Table of Contents

1. Introduction

1.1. Overview

We've decided to develop a search algorithm visualiser with a heavy focus on the 8 Puzzle as well as the Word Game and the Knight's Tour. The design for our program will implement a GUI that displays the 3 different puzzles as they are solved by various search algorithms. We chose this project in order to create a learning tool that can effectively demonstrate to students how different AI searching algorithms behave while solving the problem.

We plan to include vast amounts of customisation in the program, such as variable starting and goal states, a set of different algorithms to choose from and options for both a complete solve and a step by step walkthrough through use of the keyboard or mouse. The program will not only show the board as it is solved, but also the different routes it could take in the search tree and an indication as to why a particular algorithm may take a specific route in the tree. Each possible move will be shown and the actual move that will be taken highlighted.

The strengths, weaknesses and efficiency of the set of algorithms will be presented on screen, in a user-friendly manner for easy, fast and most importantly, effective learning. The puzzle's interface will be straightforward, and accessible to encourage people of all skill levels to use our learning aid. We have tried to make our program available for use by as many people as possible and as a result, our program will run on Linux, Mac OS and Windows. Either for teaching or learning this tool will prove a worthy utility for lecturers and students alike.

1.2. Business Context

N/A

1.3. Glossary

Pygame – A set of modules for python intended for use on the development of video games. It Includes libraries on computer graphics and sound programming.

Word Game - The word game is a game that consists of transforming one word into another word through by changing one letter at a time.

Knights Tour - The knights tour is the sequence of moves that a knight on a chessboard must make to successfully visit each square only once.

8 Puzzle - The 8 puzzle is a sliding puzzle that consists of a frame of numbered tiles and one empty tile. The goal is to get the configuration of tiles to the goal state.

GUI - Graphical user interface

Search tree - A search tree is a data structure where the values of the nodes on the left are always less than the parent node and the values on of the nodes on the right are
greater than the parent node.

2. General Description

2.1. Product / System Functions

The main functions of our application are laid out below. They allow for the addition of different puzzles and can be used universally.

- Generate random puzzle parameters
- Choose puzzle parameters
- Display the search tree
- Select different algorithms to solve the puzzle
- Solve the puzzle
- Reset the puzzle
- Go a step back in the puzzle
- Go a step forward in the puzzle
- Display puzzle statistics

2.2. User Characteristics and Objectives

The application will be available to be downloaded by anyone. It will be easily installed and will be accessed by a simple “exe” file just like any other application. The target for our application will be students who study computer science or anyone who has an interest in the area of algorithms and how they work. We also think lecturers could use it in their lectures as a demonstration tool. The application’s UI will be designed to be used by anyone so as not to exclude any individual trying to access the application.

The objective of the application is to aid people in understanding how algorithms work through use of our visualizer while solving puzzles.

2.3. Operational Scenarios

Main Menu - In the main menu the user will be able to navigate from puzzle to puzzle by clicking left and right arrows at the top of the screen to move to the next puzzle. Each puzzle will have its own set of options before creation.

The 8-Puzzle - When the 8-puzzle is selected on the main menu the user will be able to select whether they want to generate random start and goal boards or whether they want to create the boards themselves. The user can create a board by inputting numbers into the board provided. The puzzle is then initiated by clicking the start button.

Once the puzzle has been started the user can select which algorithm they would like to use. The user then has the option to solve the puzzle or to move step by step through the puzzle. The user can also reset the puzzle or step back through a solved or partially solved puzzle. The search tree will be displayed alongside the puzzle, through each move that is made by the user.

The Word Game - When the word game has been selected in the main menu, the user has the option of inputting a start word and a goal word or randomly generating them. The puzzle is then initiated by clicking the start button.

Upon starting the puzzle the start and goal words are displayed. The user can then select the desired algorithm to use on the word game puzzle. The user can then step through the algorithm to get to the goal word or solve the puzzle to display the entire list of words from the start word to the goal word. The user can also reset the puzzle or step back.

Knight's Tour - For the Knight's Tour puzzle, a standard 8x8 chessboard is given and the user picks a starting point for the knight. The user then needs to click start to begin the puzzle.

The user can select which algorithm they want to use. The puzzle can then be solved completely or stepped through. The user also has the option to take a step back or reset the puzzle to start again. The moves made are displayed on the board as numbers.

