Examples using this approach abound in many computer science books such as those used in introductory programming courses, data structures, algorithms, and AI. One concern about this approach is that some of these important topics in the field of AI are sophisticated and are difficult concepts for lower level undergraduate students to easily and clearly comprehend. Further to this, and in accordance with (Julian Estevez and Grana, 2019), "*There is a wide consensus among computer scientists that it is quite difficult to teach the basics of AI.*" Another issue of concern is how much of the knowledge gained from using this approach is retained, and can be applied when students are faced with new problems from different domains than they have been exposed?

There have been several attempts in the literature to address this concern using hands-on learning approaches to teaching AI algorithms. (Parsons and Skiar, 2004) use LEGO Mindstorms in teaching AI. The approach utilized in this work is more towards students being able to program the robots and as well test out some of the functionalities while engaging in contests among the project groups. Thus, their approach is not towards formulating problems or implementing AI solutions or algorithms. In a different hands-on method, (Bryce, 2011) uses a project-based approach in the game of Wumpus World (WW) to teach introductory AI concepts. Description of the WW environment can be found in (Russell and Norvig, 2010). Students are required to implement search, satisfiability, and declarative planning descriptions algorithms applied to the WW environment of different sizes. As interesting as the WW project is, it only provides for a single agent in the environment. Having a multi-agent environment with multiple goal states would not only be interesting, but will also provide further challenges to students. It will also open up new possibilities to seeing how different algorithms work in such more complex and diverse environments.

Another work on introductory AI by (McGovern et al., 2011) uses Java-based games. Although the games here are graphical and also have elements of multi-agent environments, two of the three projects will have students implement several variants of A^* algorithms. Finally, we looked at a related course that uses the Pac-Man game to illustrate the introduction of AI algorithms (DeNero and Klein, 2010). In this course, students are required to implement various algorithms to solve problems in the Pac-Man domain.

In contrast to the works above, our emphasis in this work is to build a tool that provides not only hands-on and experiential learning to implement basic AI algorithms as many of the works cited did, but also to teach students the process of problem formulation and development of solutions that students can be able to apply to new problem domains in future courses or careers. We have developed *ALP4AI* to be simple with little learning curve. Thus, students *do not* need several hours of study or class periods to understand the details of and how to use the simulator. The tool makes provision for several functionalities that students can use to model different AI problems and develop their solutions.

2 DESIGN METHODOLOGY

2.1 The Problem Environment

We provide a description of the problem environment that our agent-based learning platform is based. The framework is situated in a two-dimensional grid that represents the environment that agents are required to explore. Agents are given the task of locating goals that are randomly placed in the environment. Goals represent desirable states that agents need to achieve. Many interesting problems in introductory AI, including the *state space search* used to illustrate the functionalities of *ALP4AI*, can be modeled using this environment.

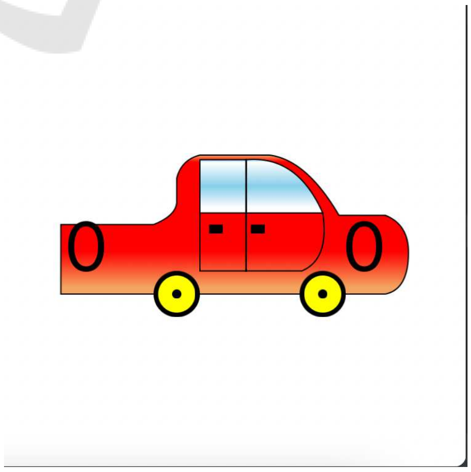

Figure 1: The TruckAgent.

We use JavaFX technology to build a GUI platform that aids students in visualizing agents' movement, interactions, and results in the environment. We first design and implement our agent tagged *TruckAgent*. See Figure 1. This is followed by creating GUI for a two-dimensional grid that represents the agents' environment. Some cells in the grid contain goals denoted by gold coins. These cells represent goal states that agents are attempting to reach. The impassable cells (i.e., obstacles) in the environment are denoted by black squares. See Figure 2.

