Final Year Project Report

Full Unit – Interim Report

Playing Games and Solving Puzzles Using AI

Kieran Karmali-Truong

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Eduard Eiben

Department of Computer Science

Royal Holloway, University of London

Declaration

This report has been prepared based on my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 10735 approx. (Not including Bibliography and Appendix)

Student Name: Kieran Karmali-Truong

Date of Submission: 12/12/2023

Signature: KIERAN KARMALI-TRUONG

Table of Contents

Abstract

Artificial Intelligence (AI) has been portrayed to emulate human intelligence with its wide range of applications such as speech recognition in smartphones and decision-making within non playable characters in video games. An intriguing aspect of AI is Constraint Programming (CP), a problem-solving paradigm combining operation research, algorithms, and graph theory to solve combinatorial search problems. Another recent development of AI through countless research, I encountered an interesting heuristic search algorithm called A* search, a powerful pathfinding and graph traversal algorithm that finds the shortest path from a starting node to a goal node among numerous possibilities calculated in cost heuristics.

This project investigates Constraint Programming and heuristic search algorithms to solve complex puzzles like Sokoban. Sokoban challenges players to strategize and move objects within a constrained grided environment that requires logic, planning, and problem-solving skills. I aim to implement different AI methodologies from CP, that includes backtracking, branch and bound or other AI heuristic searches such as A* search and priority queue to provide efficient solutions to different Sokoban puzzles.

The project's objectives include developing a Graphical User Interface (GUI) that displays to the user a variety of Sokoban puzzles, each showing unique constraints and gradual challenges throughout each level. The implementation of GUI will be done via a cross-platform library called Pygame, that contains a range of modules required to develop any multimedia application such as 2D games. The AI will be tasked with finding the optimal solutions to these puzzles, all whilst maintaining consistent complexity for a satisfactory user experience.

Furthermore, as I continue to research into AI, this contribution will help with my individual growth of knowledge in AI problem-solving and its elements of pathfinding efficiently. The main purpose is to demonstrate how the usage of AI techniques can enhance the user experience and serve as a valuable abstract tool for addressing complex real-world challenges that require logical decision-making, whilst testing the limits of AI's complexity in finding an optimal solution.

This project represents different AI concepts and its complexity of puzzle-solving, portraying the potential of AI to replicate and possibly surpass human intelligence in solving intricate and tricky problems.

Chapter 1: Introduction

In pursuit of expanding AI, researchers have continuously look to challenge the existing theory further and test the limits of computational intelligence further to measure the extent of our knowledge and exploration capabilities. AI had been developed to challenge what human intelligence can do to create a better decision-making society.

The superiority of AI is noticeable in finding quick and easy solutions to problems; however, it can be difficult for AI to recognise the problem since they will need human intervention to ensure they understand the problem [1]. To combat this issue, I researched into constraint programming that provides techniques such as backtracking and branch and bound, for AI to function efficiently. Further analysis on the two CP methods will be explained within this report.

Further advancing AI's problem-solving capabilities clarifies the exploration of complex logical search methods, among which the A* search algorithm stands out due to its versatility in complex puzzle solvers. A* search demonstrates a strategic approach in navigating complex state spaces by efficiently searching through all possible moves to find the most optimal path to a solution [13]. This method is particularly useful in the context of game-playing AI, where the ability to predict and evaluate a multitude of multiple outcomes is a necessity. Its variants had been widely used in pathfinding problems and puzzle games for an efficient method of finding the shortest path to its goal state [12].

Lastly, the game Sokoban originated from Japan in 1981 created by Hiroyuki Imabayashi. This involves the user controls a character, allowing them push boxes to the correct location to solve the puzzle. The player is only able to move within a grid sized level that displays boxes and a targeted location for these boxes to be pushed into. The puzzle is complete when all the corresponding boxes are moved onto the correct storage location [5]. This is an intriguing game to experiment with, since the puzzles withing the game continue to get gradually difficult throughout each stage. Therefore, pushing the absolute limits of AI algorithms and enforcing a challenge between AI and human intelligence.

As I delve into the depths of the project, I want to emphasise on my reasoning for pursuing this aspect of the project. A general purpose of AI is the complexity and efficiency of its problem-solving ability. A purpose of this project is to use an abstract representation to effectively display the extent of AI methodologies and how it contrasts to human intelligence for better growth in knowledge, and I believe that Sokoban can display this belief successfully and achieve the project objectives.

Chapter 2: Aims and Objectives

In this section, I will outline the aims and objectives that I underwent during my project. The objectives were used to drive the project through every stage to a comprehensive understanding of AI usability and its variety range of techniques. Given the circumstances of my project, I will give a brief explanation to what my project achieved in its aims and objectives along with some revised aims and objectives that were pivotal to my project's final deliverables.

Chapter 2.1: My previous Aims and Objectives

- **Research:**
    - Gain an understanding of Constraint Programming techniques, E.g Backtracking, Branch and Bound.
    - Understand other search methodologies that reflect finding the optimal solution, E.g A* Search.
    - Search on its implementation on other AI solvers in different games.
    - Understand how Pygame works with its interaction with AI techniques.
- **Game Development techniques**
    - Present an accurate visualisation of the Sokoban game.
    - Use the features given from Pygame to produce the functionality of Sokoban.
    - Understand how to store data onto a GUI and load the recreated data.
- **AI Development techniques**
    - Give an option for the user to play the game or solve the puzzle using AI.
    - Understand how the solver can recreate a solution with efficient complexity.
    - Know how the searches can find the shortest possible path whilst considering costs e.g A* Search (heuristics).
    - To be able to replicate the pathway and visualise the solution to the user.
- **Testing:**
    - Understand the implementation of "ad-hoc" testing, writing the code first then performing tests.
    - Use effective and purposeful testing to develop optimal functionality on the code.
- **Revision Control / Software Engineering:**
    - Usage of effective branching, knowing what each section of code represents.
    - Give effective messages on every commit.
- **Project Management**
    - Develop practical planning and implement an effective timescale strategy.
    - Adaptive planning to ensure the project aligns with the objectives and timelines.
    - Prepare for future extensions to the project post-deliverables.
- **Reflection**
    - To be able to evaluate the development process, challenges and completed work.
    - Reflect on the progression made and skills gained throughout the project.
    - Understand how to improve on the setbacks given, through effective planning and understanding.
- **Documentation**
    - To be able to report and document the project progression and portray theory understanding.

Chapter 2.2: Reflected Aims and Objectives

This is the aims and objectives that reflect on my progression and have a brief description of what actions I performed to achieve them and whether I had completed them or not.

- **Research:**
    - Gain an understanding of Constraint Programming techniques, E.g Backtracking, Branch and Bound.
        - Yes, given the circumstances of what my code needed, the original given methods were not needed for my code to begin with. However, the research done for these methods were needed to understand the progression made towards achieving the final deliverables.
    - Understand other search methodologies that reflect finding the optimal solution, E.g A* Search, Priority queue.
    - Search on its implementation on other AI solvers in different games.
        - Yes, further research and a created prototype allowed me to help plan how to create my solver and how it is implemented in other games.
    - Understand how Pygame works with its interaction with AI techniques.
        - Yes, Pygame gave a platform in terms of how to create the game and does it work with AI.
    - Create a prototype that replicates A* Search in order to understand the concept of heuristic costs to a specific desired location.
        - Yes, I created a prototype of the ideal solution of a box heuristic towards a target and how the game moves an AI to a desired pathing.
- **Game Development techniques**
    - Present an accurate visualisation of the Sokoban game.
        - Yes, it is a variation of the Sokoban game, but the concept is maintained.
    - Use the features given from Pygame to produce the functionality of Sokoban.
        - Yes, the features maintain the playstyle of Sokoban that the user can interact with.
    - Understand how to store data onto a GUI and load the recreated data.
        - Yes, there is a created map that identifies each level, which can be saved and restored when requested.
- **AI Development techniques**
    - Give an option for the user to play the game or solve the puzzle using AI.
        - Yes, there are two buttons the user can interact with depending on how they want to encounter the puzzle. They can either solve the puzzle by clicking the solve button, or they can play the game by initiating a move on the arrow keypad.
    - Understand how the solver can recreate a solution with efficient complexity.
        - Yes, the solver is able to find the most efficient path considering the minimum costs of the box to the target, along with the player to the boxes.
    - Know how the searches can find the shortest possible path whilst considering costs e.g A* Search (heuristics).
        - Yes, in this project I made use of libraries like heapq to order the successors generated to get an ideal state. This is ordered by the cost of each successor, in which rendered the shortest possible path as it's solution. This project required two different heuristic calculations to determine the shortest possible path.
    - To be able to replicate the pathway and visualise the solution to the user.
        - Yes, the help of pygame features has allowed for processed visualisation of the game state. Whilst being able to show each step that was printed in the console.
- **Testing:**
    - Understand the implementation of "ad-hoc" testing, writing the code first then performing tests.