2.4. Constraints

Size of the Screen - Due to the size of the screen we were only able to show the current node of the tree and the branches one layer down. Ideally we would've liked to show the full tree but that was not feasible with the size of the screen.

Time - We would like to make our Visualizer as helpful as possible but we face the dilemma of having a completed application by a certain time and therefore may not be able to make the application as visually helpful as we would have liked.

3. Functional Requirements

3.1. Solve the Puzzle

Description – This function will perform a complete solve of whatever puzzle has been selected, with the user defined parameters in mind. The puzzle will be visually solved in the GUI with an adjustable speed slider to either speed up the animation to get to the solution and statistics faster, or slow it down to get a more in-depth view of the moves being made. At any time the user may also pause and resume the process to stop the game in place and inspect the search tree or the stats.

Criticality – This function is essential to the project as the live-solve gathers statistics to display to the user. The animation of the solve also visually conveys the difference in speed between the set of available algorithms.

Technical issues – Implementing several algorithms to several puzzles in python, then animating these puzzles on screen using the Pygame library.

Dependencies with other requirements – The way in which the puzzle is solved depends on whichever algorithm the user picks.

3.2. Display the search tree

Description – There will be a visual search tree that develops as the algorithm runs on the puzzle. The search tree will show the current game state, each branch the game state could take, and the option the algorithm will take which will be highlighted for clarity. Displayed next to the search tree will be pseudocode that explains *why* each branch that is selected is selected.

Criticality – The main objective of this project is to present these AI search algorithms in a visually appealing, easy to grasp fashion as to make it easier for students to learn how they work by applying them to real problems. The search tree display is vital to this mission as it brings the bulk of the backend operations to the frontend in a presentable manner.

Technical issues – The entire tree will not fit on the screen most of the time so we must choose between displaying sections at a time and including a scroll bar option. Either of these options will present significant technical and graphical hurdles which we will tackle with the Pygame library.

Dependencies with other requirements – The search tree is built as a result of the puzzle being solved.

3.3. Customise/Generate random puzzle parameters

Description – The user has the choice at any time within the program to either randomly generate new starting/goal game state, or manually input their own.

Criticality – Not a critical function; however without it the system would always be operating on the same starting and goal game states, which wouldn't give a useful or accurate insight into how the various algorithms work.

Technical issues – We wish to implement parameter customising functions for each puzzle that are intuitive in nature and don't make the user think too hard about what they are doing. For example with the 8 puzzle we will tackle this by inserting a blank 8 puzzle where a user may click in individual tiles and input values.

Dependencies with other requirements – None.

3.4. Select different algorithms

Description – The working algorithm may be changed at any time in the program.

Criticality – The ability to cycle through different algorithms is important to show how each can perform when applied to the puzzles. The performance of each algorithm will show up in the statistics section to give users an idea of which algorithms are stronger. Also all the algorithms will take different paths through the search tree and we want to show this as well as why they take different paths.

Technical issues – Once the algorithms are implemented there should be no significant issues switching gears to a different algorithm.

Dependencies with other requirements – None.

3.5. Display meaningful statistics

Description – We plan on recording the total moves made, the amount of backtracks, and the time taken to solve fully. More may be added later but right now these are the chief stats. These stats get updated after each move is made and are displayed next to the search tree.

Criticality – Statistics are important to show the strength of one algorithm compared to another, however it is not as critical as showing which path the algorithm takes through the search tree, as we intend on focusing more heavily on the reasons why an algorithm behaves the way it does as opposed to which is *better*.

Technical issues – No technical issues are apparent at the moment.

Dependencies with other requirements – This function updates as the puzzle is solved and runs concurrently to it.

3.6. Reset puzzle

Description – Resets the puzzles game state back to the defined starting game state. Also resets the stats and the search tree.

Criticality – Fairly important for the functionality of the program, the user must be able to reset the puzzle to attempt applying another algorithm or to apply new game parameters.

Technical issues – No technical issues are apparent at the moment.

Dependencies with other requirements – An algorithm must be selected and the game state parameters must be set in order to reset so the puzzle knows what state to reset to.

3.7. Go forwards/backwards through puzzle in single steps

Description – Instead of solving the puzzle in one go, a user may want to step through the puzzle one move at a time to get a more in depth look at the choices being made by the algorithm. This will stop the solve in place and pause the search tree as well.