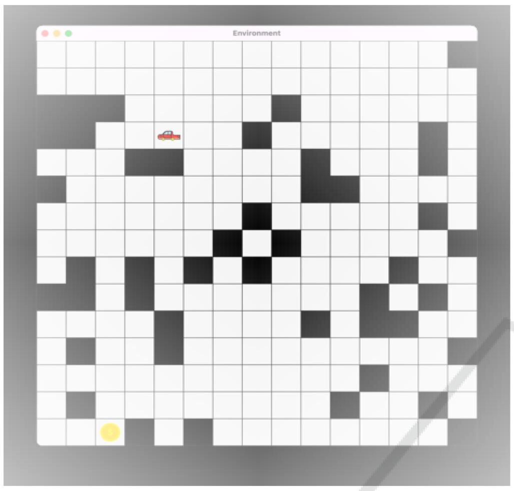

Figure 2: A 15×15 environment with one agent, one goal, and 45 obstacles.

ALP4AI allows for parameterization to generate varieties of problem environments. Different environments that can be defined among others, include *single* or *multi-agent* environments, environments with *single* or *multiple goals*, and with *obstacles* or no *obstacles*. Environments are further systematically organized into *states* so that agents can intelligently navigate them. The idea of using states to organize environments is fundamental to understanding basics of problems formulation in introductory AI. Furthermore, the state-space search problem that we consider in this work requires an understanding of the concept of states and *states' successors* to formulate problems.

2.2 Search Infrastructure

After developing the agent, environment, and the different types of cells it contains, we then focus on the search algorithms' infrastructure required by agents to navigate the environment. We define the abstraction of the world (i.e., the environment) that an agent is currently in as a state. For example, suppose a TruckAgent is at location (x_1, y_1) in a two-dimensional grid environment and facing the north direction, that description of the environment corresponds to a certain state say, s_1 for the agent. However, the agent is perceived to be in a different state, say s_2, if it remains in the same location but now facing the west direction. Thus, state s_1 is not the same as s_2.

The TruckAgent is armed with certain actions that it can perform in different states of the world. Possible actions include, *goForward*, *turnRight* and *turnLeft*. Agent's actions are only applicable in a state if the preconditions for the actions are satisfied. For example, it is not possible to go forward when the agent is facing an impassable cell, thus the action *goForward* is not applicable in this state. Each action has a step cost, and the step costs of a state and its parent add up to a path cost up to that state.

Note that the concept of a state is an *abstract* description of the world that the TruckAgent is currently in. To provide a *concrete* description of a state, we implement a *node data structure* that describes information contained in a state. Some data members in the node data structure are:

- parentNode - a node representing the parent's state of the current node
- parentAction - the action taken to reach the current state from the parent state
- pathCost - the total number of steps taken by the agent from the root node (where the agent started) to the goal node
- action list - a list of actions taken from the root node to the goal state (if found) or the last place the agent searched (if no goal state exist)

It is noteworthy to know the distinction between a state and its corresponding node.

Students will first be required to implement a *transition model or successor function* for the state-space search framework. Let s be a state that a TruckAgent is currently in; we define the *successor states* as the possible states that the agent can transits from s. Given a particular state of a TruckAgent and the set of possible actions by the agent, the successor function computes and returns the successor states for the current state. The node that corresponds to the given state is thus said to have been *expanded*. The successor nodes also correspond to different valid successor states that the agent can transit to. One essential skill that is expected to be gained here is the understanding of the *concept of a node in a tree and how it relates to the abstraction of a state in an environment.*

2.3 Problem Formulation

The following is the formal problem formulation that we used to model the state-space search problem that we consider in this work:

- **States:** A state is a combination of a cell (identified by its location) in the two-dimensional grid environment and the direction (i.e., north, east, west, or south) that a TruckAgent is facing. In this type of setting, each cell has four states.
- **Initial State:** Agents are randomly placed in a state in the environment when starting a new world. So any state in the environment may be designated as the initial state. However, in order to allow for repeatable experiments that can compare performance of search strategies, we also make provisions for placing agents and goals in desired starting states of the environments.
- **Actions:** The TruckAgent is equipped with the following three actions: *goForward*, *turnLeft*, and *turnRight*. Each of these actions places the TruckAgent in the successor state if the action is applicable in the current state. Note that both of the actions *turnLeft* and *turnRight* are applicable in all states. Although the location of the agent remains the same for both actions, the direction of the agent is modified to that of the new state. On the other hand, the action *goForward* places the agent in a different state with a different location when the action is applicable in the current state.
- **Transition Model:** Appropriate transition models (also referred to as successor functions) that agents can use to select successor states are developed. Given a particular state of a TruckAgent and the set of possible actions for the agent, the successor function computes and returns the successor states for the current state.
- **Goal Test:** When an agent is initially placed in a state or navigates to a new state based on the result of the successor function, the agent checks to see if the state it is currently in is a goal state.
- **Path Cost:** The total number of steps by an agent in a path defines the path cost. Each step costs 1.

3 SEARCH ALGORITHMS

The *ALP4AI* tool allows for the implementation of search algorithms using uninformed and informed search strategies where a TruckAgent navigates the environment to reach a goal state (i.e., finding the gold coin). It also supports implementation for multi-agent and multi-goal environments. Some uninformed search algorithms supported by *ALP4AI* include *Breadth First Search (BFS), Depth First Search (DFS), Iterative Deepening Depth First Search (IDDFS), and Uniform Cost Search (UCS)*. Additional support to implement informed search strategies including *Greedy Best First Search (GBFS) and A* Search algorithms* is provided.

3.1 Uninformed Search Algorithms

The first of the five uninformed search algorithms implemented in *ALP4AI* is the BFS. A queue implementation of the BFS algorithm is achieved. The first node that is inserted into the queue is the first to be removed for expansion. The successors of this node are then added to the back of the queue since they are at a different depth than their parent. Thus, the BFS algorithm searches all nodes at a particular depth before increasing the depth. For instance, the BFS algorithm will search all nodes at a certain depth, say 3 and expand on those nodes before searching at depth of 4 and their expanded nodes.

Figure 3 is an illustration of two agents utilizing the BFS algorithm to find goals. The path taken by each agent is shown with arrow-heads.

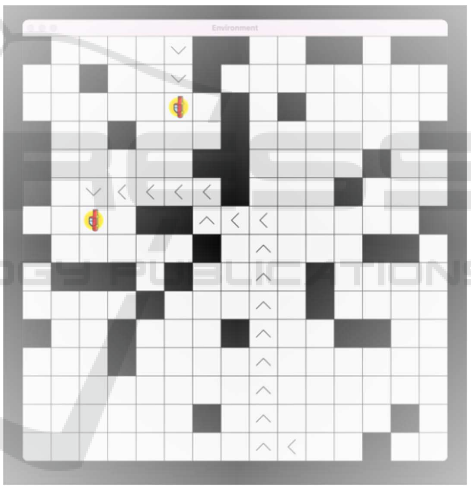

Figure 3: Two TruckAgents utilize a BFS algorithm on a 15×15 grid size with two goals and 45 obstacles.

The next search algorithm implemented is the DFS algorithm which uses a stack data structure to search and expand nodes. The last node that is pushed into the stack is the first to be removed for expansion. The successors of this node are then added to the top of the stack since we intend to explore a particular path until reaching a leaf node. This implementation searches a branch of the nodes before searching through the next branch. Because this method will continue to search until a goal is found or not, the number of actions required to reach the goal is relatively high. Figure 4 illustrates an agent utilizing a DFS to search the two-dimensional grid environment. As seen from the figure by the directions of the arrow-heads, the agent took more steps than necessary to reach the goal state.