        - Yes, while running into potential bugs and issues concerning my code, I regularly conducted tests in order to find the simple concerns and issues involving my code.
    - Use effective and purposeful testing to develop optimal functionality on the code.
        - Yes, to figure out the issues concerning my code, I created purposeful print statements to adapt to bugs and find solutions.
- **Revision Control / Software Engineering:**
    - Usage of effective branching, knowing what each section of code represents.
        - Yes, I utilised on what each branch represented with meaningful names to represent each inclusion of code.
    - Give effective messages on every commit.
        - Yes, every commit includes meaningful information of what had been altered within the project.
- **Project Management**
    - Develop practical planning and implement an effective timescale strategy.
        - Given the difficult start, I had tried to keep consistent to my timescale in this term and given plenty of time to having a consistent plan first then working on a section of code.
    - Adaptive planning to ensure the project aligns with the objectives and timelines.
        - Yes, had to adapt to a slow beginning of the project and adapt the project plan in order to keep it on track towards its final deliverables.
    - Prepare for future extensions to the project post-deliverables.
        - No, unfortunately the project could not reach the possibility of including future extensions due to time constraints.
- **Reflection (This is in regard to the final report, so its objectives are represented later in the report)**
    - To be able to evaluate the development process, challenges and completed work.
    - Reflect on the progression made and skills gained throughout the project.
    - Understand how to improve on the setbacks given, through effective planning and understanding.
- **Documentation**
    - To be able to report and document the project progression and portray theory understanding.
    - Represent the documentation within the code with the use of docstring in python.
        - Yes, all the methods and functions of the code contain description of how it functions and its purpose.

Chapter 3: Literature Survey

This literature survey conveys the different type of methodologies, challenges, and approaches to the project itself, applying an expansive view on the subject itself. The implementation of AI in playing game and solving puzzles has been an expansive area of research, portraying the adaptability and problem-solving capabilities of AI algorithms in different scenarios.

An overview of the survey describes where AI has made significant progress to game playing and puzzle solving.

Relating to my project, AI researchers used game-playing to portray the gaps of knowledge between humans and AI. The survey example is DeepBlue, the first chess playing system, used heuristic searching, reinforcement learning, deep neural networks, and Monte Carlo Tree Search to find the most optimal method to play the game, ultimately beating a world chess champion [2].

The literature highlights a range of AI techniques to overcome the chess champion. An example involves heuristic search algorithms, the exploration of a solution space and prioritising different paths based on set heuristic factors regarding the path. Referencing the literature, the World Chess Champion made a move, and within the search space the DeepBlue will search for the optimal solution whist in consideration with the heuristics [2].

Another historical study relative to my project involves AlphaGo, that has a similar usage of techniques as DeepBlue but a prior focus on MCTS and deep neural networks, to find the optimal path that outmanoeuvred the various Go players including a Go champion Lee Sedol in 2016 [14].

The literature demonstrates a detailed evaluation of the Monte Carlo tree search and how it was used to encounter against convolutional networks from Go players. It performs this by expanding tree nodes to be processed by the policy network and the output of each policy network is stored as probabilities for every action it undertakes [14]. This is calculated and processed to find the most likely move to disrupt their opposition’s board and optimal path to win the game.

Analysing the two studies, it reveals a diverse range of approaches to solving the problem. AI’s adaptive ability to find solutions quickly within search space demonstrates its power. Comparing them to my own project, DeepBlue utilise the extent of AI’s adaptive ability and can use multiple searches to find a specific solution, representing some of my project objectives. This became significantly important when developing a cognitive AI to withhold multiple states and determine which state became the most optimal goal. My project has different approaches but uphold similar objectives like the movement within a grided environment can be manipulated. Whereas AlphaGo includes its expansion on AI’s calculative findings to understand every position the board can take, and what move is the most probable to ensure the best solution, demonstrated in my project’s objectives and deliverables. In contrast to my project, I used a different approach than relying on trees, since it is a search algorithm that requires optimality of the goal state, than using probability to the correct path. Thus, this survey has demonstrated a diverse approach to my project and with further research, has contributed to the growth of my project aims and objectives.

Chapter 4: Programming Language Used

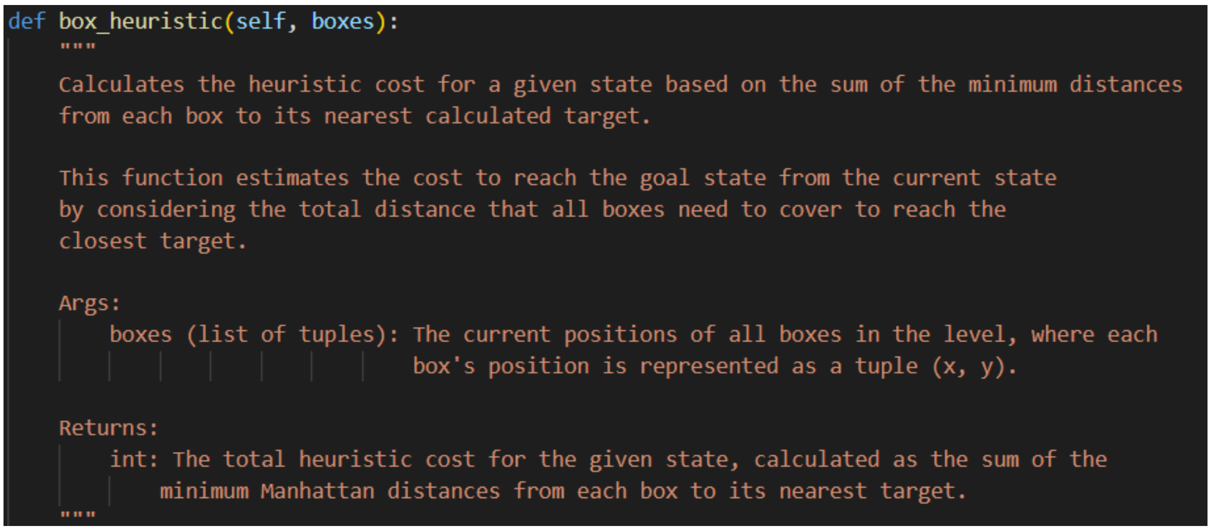
This chapter highlights my usage of the programming language and reasoning to why these foundations were used to program my project. Demonstrations of key methods used and why this interface may synergise with Constraint Programming and heuristic search algorithms.

Chapter 4.1: Python

Python is a high-level programming language that can synergise well with the AI methods mentioned in the objectives of my project due to its multiple features and characteristics of the language. I have listed a few reasons why Python is well-suited for implementing AI methods such as backtracking, branch and bound, and other AI search algorithms such as A* search that helped me achieve my project objectives:

- **Simplistic and Readable Syntax (Usage of Docstring):**

  Python's clear and readable syntax allows for the concise representation of complex algorithms. Portrayed to have an "uncluttered visual layout", due to its consistent usage of English keywords, whereas other programming languages such as Java, uses punctuation [9]. This shows the readability is beneficial when dealing with advanced AI methodologies, making the code more understandable and maintainable. Within the project, I maintained consistent usage of a documentation called Docstring to help gauge what the code does and its parameters along with its returns.



```
def box_heuristic(self, boxes):
    """
    Calculates the heuristic cost for a given state based on the sum of the minimum distances
    from each box to its nearest calculated target.

    This function estimates the cost to reach the goal state from the current state
    by considering the total distance that all boxes need to cover to reach the
    closest target.

    Args:
        boxes (list of tuples): The current positions of all boxes in the level, where each
                                box's position is represented as a tuple (x, y).

    Returns:
        int: The total heuristic cost for the given state, calculated as the sum of the
            minimum Manhattan distances from each box to its nearest target.
    """
```

  This example demonstrates all the features of Docstring and its purpose to keeping code maintainability in software development. I have a brief description of what the code does, below with a description of how it works. Next is the arguments the function takes from it's called predecessor and a description of what the argument is. Finally, is a small description of what the code returns at the end of the code.

- **Integration Capabilities:**

  Python's ability to easily integrate with other languages, such as C, C++, and JavaScript, is crucial when implementing AI algorithms into a program [9]. Thus, its effectiveness is relevant when connecting the AI components to the Pygame interface for game development.