Criticality – We view this as a critical feature as the user may not have time to read and understand the pseudocode or the search tree as it whizzes by them throughout the full solve.

Technical issues – Each move made will have to be remembered to be able to go back. In each loop of the main function there will be a check to see if the user wishes to advance or return to a previous game state.

Dependencies with other requirements – This function will work off the solve puzzle function.

4. System Architecture

4.1 System Architecture diagram

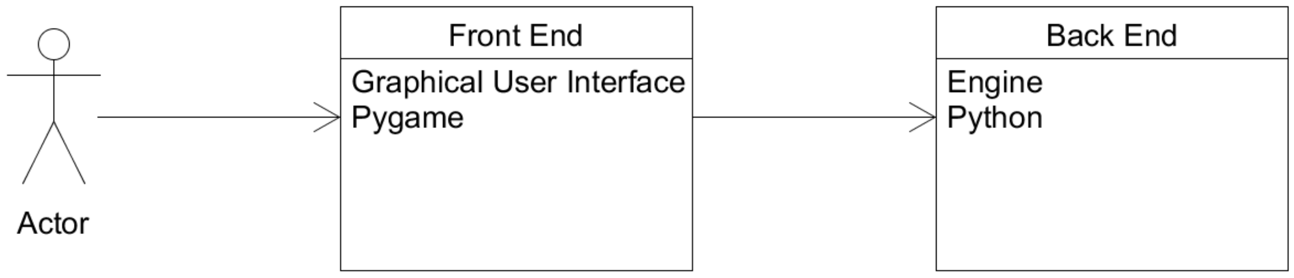


Fig 4.1 - The above diagram illustrates the architecture of the system and the relationship between the user and the system. The front-end that the user interacts with is a GUI built using the Pygame library in python. The back-end, or engine, is built using Python and will receive inputs from the front-end and send outputs to the graphical user interface.

GUI - The front-end is in charge of taking in the user inputs such as mouse clicks and button presses and is the mediator between the back end and the user. This is an imporant layer in the system architecture as the main focus of our project is visualisation (of the algorithms), and as such we must prioritise clear, quality graphics to ensure that our users can easily pick up the concepts being taught within the program. The inputs going through the front-end are tossed down to the engine where our logic and computation is performed.

Engine- Here in the engine, or back-end, the bulk of the computation is done. Inputs from the user are sent here so that we may alter the program to the needs and wants of the user. For example; what parameters to set as our start or goal game-state, which algorithm to apply to a puzzle, which puzzle to solve, etc.

5. High-Level Design

5.1 High Level Context Diagram

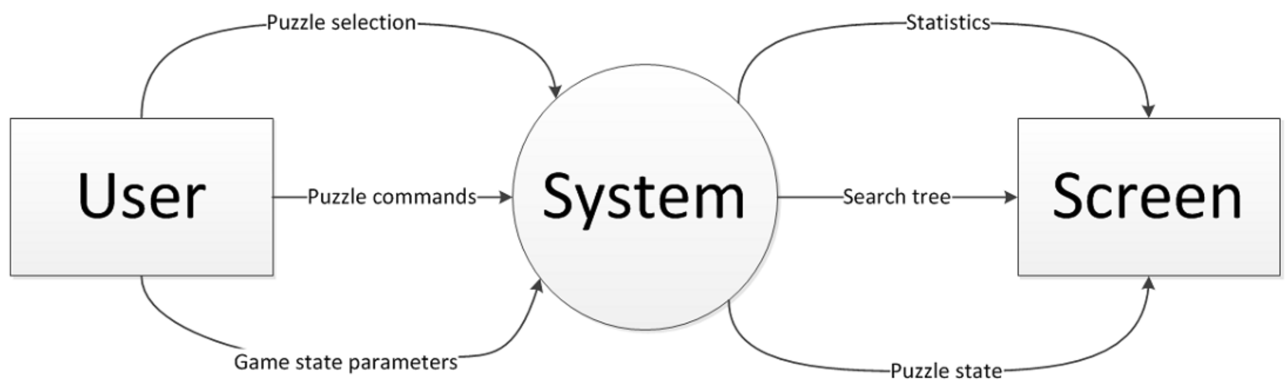


Fig 5.1 - The high level context diagram shows the relationship between the central system and the other entities such as the user and the screen. The various inputs and outputs between the system and the entities are shown here.