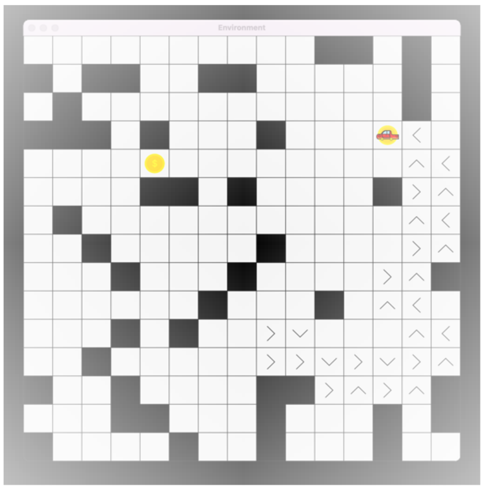
Figure 4: One agent utilizes a DFS algorithm on a 15 × 15 grid size with two goals and 45 obstacles.

The DLS also utilizes a stack data structure but implements a limit to the number of actions an agent can perform. Once the tree of nodes has been traversed down a branch to a given depth limit, the search will then start with the next branch. Because of this limit, it is possible with a DLS that no goal will be found, however may usually find goals with less actions than a DFS. Figure 5 illustrates a DLS algorithm in action.

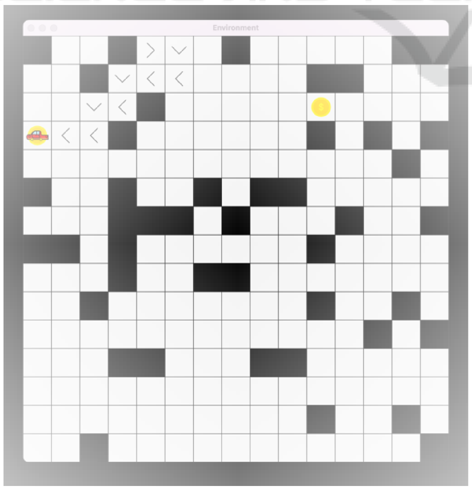
Figure 5: One agent utilizes a DLS algorithm on a 15 × 15 grid size with two goals and 45 obstacles with a limit of 25 steps.

The IDDFS is implemented next. A stack implementation of the DLS algorithm is first achieved. The DLS performs depth first search at a particular depth specified by the IDDFS to determine if the goal state exists at that depth. Otherwise, IDDFS increases the depth and repeat a new DLS search. For example, the search algorithm will search to a depth of 1, if no goal is found the search will restart and search to a depth of 2. If no goal is found still, then this process will continue until goal is found or not. The iterative pattern allows the agent to find the goal in less steps than all previous search that uses the stack data structure. Figure 6 illustrates the IDDFS algorithm in action.

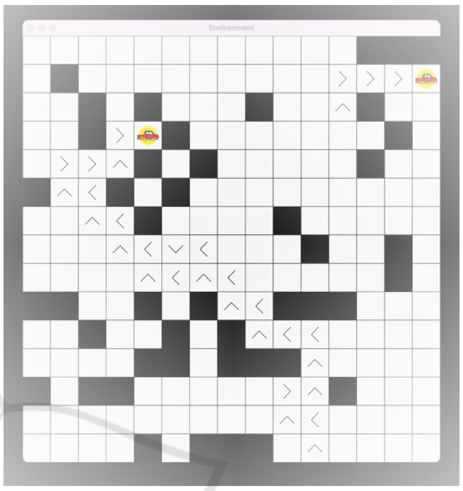
Figure 6: Two agents utilize IDDFS algorithm on a 15 × 15 grid size with two goals and 45 obstacles.

The last uninformed search algorithm implemented is the UCS which uses a priority queue data structure to search the environment. A priority queue which uses the path cost as comparator is maintained. The node having the least path cost thus far is removed next to be expanded in order to advance the search. If no goal is found, then the process will continue. This data structure will result in low path costs for the node due to the lowest path cost node being removed and expanded first. Figure 7 illustrates the UCS algorithm in action.

3.2 Informed Search Algorithms

We implement the following two informed search strategies: GBFS and A^* search algorithms while using heuristic functions, $h(n)$ and $g(n)$. The n in the function represents a node reference in the function. We first implement the well-known *straight line distance heuristic* described next.