- **Cross-Platform Compatibility:**

  Python's cross-platform compatibility allows for the simple deployment of AI solutions on other programming applications. This is crucial when considering the integration of AI algorithms within a game development framework like Pygame and find all the necessary modules needed to achieve the project's objectives [10].

- **Community Support for AI:**

  Python active community that constantly contributes to AI-related projects. An example is iRobot, where Python was used "to develop commercial robotic vacuum cleaners" [10]. This ensures an abundance of resources, documentation, and community support for implementing and troubleshooting AI algorithms thus a better understanding of the project.

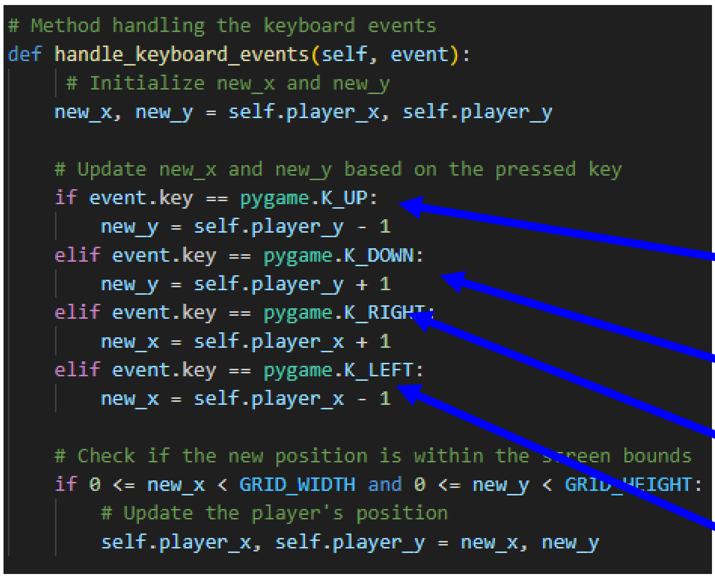
To summarise, Python's versatility, readability, extensive libraries, and community support make it an ideal choice for implementing the AI methods outlined previously in my project, ensuring a smooth integration of these methodologies into the Sokoban puzzle-solving context.

Chapter 4.2: Pygame

Using Pygame for my project, especially in the context of developing a Sokoban puzzle-solving game with integrated AI, offers several benefits. There are a few factors to why Pygame is a suitable choice for my project with some examples from my code:

- **Sprite and Event Handling:**

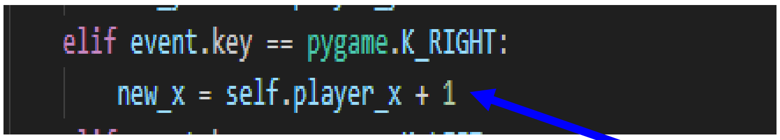
  Pygame provides convenient features for handling sprites and events, key components for the visualisation aspect of Sokoban. The sprite system allows me to manage game entities efficiently, while event handling ensures responsiveness to multiple user inputs, critical for the interaction aspect of my game.



```python
# Method handling the keyboard events
def handle_keyboard_events(self, event):
    # Initialize new_x and new_y
    new_x, new_y = self.player_x, self.player_y

    # Update new_x and new_y based on the pressed key
    if event.key == pygame.K_UP:
        new_y = self.player_y - 1
    elif event.key == pygame.K_DOWN:
        new_y = self.player_y + 1
    elif event.key == pygame.K_RIGHT:
        new_x = self.player_x + 1
    elif event.key == pygame.K_LEFT:
        new_x = self.player_x - 1

    # Check if the new position is within the screen bounds
    if 0 <= new_x < GRID_WIDTH and 0 <= new_y < GRID_HEIGHT:
        # Update the player's position
        self.player_x, self.player_y = new_x, new_y
```



```python
    elif event.key == pygame.K_RIGHT:
        new_x = self.player_x + 1
```

Demonstrated in my video, the user/player can move in four different directions within the grid using the "def handle_keyboard_events" method.

ARROW KEY UP -> event.key == pygame.K_UP:

ARROW KEY DOWN -> event.key == pygame.K_DOWN:

ARROW KEY RIGHT -> event.key == pygame.K_RIGHT:

ARROW KEY LEFT -> event.key == pygame.K_LEFT:

Once the user presses on the selected key, the sprite will move on the grid according to its position on the grid. For example, when the user is moving to the right and the default position of the player is at coordinates (1,1) in the grid. The player moves to the right as the coordinates of the player increases by 1 at the x position. Thus, making the position of (2,1) on the grid.

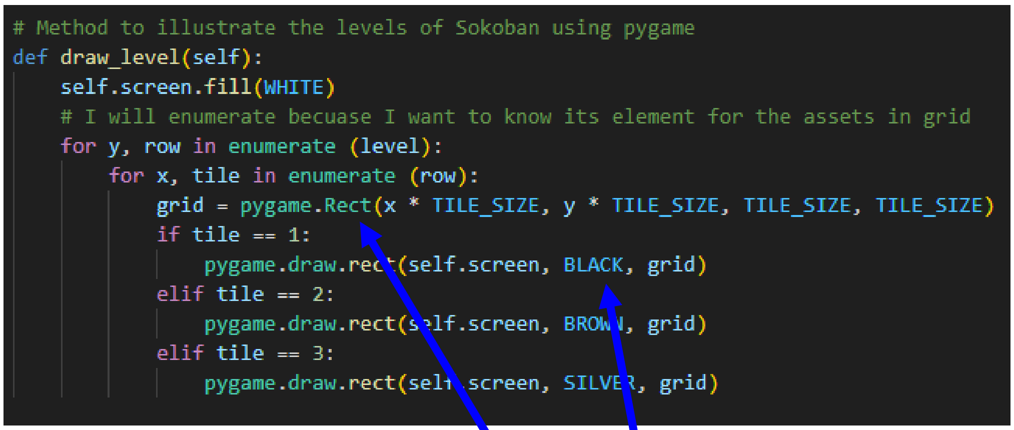
- **Usability in 2D games:**

  Pygame is specifically tailored for 2D game development, making it an ideal choice for implementing the Sokoban puzzle, which operates in a 2D grid environment as shown on my video. The simplicity of Pygame's design aligns well with the overall visualisation of my project.

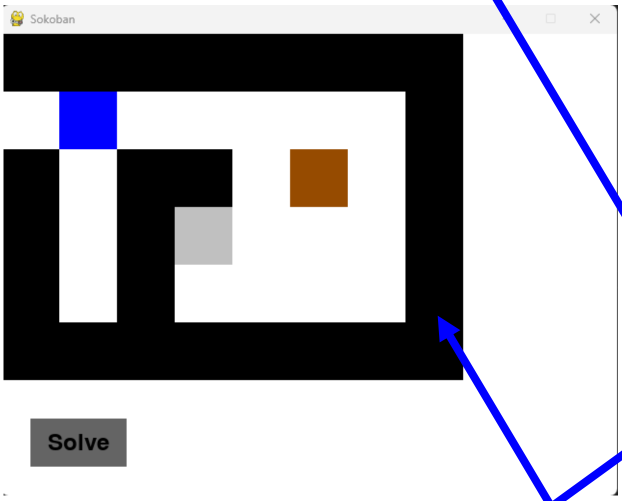
- **Simple Prototyping and Development:**

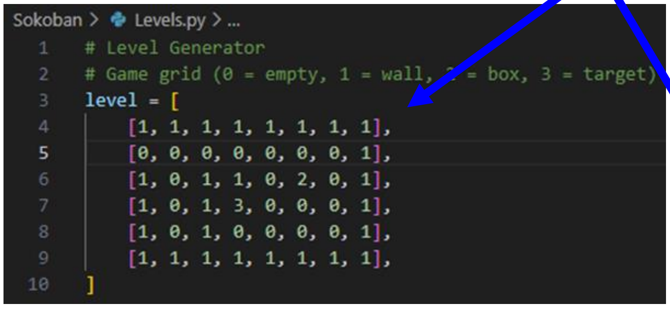
  Pygame's simplicity and straightforward design enable for quick and consistent prototyping through the usage of a simplistic design. This was crucial for my project, as it allowed me to quickly iterate through different game mechanics, level creation, and AI implementations to find the most optimal solution as quickly as possible.