5.2 High Level Data Flow Diagram

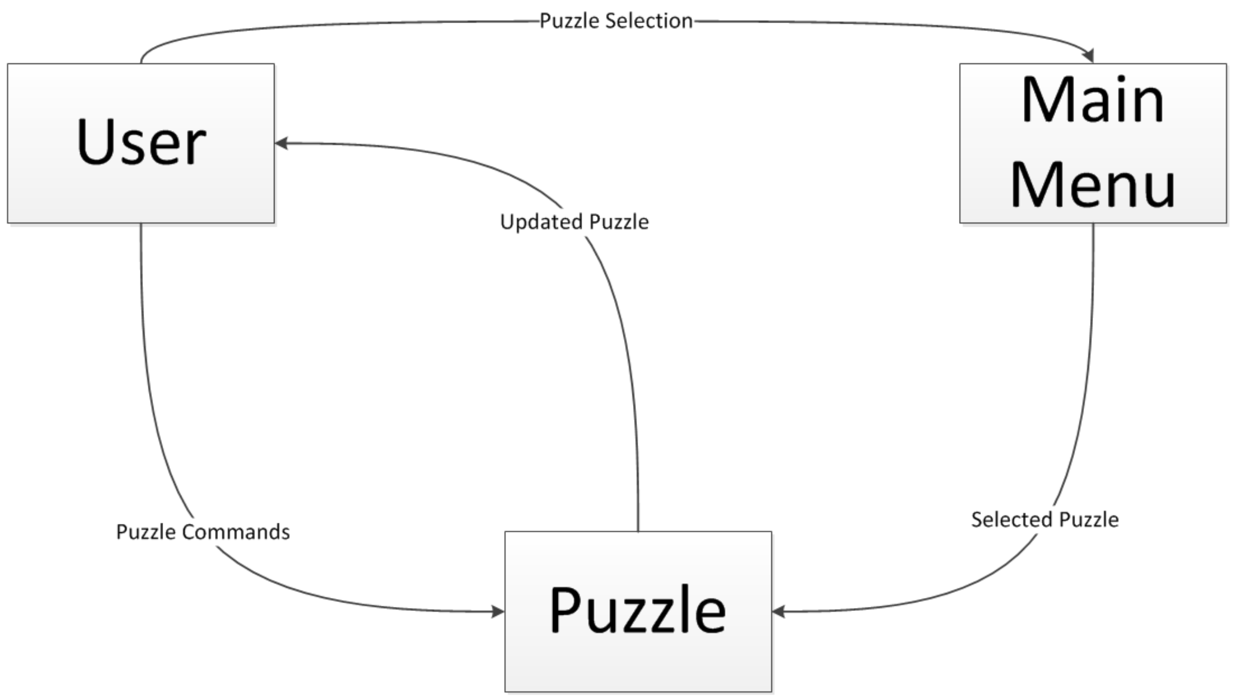


Fig 5.2 - The high level DFD shows how the user interacts with the system and what data is passed in order to carry out their specific requests.