Let (x_i, y_i) and (x_j, y_j) define the cartesian coordinates of the TruckAgent and goal cell respectively in the environment. The straight line heuristic is defined as the euclidean distance between the two points. Specifically, the value of the heuristic, denoted $h(n)$ is computed as follows

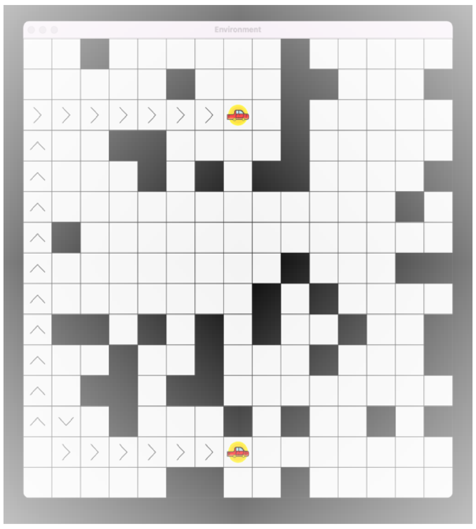

Figure 7: Two agents utilize a UCS algorithm on a 15×15 grid size with two goals and 45 obstacles.

$$h(n) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

The value $h(n)$ is the estimated distance of a node, n, relative to the goal node. Furthermore, we define another function $f(n)$ while using the A^* search algorithm as follows:

$$\begin{aligned} f(n) &= h(n) + g(n) \\ \text{where} & \\ g(n) &= \text{path cost thus far.} \end{aligned}$$

A priority queue implementation of the GBFS is achieved. A priority queue is maintained which uses the heuristic function $h(n)$ as comparator for the priority queue. The node having the least $h(n)$ value is usually removed next to be expanded with the inherent knowledge that the selected nodes lead to solution quickly. If no goal is found in adjacent nodes in the two-dimensional grid environment, then the process continues. This strategy typically results in low path costs. Figure 8 illustrates the GBFS algorithm in action.

The second informed search strategy is the A^* search algorithm. The implementation is similar to the GBFS except that we set the heuristic function as $f(n) = h(n) + g(n)$, where $g(n)$ is path cost from the start node to the current node n, and $h(n)$ is the value of the heuristic that is used to estimate the cost from the current node n to the goal state. This algorithm operates using a priority queue data structure to search the environment. The algorithm selects the next node as the one with the smallest $h(n) + g(n)$ value. The algorithm always estimate the distance to the goal cell while minimizing excessive cost. If no goal is found

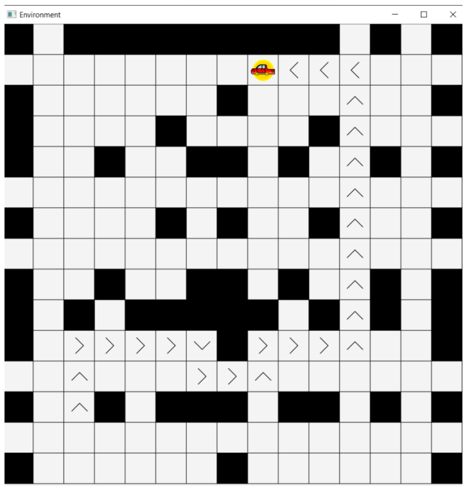

Figure 8: One agent utilizes a GBFS algorithm on a 15×15 grid size with one goal and 45 obstacles.

in adjacent nodes, then this process continues. The strategy always results in minimal path costs because it factors in the total path cost from the start node to the goal node in its heuristic function. Figure 9 illustrates the A^* search algorithm in action.

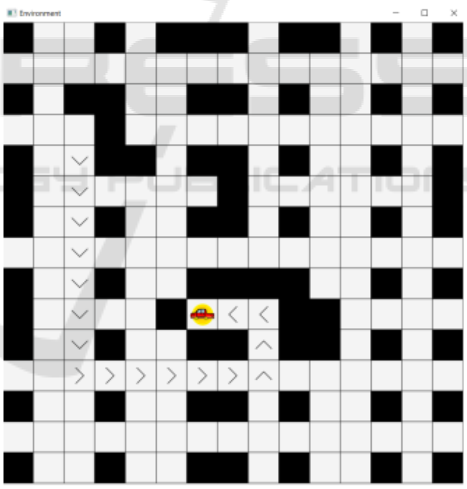

Figure 9: One agent searches utilizing A* search algorithm on a 15×15 grid size with one goal.