An example of this is demonstrated in my code:



```
# Method to illustrate the levels of Sokoban using pygame
def draw_level(self):
    self.screen.fill(WHITE)
    # I will enumerate becuase I want to know its element for the assets in grid
    for y, row in enumerate (level):
        for x, tile in enumerate (row):
            grid = pygame.Rect(x * TILE_SIZE, y * TILE_SIZE, TILE_SIZE, TILE_SIZE)
            if tile == 1:
                pygame.draw.rect(self.screen, BLACK, grid)
            elif tile == 2:
                pygame.draw.rect(self.screen, BROWN, grid)
            elif tile == 3:
                pygame.draw.rect(self.screen, SILVER, grid)
```





```
Sokoban > Levels.py > ...
# Level Generator
# Game grid (0 = empty, 1 = wall, 2 = box, 3 = target)
level = [
    [1, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 0, 2, 0, 1],
    [1, 0, 1, 3, 0, 0, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1],
]
```

In this example, the grid is generated via an enumeration of a list of lists, representing a 2D array. Each element of the list represents a value in numbers. The numbers are portrayed as sprites that represent the visualisation of the grid shown in my documentation.

The grid is represented as its dependant TILE_SIZE which revolves the grid being drawn out using the "pygame.Rect" method.

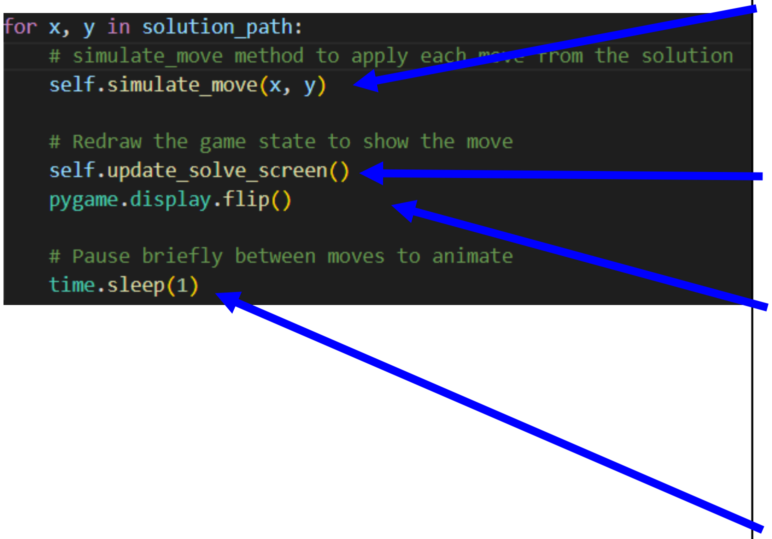
Pygame.Rect is an object for storing rectangular coordinates that passes the parameters (left, top, width, height). The calculations made involved using the constant TILE_SIZE multiplied by the the x and y coordinates to represent each tile in the grid.

Pygame.draw.rect is a module for drawing rectangles that passes the parameters (surface, color, rect). With the example "pygame.draw.rect(self.screen,BLACK,grid)" , we can see that whenever the element is 1 represented on the Level variable, the tile drawn will be black representing the walls of the grid.

The simplicity is represented here since the level does not seem visually sophisticated but maintains functionality as it is just a basic prototype to the final objectives.

- **Versatility for AI Integration:**
  Pygame's versatility allows for a smooth integration of AI methodologies into the Sokoban game. Since Pygame is the cross-platform to Python, it maintains the programming language syntax therefore, it is still compatible with methods unrelated to Pygame. Whether it's controlling the user's movements or implementing the AI-driven solutions for hints to solve the Sokoban puzzles, Pygame provides the flexibility needed for these integrations.

An example of its flexibility with AI integration is the animation of the solution path:



```
for x, y in solution_path:
    # simulate_move method to apply each move from the solution
    self.simulate_move(x, y)

    # Redraw the game state to show the move
    self.update_solve_screen()
    pygame.display.flip()

    # Pause briefly between moves to animate
    time.sleep(1)
```

This code displays each step in the solution path shown in the video, as it simulates the player's movements. It does this by updating the game screen to show the path states and pauses briefly to show the animation to the user.

self.simulate_move(x,y): This method calculates the new position of the player based on the solution path’s x and y offsets and then attempts the move.

self.update_solve_screen(): This method clears the screen, redraws the level and the player in their current states, and then updates the display.

Pygame.display.flip(): This is Pygame compatibility with AI, used to update the entire screen by swapping the back buffer with the front buffer. This function helps in displaying all the graphics smoothly and used to complete the frame.

Time.sleep(1): From the time library, used to slowly display the states to the user by briefly pausing the animation in between moves.

This demonstrates the flexibility of pygame complementing with AI techniques as it establishes the link to continue applying its complex logic through pygame’s drawn elements.

To conclude, Pygame is a well-suited choice to represent the visualisation of the Sokoban puzzle-solving game with AI integration due to its ease of use, cross-platform compatibility mentioned in the Python section, 2D game development focus, versatile features, and compatibility with Python. These factors all contribute partly towards a successful development process, to better understand my project’s objectives.

Chapter 5: AI Implementation

In this section, the clarification of both constraint programming (CP) and search heuristic algorithms will be displayed and the power it holds with its implementation of AI techniques. I will go through the comparison between the two since they heavily impacted the decisions, I made to complete my project's aims and objectives. Furthermore, showcasing its techniques and displaying how I manage to implement my AI into the Sokoban puzzle.

Chapter 5.1: Constraint Programming & Heuristic Search Algorithms

Constraint programming and heuristic search algorithms are two fundamental approaches AI for solving puzzles and other complex problems. They have distinct characteristics and impacts on the effectiveness and efficiency of AI solvers. During my project, a critical analysis and evaluation was performed to understand their differences and applications that can provide insights into choosing the appropriate method for this Sokoban project.

- **Constraint Programming**

  Constraint programming involves formulating a problem in terms of constraints on acceptable solutions. This approach is typically used for problems where the solution requires meeting a set of pre-defined conditions. The solver searches for solutions by trying to satisfy these constraints.

  Strengths:

  Efficient in Specific Problems: Constraint programming is highly efficient for problems well-defined through constraints, these include scheduling, planning, and resource allocation [7]. Problems like Sudoku where it has multiple unnecessary solutions can be easily defined and deleted after described thoroughly in a constraint [15].

  Domain Reduction: Touched on the previous strength, it reduces the search space significantly by pruning possibilities that do not meet the constraints. This is efficient when having a randomly generated problem since it can discard on any solutions that does not meet the constraints set [4].

  Weaknesses:

  Scalability: Performing exhaustive depth-first searches can become inefficient if the number of constraints becomes very large or if the constraints are too complex, as the process of checking all constraints can be computationally expensive.

  Flexibility: Less flexible in problems where the constraints are not well-defined or are too dynamic [7].

- **Heuristic Search Algorithms**

  Heuristic search algorithms guide the search for solutions by using heuristics to estimate how close a given state in the search space is to a goal state. Algorithms like A*, greedy best-first search, and others prioritize moves based on heuristic information that predicts how promising a particular move is [13].

  Strengths:

Speed and Efficiency: By prioritizing paths that appear to be the most likely towards the goal, heuristic searches can find solutions quicker than methods that explore all paths equally, comparing the costs of each state. [14].

Adaptability: They are highly adaptable to a variety of problems, especially where an approximate solution is more feasible than an exhaustive search. The adaptability can allow for versatile problem solving, since it can adjust to the given problem without having to worry about too many specific constraints.

Optimality and Completeness: Algorithms like A* are both optimal and complete, ensuring the best solution is found if the heuristic has allowed the solution.

Weaknesses:

Resource Intensity: These algorithms can be resource-intensive, consuming significant memory and time, particularly in large or complex search spaces [13].

To summarise, the choice between constraint programming and heuristic search algorithms depends on the nature of the desired puzzle, the clarity of the problem constraints, and the required efficiency of the solution. My project required a specific path finding puzzle since it does not require multiple defined goals so, I opted to an A* Search approach. However, each approach offers powerful tools for AI development, with their respective strengths making them suitable for different types of challenges in puzzle solving. Thus, depending on specific specifications the puzzle or project undertakes, other methods might become feasible in classifying different AI techniques.

Chapter 5.2: Considered Methods Explained

Upon further analysis on constraint programming search algorithms, there are two methods that were considered when tackling the aims and objectives:

- **Backtracking:**

  Backtracking performs a depth-first search of the solution space, where variables are instantiated to a constraint. To solve my project problem, I will need to create a domain variable that controls the character, moving in four directions [6]. Every character movement is tied to a variable through the backtracking method. Any instantiated variable that violates the constraint are backtracked to the recent instantiated variable with alternative values available [4]. This continues until the given optimal solution has been found.

- **Branch and Bound:**

  Branch and bound where it involves depth-first traversal of a search tree. Within Sokoban and the representation of AI in general, we seek for the optimal solution via a tree with lower bounds and upper bounds [8]. The lower the bound, the lower the cost incurred by the solver on finding the optimal solutions against the constraint problems. We would use the upper bound to prune branches that do not contain the solution [7]. For instance, if the AI contained the first move to move left at the beginning but it does not have any pathway of finding the ideal solution, the movement is pruned, and the AI controlled player does not move left as the first move.