6. Preliminary Schedule

6.1 Schedule

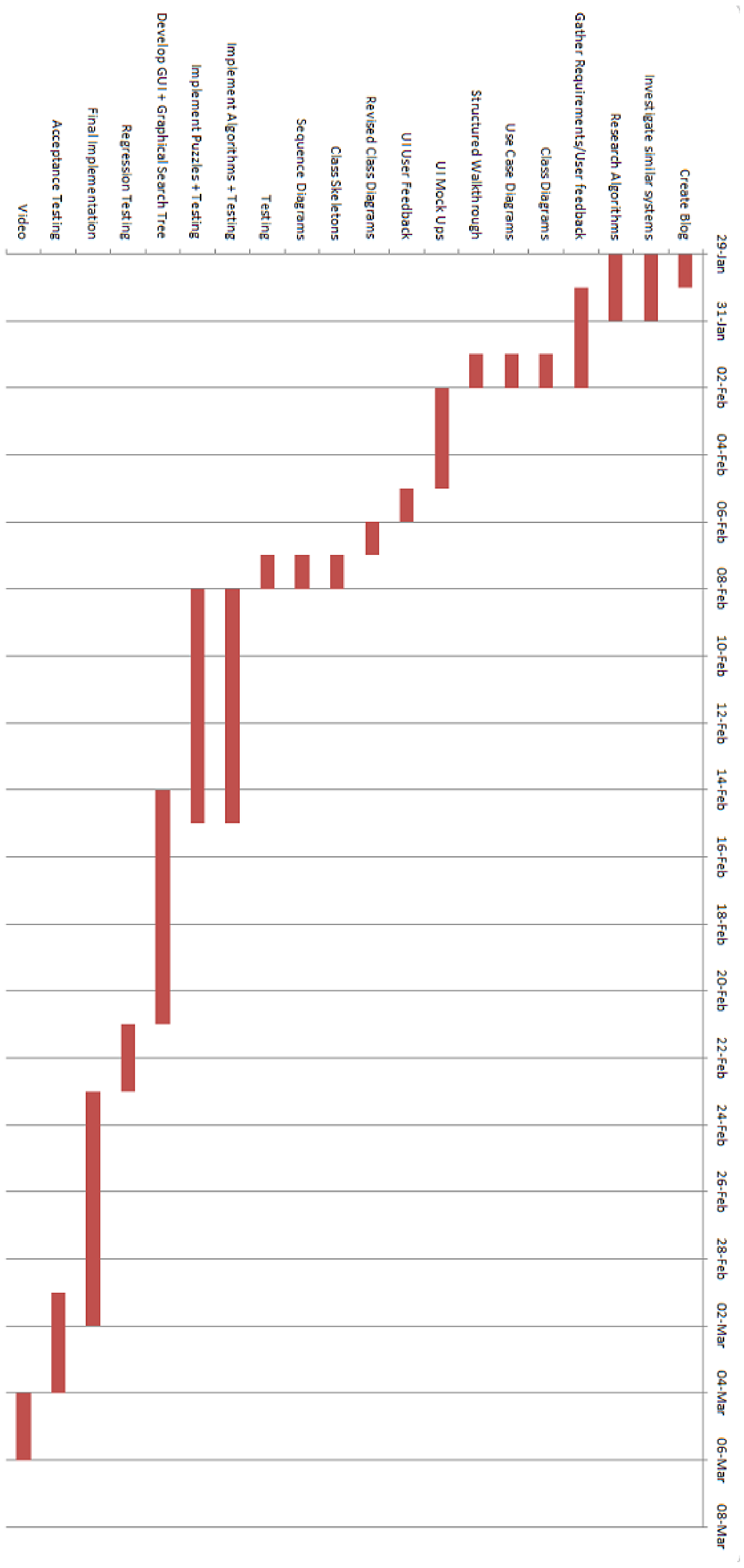
We decided to map out our schedule for our expected outcomes of our project. To do this we chose a Gantt chart to keep us on track. We chose a Gantt chart over a Pert chart as we had split up our tasks into many low level tasks. Below is our Gantt chart with all our tasks, start dates and due dates which we plan to stick by.

6.2 Gantt Diagram

Fig 6.1 - Our Gantt diagram presents our preliminary schedule. It was made using microsoft excel.

X-axis - Time in 2 day increments

Y-axis - Tasks


Create Blog
Investigate similar systems
Research Algorithms
Gather Requirements/User feedback
Class Diagrams
Use Case Diagrams
Structured Walkthrough
UI Mock Ups
UI User Feedback
Revised Class Diagrams
Class Skeletons
Sequence Diagrams
Testing
Implement Algorithms + Testing
Implement Puzzles + Testing
Develop GUI + Graphical Search Tree
Regression Testing
Final Implementation
Acceptance Testing
Video
29-Jan
31-Jan
02-Feb
04-Feb
06-Feb
08-Feb
10-Feb
12-Feb
14-Feb
16-Feb
18-Feb
20-Feb
22-Feb
24-Feb
26-Feb
28-Feb
02-Mar
04-Mar
06-Mar
08-Mar

7. Appendices

- https://www.smartsheet.com/blog/gantt-chart-excel
- https://www.pygame.org/docs/
- http://www.yiannisgabriel.com/2013/09/turning-hate-into-love-wordgame.html