4 RESOURCES

We have explored the idea of problem solving as search in this work and provided detailed description of the process of the development and implementation of classical search algorithms that students encounter in an introductory AI course using the *ALP4AI* learning platform. Our approach is to provide students with only the *ALP4AI* game engine and the search infrastructure that supports the development and implementation of these search algorithms. Thus, students will be required to provide their own implementations of the search algorithms, conduct experiments using specified setups, and compare their results with the experimental results described in Section 5.

In addition to making the game engine and the search infrastructure available to the students, we provide and describe the following supplement resources that aid the students to seamlessly use the tool.

4.1 Seed Generation

The environments that the TruckAgent operates on and upon which the search algorithms are developed are designed to be randomly generated using unique seeds. Seeds are used to ensure consistency with how and where obstacles and goal states are placed in the environments, thus facilitating opportunities for repeatable experiments. The seed generation is designed to be scalable so the grid size can be changed with patterns of the obstacles remaining the same. These seeds allow for experimentation to remain consistent across different search algorithms.

ALP4AI currently has 15 seeds corresponding to 15 different patterns that are unique in the environments. Some seeds are solvable by the TruckAgent, i.e., the agent can reach the goal, while some are not. These environments that are not solvable provide additional performance measures for search algorithms to be evaluated when there are no goals found. Figures 10 and 11 provide two environment examples for seeds 7 and 10, respectively.

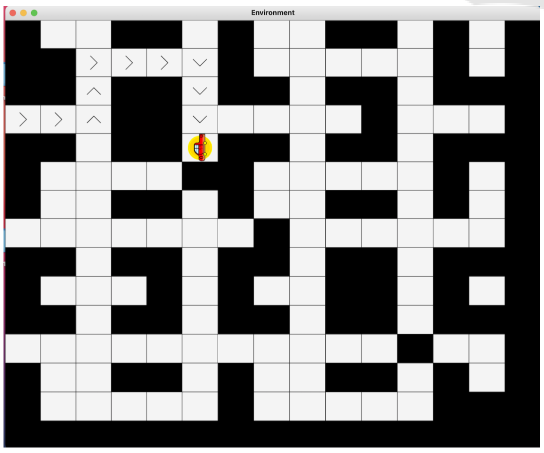

Figure 10: Two-dimensional 15 × 15 grid environment generated from seed 7.

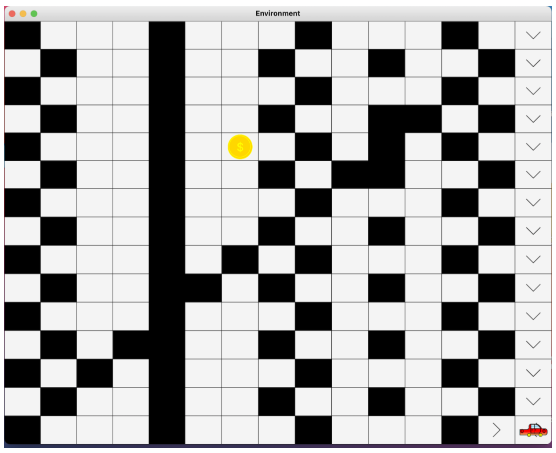

Figure 11: Two-dimensional 15 × 15 grid environment generated from seed 10.

4.2 Documentation

There are currently 24 Java and JavaFX classes in the project organized into five packages:

- ClassicalSearch
- Experiments
- GameEngine
- SeedGeneration
- Infrastructure

To seamlessly be able to bring students to speed in order to understand what each of the classes does and use it, we provide detail documentation for these classes. The documentation comes in the form of a UML diagram for a class followed by a brief description of the class, and explanations of the methods (functions) in the class. See the Appendix for sample documentation of the TruckAgent and Location classes implemented in the *ALP4AI* tool.