In conclusion, CP, and its search algorithms, including backtracking and branch and bound, showcases my project's commitment to researching advanced AI techniques for efficient and optimal puzzle-solving. Despite their significant strengths to finding an efficient solution in not only my project but other AI solvers as a collective, I found that using a heuristic search algorithm was the most efficient way to generating a solution path for Sokoban.

Chapter 5.3 Used Methods Explained

After learning different possibilities my project could efficiently create a solution, I opted to choose a heuristic search algorithm, and the technique that understood the project's final deliverables was A* Search.

A* Star Search

A* (A-star) search is a powerful pathfinding and graph traversal algorithm. It finds the shortest path from a starting node to a goal node among numerous possibilities, utilizing both actual costs (from the start to the current node) and heuristic estimates from the current node to the goal to determine the path that minimizes the total estimated cost. The node with the lowest estimated cost is selected to be evaluated next [16].

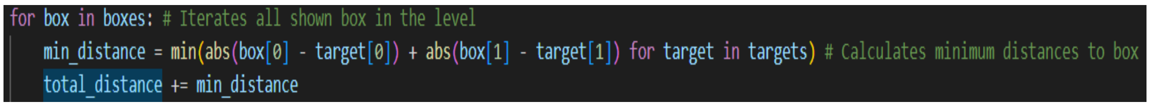
- **Costs of A * Search:**

  g(n): The actual cost from the start node to any node n [16].

  h(n): The heuristic estimate of the cost from n to the goal. This cost should never overestimate the actual cost to reach the goal [16].

  f-score: Estimated total cost of the cheapest solution through n. It is calculated as: f-score = g(n) + h(n) [16].

  These costs are represented within my code using the heuristic function, the box_heuristic function represents the distances and the calculated f-score:



```
for box in boxes: # Iterates all shown box in the level
    min_distance = min(abs(box[0] - target[0]) + abs(box[1] - target[1]) for target in targets) # Calculates minimum distances to box
    total_distance += min_distance
```

  This function estimates the cost to reach the goal state from the current state by considering the total distance that all boxes need to cover to reach the closest target.

  As the box positions is iterated in the set, it calculates the Manhattan distances (the sum of absolute differences between real vectors) of the box to the target. These heuristic respects the constraints of the game and helps prioritize moves that appear to lead boxes closer to targets:

  - **min_distance:** this contains the operating costs for h(n) that A* Search requires. It is to estimate how close each box is to being placed on a target, which is critical in puzzle games like Sokoban where the goal is to move all boxes to designated target locations. It is calculated for Manhattan distance between two points (x1,y1) and (x2,y2) is |x1 – x2| + |y1 – y2|. This calculation represents the shortest path that every iterated box could be moved directly to a target [17]. An example could involve:

    Suppose you have a box at position (1,2) and targets at positions (3,2) and (1,5). The Manhattan distances to these targets would be:

    To (3,2): $|1-3|+|2-2|=2$

    To (1,5): $|1-1|+|2-5|=3$

    The min_distance for this box would be the minimum of these two values, which is 2. This value would then contribute to the heuristic cost used by the A* algorithm to prioritize this node expansion.

- **total_distance:** This adds up all the boxes within each level's estimated costs to reach the goal from the current state, contributes directly providing the overall cost in the solve_level.

- **Function of A* Search:**

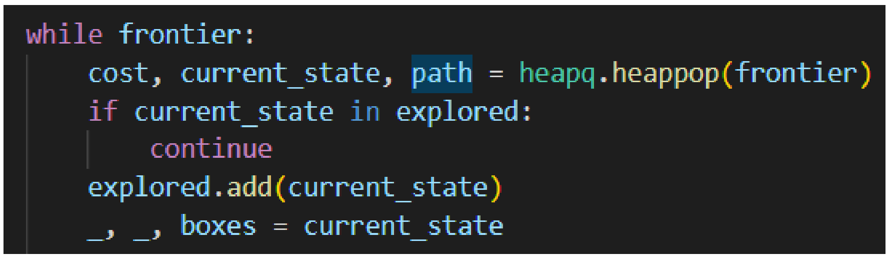
A* maintains a priority queue, represented in the heapq library, where it stores all nodes to be explored, with nodes prioritized by the lowest f-score value [16]. The algorithm repeatedly extracts the node with the lowest f-score and explores all the generated successors:

A mini pseudocode representation:

```
While goal node
    For each successor, it calculates g, h, and consequently f-score.
    If a successor has not been visited or a cheaper path to it is found
    Updated or added to the priority queue
```

This process continues until the goal node is dequeued, at which point the optimal path has been found, or the priority queue is emptied, shows no solution [16].

The solve_level touches on the function in depth, and how A* search function is applied:



```python
while frontier:
    cost, current_state, path = heapq.heappop(frontier)
    if current_state in explored:
        continue
    explored.add(current_state)
    _, _, boxes = current_state
```

This code represents function of how A* Search begins its operations that represent 'goal node':

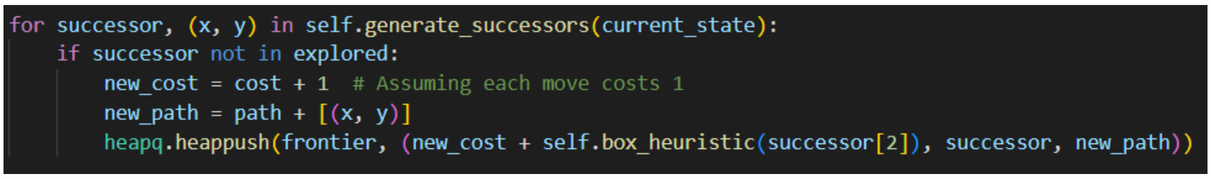
- **While Frontier:** The while loop continues as long as there are states left in the frontier (priority queue). It stores states of the game along with their associated costs and the path taken to reach them.
- **cost, current_state, path = heapq.heappop(frontier):** The main function to this is to extract the state with the lowest cost.
  - **heapq.heappop(frontier):** This function pops the state with the lowest cost from the priority queue. Each element in the frontier is a tuple containing:
  - **cost:** The current cost to reach this state, which may include both the actual path cost so far and the heuristic cost.
  - **current_state:** The current state of the game, which includes information like the player's position and the positions of all boxes.
  - **path:** The sequence of moves taken to reach this state from the start state.
- **if current_state in explored: continue:** This line checks if the current_state has already been explored. If it has, the algorithm skips the remaining part of the loop and moves on to the next iteration. This prevents redundancy in the code, processing the same state multiple times and ensure efficiency.
- **explored.add(current_state):** This adds the current_state to a set of explored states, indicating that this state has been processed.
- _, _, boxes = current_state: This line decomposes the current_state with the use of underscores _ are placeholders indicating that the algorithm is discarding some parts of the

current_state's tuple. The variable boxes capture the positions of the boxes, which is significant for checking the goal state and generating successors.

This code successfully captures the process of the 'goal node' since it truly evaluates the states of which the puzzle is allowed to be in. Each state in Sokoban can be represented as a combination of the positions of the player and all boxes. A state transition occurs when the player moves or pushes a box to a new position. The constraint factors from the heuristics are used to evaluate which of the states have the lowest cost, thus dequeuing that state after being processed. This continues until all the states have been evaluated. The goal is achieved when all boxes are placed on all targets. This is valued state is known as the 'goal state' represented in my code. The A* search stops when it dequeues a state that meets this goal condition.