5 EXPERIMENTS AND RESULTS

We conduct experiments on the search algorithms in the *ALP4AI*'s two-dimensional grid environment. This section compares the performance of the search algorithms (uninformed and informed search strategies) considering different grid dimensions and seeds. The comparisons are performed under the following headings: number of nodes expanded, time to complete search in milliseconds, total time to complete search in milliseconds, and number of actions taken by agents. The number of nodes expanded is useful in understanding the amount of the tree that was searched before a goal is found. The time for

Table 1: Data set of a single agent, time were recorded in milliseconds. Data collection was obtained using an intel i5 8th generation 4 core processor.

Search Type	Expanded Nodes	Search Time	Actions Taken	Total Time
BFS	321.4	43.6	15.2	7664.8
DFS	313.7	147.2	134.9	47346.4
DLS	332.9	136	91.6	45990.5
IDDFS	2386.2	125.4	19.4	9851.3
UCS	230.5	37.7	11.7	5909.6
GBFS	23.4	18.9	16	8041.3
A^*	43.6	20.9	10.4	5239.7

search is important in understanding the difference in efficiency of the searches in terms of time taken. The total time takes into account the actions taken by the agent and difference in efficiency. The number of actions taken by an agent differs greatly among the searches.

Sets of data were collected from fixed environments. The data was compiled and then evaluated to find the average for each of the four main data sets: number of nodes expanded, time to complete the search in milliseconds, total time to complete in milliseconds, and number of actions taken by agent. Table 1 shows the results of the experiment.

When comparing the uninformed search strategies the data shows UCS to be the most efficient in all categories of recorded averages. The closest search strategies among the uninformed searches is the BFS. The DFS and DLS were the least efficient in terms of time to complete. Furthermore, the IDDFS is the least efficient by a large margin in expanded nodes.

Between the two informed searches involved in the experiment, the A^* search is the most efficient in actions taken and total time used. The GBFS however had a lower average search time and lower number of expanded nodes. Overall, when comparing both the informed and uninformed searches the GBFS and A^* hold the most efficient averages of all searches involved in the experiment. The visualizations for the data are depicted in the four bar graphs in Figures 12 – 15.

6 CONCLUSIONS AND FUTURE WORKS

We develop ALP4AI, an Agent-based Learning Platform for Introductory Artificial Intelligence. The tool we have developed is suitable for solving problems in the state space search problem domains. ALP4AI allows for parameterization of the environments. Different parameters that can be defined include provisions

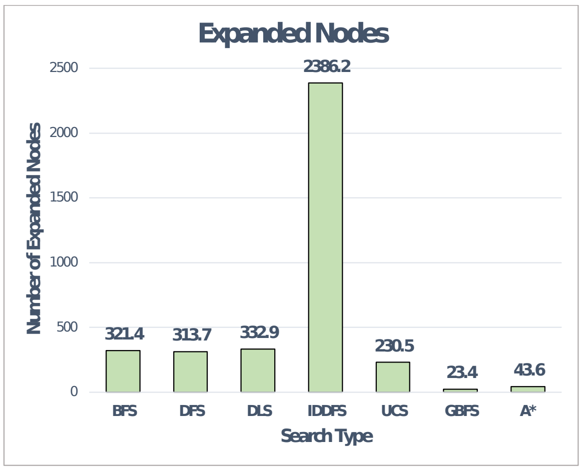

Figure 12: Number of Expanded Nodes.

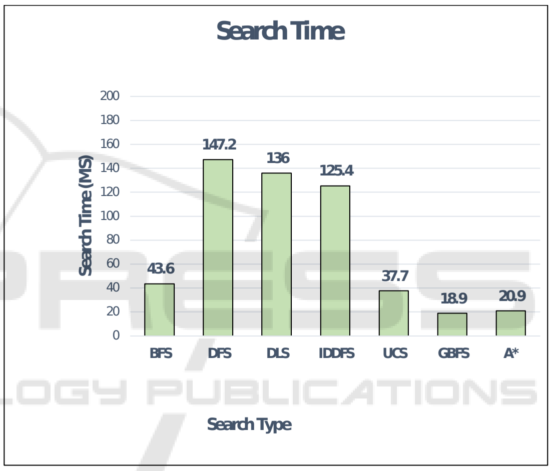

Figure 13: The Search Time.

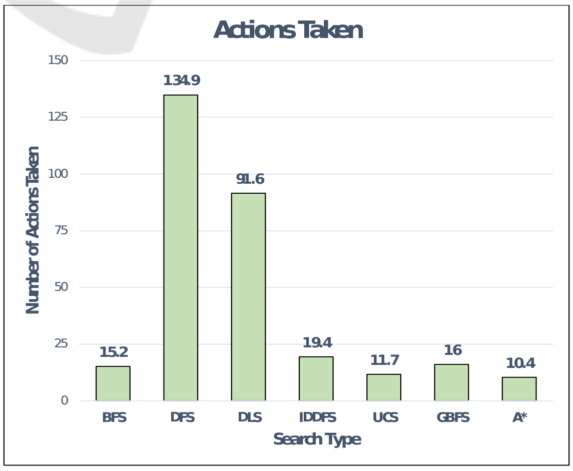

Figure 14: Number of Actions Taken.

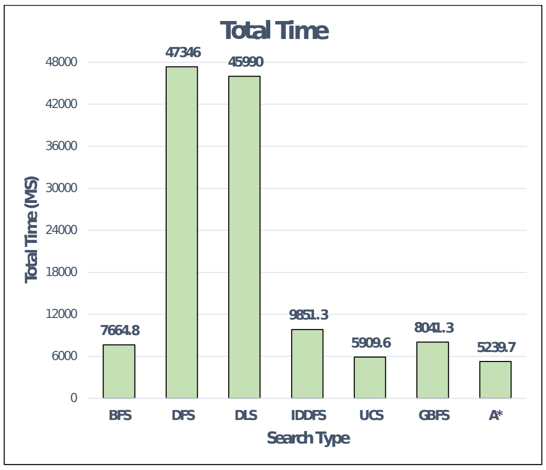

Figure 15: Total Time Taken.

for single or multi-agent environments, presence of single or multi-goal, and obstacle-free or obstacles-present environments.

The outcomes of this research are expected to have a high positive impact in enhancing experiential learning in computer science education as a whole, and AI course delivery in particular. This project is expected to provide a new frontier of a simple, yet theoretically grounded hands-on learning tool with visualization to aid in AI education and provision of vast resources that benefit the academic community. This will be impactful for helping students see exactly how each algorithm developed by students to control the agents behaves. The outcomes are expected to aid faculty in the teaching of AI and related courses in schools.

Having evaluated the functionality of the ALP4AI tool for its technical functionality, future work will involve the evaluation of the relevancy of the tool for teaching introductory AI courses. Surveys will be conducted and data collected of the usage of the tool in teaching AI in the classrooms. Further future work will include the extension of the functionality of the tool to other problem domains in AI, such as reasoning using propositional and first order logics.

REFERENCES

Bryce, D. (2011). Wumpus world in introductory artificial intelligence. In *Consortium for Computing Sciences in Colleges*, pages 58–65.

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

Julian Estevez, Gorka Garate, L.-G. and Grana, M. (2019). Using scratch to teach undergraduate students' skills on artificial intelligence. In *arXiv:1904.00296. [Online]. Available: https://arxiv.org/abs/1904.00296*.

McGovern, A., Tidwell, Z., and Rushing, D. (2011). Teaching introductory artificial intelligence through java-based games. In *Second Symposium on Educational Advances in Artificial Intelligence*, pages 1729–1736.

Parsons, S. and Skiar, E. (2004). Teaching ai using lego mindstorms. In *Greenwald, L., Dodds, Z., Howard, A., Tejada, S., Weinberg, J. (eds.) Accessible Hands-on AI and Robotics Education*, pages 8 –13.

Perhinschi, M. G. (2017). Wumpus world in introductory artificial intelligence. In *An Introductory Course on Computational Artificial Intelligence Techniques for Engineering Students*, pages 1–9.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

von Hausswolff, K. (2017). Hands-on in computer programming education. In *ACM Conference on International Computing Education Research*.