To represent the successors' role (which are generated states that simulate the player's actions) in A* Search, both the solve_level method to demonstrate the generation of states and its estimated cost:

solve_level:



```
for successor, (x, y) in self.generate_successors(current_state):
    if successor not in explored:
        new_cost = cost + 1  # Assuming each move costs 1
        new_path = path + [(x, y)]
        heapq.heappush(frontier, (new_cost + self.box_heuristic(successor[2]), successor, new_path))
```

The code demonstrates the crucial part of A* search algorithm's implementation for generating and exploring all possible successor states from the current state in the Sokoban game. Also demonstrates the heuristic cost formula, generating the f-score = g(n) + h(n) [16]. Furthermore, this continues when the heapq (priority queue) is still running through a while loop. Here's what each line does:

- **for successor, (x, y) in self.generate_successors(current_state):** This line calls the generate_successors method on the current game state, which computes all possible states that can be reached from the current state by making one valid move. **successor** is a new game state resulting from a generated move, and **(x, y)** is the tuple representing the move made from the current state to reach the successor state. This move could involve moving the player up, down, left, or right, and potentially pushing a box, represented from the generate_succesors method.
- **if successor not in explored:** This line checks if the successor state has already been explored. If it hasn't, the algorithm proceeds to process this successor and evaluates it.
- **new_cost = cost + 1:** The cost to reach the new state (new_cost) is calculated by adding 1 to the current state's cost (cost). This implies that each move has a calculated cost of 1, regardless of what the specific action was taken. Whether it's the player's movement or pushing the box. This represents the g(n) part of the formula as the **cost** keeps track of the total cost to reach the current state from the start node.
- **new_path = path + [(x, y)]:** This line constructs the new path to reach the successor by appending the current move (x, y) to the path used to reach the current state. This updated path represents the sequence of moves taken from the initial state to the successor state. In other words, it represents the as part of the calculation to determine the estimated heuristic cost, and finding the most optimal path.
- **heapq.heappush(frontier, (new_cost + self.box_heuristic(successor[2]), successor, new_path)):** This line pushes the new state into the frontier (priority queue). This represents the given f-score that is calculated to determine the goal state.
  - **self.box_heuristic(successor[2]):** This method calculates a heuristic cost for the successor state, which estimates the minimum cost from the successor state to the goal state. The heuristic measures the sum of the shortest distances from each box

to the nearest goal position, described in a previous code analysis. This represents the h(n) part of the formula.

- **successor[2]:** This part of the successor tuple that contains the specific information needed by the heuristic function to calculate its value, like the positions of boxes.

This code segment effectively implements the core functionality of the A* algorithm, managing the exploration of game states, that prioritizes those that has the lowest cost according to the sum of actual path cost and estimated remaining cost, which are dequeued after the evaluation of every state is made. This prioritization helps find the optimal solution efficiently by exploring more promising paths efficiently.

- **A* Search's Role in my Project**

  A* Search indicated a clear role that lasted within my project, in order to achieve the final aim's and objectives. The usefulness of the search provided efficient optimal pathfinding, that provided solutions to this project's problems, in order to provide the final deliverables:

  - **Deadlocks:** Certain positions can render a box immovable (deadlocks), like a box getting stuck in a corner not adjacent to a target. Advanced solvers include mechanisms to identify and avoid these states.
  - **Multiple Solutions:** Multiple paths might lead to the solution. A* inherently handles this by always pursuing the path with the lowest estimated total cost.
  - **Complexity:** Sokoban levels can vary greatly in complexity. Heuristic functions do not always capture the intricacies of each level, potentially leading to inefficiencies in pathfinding.

The implementation includes methods like generate_successors to explore possible moves, box_heuristic to compute heuristic values, and solve_level to control the elements of A* search. Each method contributes to handling the complexities of Sokoban within the framework of A* search, leading to an effective solver for many other AI puzzle solvers.

Chapter 5.4 Conclusion

To conclude, the choice of A* Search over other methods like Backtracking and Branch and Bound was decided by several key aspects of the problem and the algorithm's characteristics that align well with the game's requirements and the project's final deliverables. If we look at the elements of Sokoban, we can consider the key features to determine why A* Search was chosen for my project:

- **Optimal Pathfinding:** Sokoban not only requires solving the puzzle but also often seeks the shortest solution in terms of moves.
- **Complexity and Scalability:** Levels can vary greatly in layout and complexity. Sokoban is known for its increase of difficulty on levels that human's struggle to solve, therefore a complex and effective AI solver is necessary to solve the puzzle [5].
- **Real-time Solution:** The need for a method that efficiently navigates through complex search states to provide solutions within a short and consistent time frame.

A* Search throughout research, is known for its efficiency and ability to find the most cost-effective path to the goal. It combines the benefits of the search techniques by using a heuristic to estimate the cost to the goal state and a cost function that tracks the path length from the start [12]. In contrast to, both CP methods that may be inefficient for large or complex puzzles like Sokoban. Backtracking is great with handling constraints, however exploring paths recursively without any heuristic guidance could lead to a significant exploration of non-viable paths [6]. Branch and bound could be effective in finding optimal solutions, but typically performs better in problems structured around optimization rather than problem-solving puzzles like Sokoban [7].

The selection of A* Search for the Sokoban AI solver was based on its proven capabilities in pathfinding within grid-based puzzles, its efficient use of heuristics to guide the search, and its optimal performance in both theory and practice. This made it a superior choice compared to Backtracking and Branch and Bound, which, while powerful methods, do not meet Sokoban’s challenging and complex deliverables as optimally as A* Search. It not only meets the project’s needs for efficient and effective puzzle solving AI but also ensures that solutions are found optimally, making it an ideal technique to create an AI solver for my project.

Chapter 6: Software Engineering

This chapter discusses the importance of revision control and testing to achieve the requirements of being a software developer. This involves my progression in the GitLab and the approach to planning my testing done in term two.

Chapter 6.1: Revision Control (GitLab)

GitLab, a web-based Git repository manager, provides a comprehensive set of tools and features that bring significant importance to software engineering and can greatly benefit for its usage in my project. Some key aspects of GitLab's importance includes:

- **Version Control:**

  GitLab serves as a robust version control system, enabling software engineers to track changes, collaborate, and manage different versions of the source code. This ensures that the project's code is well-organized and structured, allowing for smooth working on its features or bug fixes without conflicts.

  For instance, the branches I created on my GitLab represent efficient version control through concise branch names like input-events. Utilising the aspect of branching, I can be diverse on different aspects of my code to see bug fixes and functionality issues to then use efficient merging whenever I finish the branch of the code.

- **Visibility:**

  GitLab provides visibility into the project's development process. From code changes to deployment status, developers can easily track progress and have a clear understanding of the state of the project at any given time.

  An example revolves the usage of my commits, demonstrate great visibility since I can easily track where specifically my code changes since the previous commit. This is beneficial since I can plan where I should approach the next problem ahead when given a clear concise message from my commit for better understanding of project objectives.

- **Documentation:**

  GitLab allows developers to create and maintain documentation within the repository. This is a crucial feature for maintaining a comprehensive and updated knowledge base, making it easier for developers to understand the project structure, progression, and code.

  Demonstrated in the text files like the diary and the README.md, this represents the progression made in the code thus far. Providing updates on my aims for the week to progress towards the project's overall objectives.

To summarise, GitLab brings an importance to software engineering by providing a platform for version control, visibility, and documentation. For my project, GitLab enhances my code quality, update my progression for better understanding, and structurally present my project effectively.

Chapter 6.2: My Testing Plan

I decided to give a detailed description of my plans for testing under this section, since it represents a valuable aspect of software development and the progressions for my project. Testing is a critical and integral part of the software development process, playing an impactful role in ensuring the quality, reliability, and performance of software applications. This section will provide the specific tests that I had applied in my project to ensure a smooth software development process.

My approach is a simplistic method to software development through ad-hoc testing however, I believe this will help me achieve the project's objectives efficiently and comfortably. Ad-hoc is testing and exploring the software's functionality without predefined test cases before coding. Some key benefits to ad-hoc testing includes:

- **Unexpected Scenarios:**

  My game may experience tricky bugs that become unnoticeable at first glance. Ad-hoc testing allows me to do consistent testing to discover behaviour within a class to fully exploit its functionality [11]. This is beneficial for exploiting usability issues and scenarios that may not be covered by any predefined case I will test.

- **Effectively Structure Testing Plans:**

  During testing, I can prioritise testing strategies depending on performance of other tests. For instance, if I test the feature for the AI to move the player and can push the box along the grid and it works. I can adapt my focus to another test to efficiently test the project objectives [11].

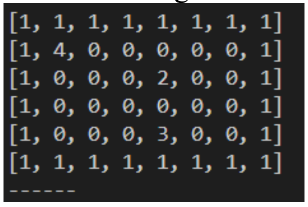
Here I will elaborate on the tests that were performed in question to the project objectives as an outlined perspective:

- **Visualisation of Sokoban**
  - Does it display the correct sprites?
  - Does it generate the level correctly?
- **Functionality of Sokoban**
  - Can the user move around the grid?
  - Can the user push the box?
  - What happens when the player collides with the grid boundary?
  - Does the level end when all boxes are moved to their desired location?
  - Can the AI move around the grid?
  - Can the AI move the boxes in an efficient consistency?
  - When requested by the reset button does the level revert to its original state?
- **AI implementation of Sokoban**
  - When the solve button is pressed, what will be visualised?
  - Does the AI eliminate other pathways to the solution?
  - Does the AI find the quickest path to the solution if so, how does it know?
  - Does it generate heuristics correctly? What heuristics benefit the project?
  - Does it generate states/successors as accurate as possible?
  - Does the correct state become dequeued from the queue?

Chapter 6.3 Results & Testing Reflection

Using the outlined tests, I will determine whether the tests results of them satisfied each test case, some of which including evidence from the print stated console, which were used to determine these tests, and debugging for a solution:

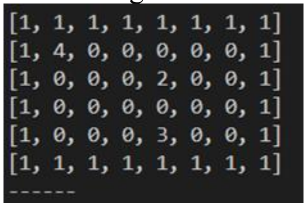
- **Visualisation of Sokoban**
  - Does it display the correct sprites?
    - Yes, the sprites consistent of boxes that represent different elements of the game shown in the video.
  - Does it generate the level correctly?
    - Yes, uses clear and effective tuple creation, to demonstrate a grid level of the game. Each level is represented by a dictionary and is called when it is the current key when switching between levels.



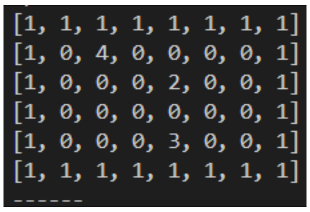
```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 4, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```

Replicated successfully from the levels.py module.

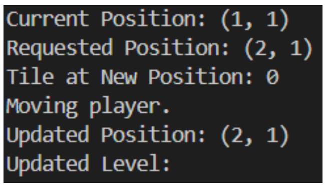
- **Functionality of Sokoban**
  - Can the user move around the grid?
    - Yes, demonstrated in the video and on the console grid, the player can converse within the grid and set walls.



```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 4, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```



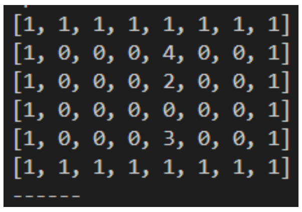
```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 4, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```



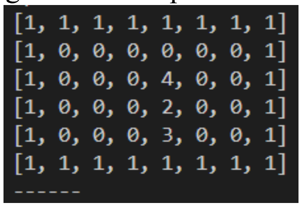
```
Current Position: (1, 1)
Requested Position: (2, 1)
Tile at New Position: 0
Moving player.
Updated Position: (2, 1)
Updated Level:
```

Updated when the player requests to move from one position to the requested position.

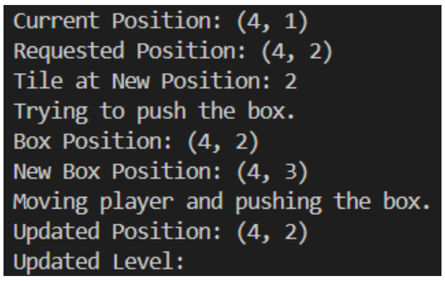
  - Can the user push the box?
    - Yes, the player can successfully move the box within the grid, can encounter deadlocks that the user could cause when attempting to solve the puzzle.



```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 0, 4, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```



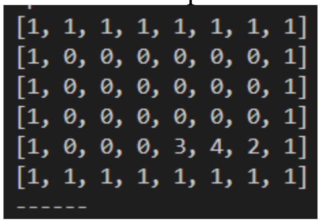
```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 4, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```



```
Current Position: (4, 1)
Requested Position: (4, 2)
Tile at New Position: 2
Trying to push the box.
Box Position: (4, 2)
New Box Position: (4, 3)
Moving player and pushing the box.
Updated Position: (4, 2)
Updated Level:
```

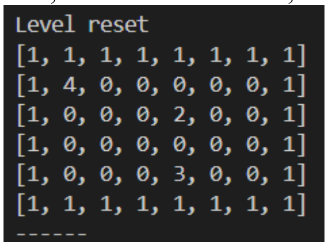
Updated when the player and box requests to move from one position to the requested position. Both showing their updated positions.

Deadlock example:



```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 4, 2, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```

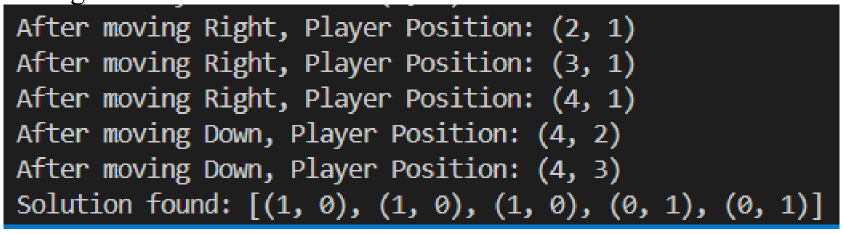
- What happens when the player collides with the grid boundary?
  - Yes, demonstrated in the video when the character collides into the wall, player stops moving in the requested direction by the user.
- Does the level end when all boxes are moved to their desired location?
  - Yes, shown in the video the level switches when the game moves from one level to the next when all the boxes are moved to the target.
- Can the AI move around the grid?
  - Yes, shown in the video when the user presses the solve button, the player is seen moving around the grid without any user input handling.
- Can the AI move the boxes in an efficient consistency?
  - Yes, creates an efficient pathing in every predefined level that leads the player to the boxes and then to the target.
- When requested by the reset button does the level revert to its original state?
  - Yes, shown in the video, the game resets successfully when the user requests it.



```
Level reset
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 4, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 2, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 3, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
------
```

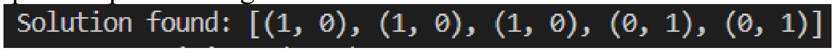
Shown in the log the level resets when the game has requested for it to be reset.

- **AI implementation of Sokoban**
  - When the solve button is pressed, what will be visualised?
    - As the button is pressed it will update the screen and display the solution animated to the user. Showing the most optimal path, the best movement to the box and box to the target. This is shown within the demonstration video.
  - Does the AI eliminate other pathways to the solution?
    - Yes, revealed in the video that it only shows one solution to every predefined level in the game.



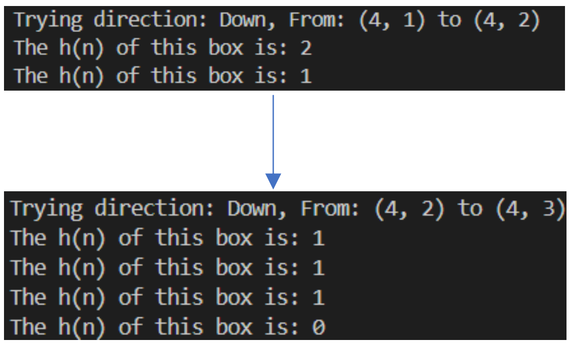
```
After moving Right, Player Position: (2, 1)
After moving Right, Player Position: (3, 1)
After moving Right, Player Position: (4, 1)
After moving Down, Player Position: (4, 2)
After moving Down, Player Position: (4, 3)
Solution found: [(1, 0), (1, 0), (1, 0), (0, 1), (0, 1)]
```

  - Does the AI find the quickest path to the solution if so, how does it know?
    - Yes, it is able to search through different states of the level and show a definitive quickest path to the goal state.



```
Solution found: [(1, 0), (1, 0), (1, 0), (0, 1), (0, 1)]
```

  - Does it generate heuristic costs correctly? What heuristics benefit the project?
    - Yes, it is able to get the minimum value resulting from the Manhattan formula and get absolute value for each state the successor is accessible to.

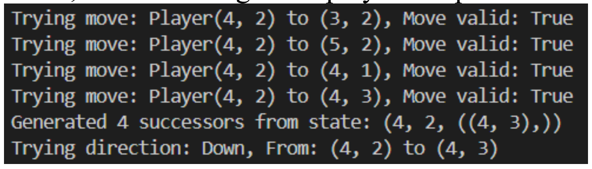


```
Trying direction: Down, From: (4, 1) to (4, 2)
The h(n) of this box is: 2
The h(n) of this box is: 1
```

```
Trying direction: Down, From: (4, 2) to (4, 3)
The h(n) of this box is: 1
The h(n) of this box is: 1
The h(n) of this box is: 1
The h(n) of this box is: 0
```

Represents the cost for the box iterated through every state. The closer the box is to the target, the closer smaller the cost gets when in that state. For example (4,1) to (4,2) represents the h(n) of 1 because that state moves the box closer to the target within 1. The state (4,2) to (4,3) costs 0 because the box takes up the target's position.

- Does it generate states/successors as accurate as possible?
  - Yes, it is determined from the method is_move_valid() and generates multiple states, authenticating if the player can perform this action.



```
Trying move: Player(4, 2) to (3, 2), Move valid: True
Trying move: Player(4, 2) to (5, 2), Move valid: True
Trying move: Player(4, 2) to (4, 1), Move valid: True
Trying move: Player(4, 2) to (4, 3), Move valid: True
Generated 4 successors from state: (4, 2, ((4, 3),))
Trying direction: Down, From: (4, 2) to (4, 3)
```

Generates all the successors and shows which successor can become a state. This is indicated by the 'Move valid' determined from a Boolean value.

- Does the correct state become dequeued from the queue?
  - Yes, once the optimal state is found, the state becomes dequeued and is shown as the optimal path.

Overall, the ad-hoc testing approach for the Sokoban project, has proven effective in revealing and resolving a range of problems and bugs, from basic functionality to complex AI-driven interactions. This flexible and dynamic approach allowed for immediate and situational testing without the constraints of predefined test cases, giving accurate and efficient results, allowing for a deeper understanding of the game's mechanics and the AI's performance.

The ad-hoc testing approach, while less structured than methods like TDD, was well suited to the exploratory nature of this puzzle-solving development project. It allowed for flexibility in fixing unexpected scenarios and adapting the testing focus based on immediate results and provable observations. This approach ensured that all aspects of the game, the integration and functionality of the AI, were thoroughly optimized for the best user experience and to achieve the project's deliverables.

However, despite the given results, finding a way to incorporate a structured testing approach such as unit testing or Test-Driven Development (TDD), could complement the ad-hoc strategies or be a better way of testing for my project. This would be particularly beneficial for ensuring the robustness of new features and functionalities as the project evolves, providing a more systematic testing framework that could lead to even higher reliability and performance. It is performed through purposeful test failures to represent desired behaviours of my code and to refactor purposely after producing the solution. Although, described in my interim report, I opted out of TDD or unit testing for the earlier stages was because of its impact in GUI creation and visualisation process seemed unnecessary to perform in. Due to its visibility to notice whether there was a bug or issue, it felt that TDD or unit testing would be complex, instead would be simpler to just fix the issue by hand through predefined testing and debugging rather than predefined test cases.

To summarise, the testing strategies implemented in the Sokoban project effectively ensured that the game met its project's deliverables and provided an enhancement in engaging and challenging experience for users. The AI components were tested to demonstrate not only functionality but also efficiency and effectiveness, showcasing the project's success in creating a dynamic and intelligent game environment.

Chapter 7: Review & Reflection

In this section, it describes progression throughout the project by comparing my timeline. Furthermore, giving an insight into my overall thoughts on the progression made in this project. Including the positives and negatives whilst challenging this project, reflecting on the successful choices made and evaluating on the lessons learnt from this project to improve upon my future software development projects.

Chapter 7.1: All Stages of Timelines:

Here is the timeline from my final year project plan:

Term One:

- Week 1-2 Implementing the Concept
    - Develop the main objectives for my game.
    - Develop a simple GUI for the game Sokoban to run on.
    - Create a stage layout of a puzzle.
- Week 3-4 Installing simple Algorithms
    - Create a playable puzzle for a user to test.
    - Start reading on the usage of Constraint Programming and apply to my code.
- Week 5-7 Further development to AI
    - Start developing the AI solver for Sokoban.
    - Apply techniques such as backtracking, branch and bound etc.
    - Solve various constraint problems using AI.
    - Make a few more puzzle layouts.
- Week 8-9 Interim Report and further evaluation of complexity
    - Evaluate the complexity of the current AI on the made puzzles.
    - Prepare for presentation and make tune ups to the report.

Term Two:

- Week 1-2 Testing
    - Perform testing of the current Sokoban solver and the game
    - Address any bugs in the solver or GUI.
    - Start seeking if it achieves the objectives.
- Week 3-5 Extension features and Final Deliverables
    - Create a different type of Sokoban puzzle that is more advanced to solve.
    - Possible display to give the users to hints when they are stuck on the puzzle.
    - Implementation of a splash screen.
    - GUI should be able to operate Sokoban puzzles and playable for the user.
- Week 6-8 Final Testing
    - Search for any final bugs or issue.
    - Evaluate if the game represents its final objectives, has solved constraint problems.
- Week 9-11 Final Report and Evaluation of Project
    - Prepare the final report and presentation.

Here was my timeline from my interim report:

Christmas Holiday:

- Further implementation of AI techniques
    - Continue developing the AI solver for Sokoban.
    - Apply techniques such as backtracking, branch and bound and A* Search etc.
    - Look up other possible techniques like pruning and understand it's purpose.
    - Solve various constraint problems using AI.
    - Make a few more puzzle layouts in a json file.

Term Two:

- Week 1-2 Testing
    - Perform testing of the current Sokoban solver and the game
    - Address any bugs in the solver or GUI.
    - Start seeking if it achieves the objectives.
- Week 3-5 Extension features and Final Deliverables
    - Create a different type of Sokoban puzzle that is more advanced to solve.
    - Possible display to give the users to hints when they are stuck on the puzzle.
    - Implementation of a splash screen.
    - GUI should be able to operate Sokoban puzzles and playable for the user.
- Week 6-8 Final Testing
    - Search for any final bugs or issue.
    - Evaluate if the game represents its final objectives, has solved constraint problems.
- Week 9-11 Final Report and Evaluation of Project
    - Prepare the final report and presentation.

This was a reflected timeline of my progression in this project:

Term One:

- Week 1-5 Implementing the basic Concept
    - Develop the main objectives for my game.
    - Research that concerns my project.
    - Develop a simple GUI for the game Sokoban to run on.
    - Create a stage layout of a puzzle.
- Week 5-7 Installing simple Algorithms
    - Create a playable puzzle for a user to test.
    - Start reading on the usage of Constraint Programming and apply to my code.
- Week 8-9 Interim Report and start development of AI
    - Begin developing AI techniques whilst researching the most ideal method.
    - Prepare for presentation and make tune ups to the report.

Christmas Holiday:

- Further implementation of AI techniques
    - Continue developing the AI solver for Sokoban.
    - Continue developing the visuals and logic of Sokoban.
    - Apply techniques such as backtracking, branch and bound and A* Search etc.
    - Look up other possible techniques like pruning and understand it's purpose.
    - Solve various constraint problems using AI.
    - Effectively researching on other AI's that replicate this project.

Term Two:

- Throughout Term Two Testing
    - Perform testing of the current Sokoban solver and the game
    - Address any bugs in the solver or GUI.
    - Start seeking if it achieves the objectives.
- Week 1-4 Slowly Implement A* Search
    - Research the usage of A* Search.
    - Explore the usage of heuristic costs in algorithms.
    - Start implementing A* Search.
- Week 5-8 Final Testing and possible enhancements to the project
    - Search for any final bugs or issue.
    - Evaluate if the game represents its final objectives, has solved constraint problems.
    - Continue implementing and improving the A* Search.
    - Research the usage of states.
    - Quality of life changes to increase usability of the code.
- Week 9-11 Final Report and fully install the AI solver
    - Prepare the final report.
    - Fully install the AI solver in the game.
    - Write documentation for every method using Docstring

Chapter 7.2: Reflection on Timeline

Given my final year project timeline throughout its different stages and constant alterations to the original, there was a positive amount of progression but limitations to success. In my plan, it was stated that time management was going to be a risk for my completion of work. Previously, I accounted this by stating to understand each task by decompose into sub tasks for better efficiency. However, I didn't account for the difficulty of the task, and each sub task's time consumption. Furthermore, external assignments and other circumstances had affected my impact on the project overall.

When combating this issue, I would structure my workload throughout the week to have time to work on the project's subtasks throughout term two. This would be updated on my diary, understanding when I needed to make a progression within my code, to meet the timeline requirements. Despite plenty of effort to keep consistent, there were changes to the timeline that had to be left out of my project such as removing the extension deliverables, with further testing capabilities like TDD or unit testing.

Despite the issues of not keeping full track of my timeline, the timeline I persevered with, helped completing majority of the aims and objectives I set out for the project. The timeline structure made it flexible to adapt to many situations that I could have experienced, making it simpler to follow and control on my own.

Chapter 7.3: Full Reflection

Earlier elaborating from my timeline reflection, there were gaps in my planning that were not initially considered when progressing my overall progression. To clarify, I will delve into the objectives of the project, revealing areas where additional attention and refinement were necessary for me to achieve this project's final deliverables and creating a successful AI puzzle solver.

In retrospect, I acknowledge that my initial approach to the project may have been optimistic, overlooking specific difficulties that have presented itself. The details of integrating AI techniques, game development, and various search algorithms within the Sokoban puzzle context present challenges that demanded a structured and refined strategy to challenge such feats. This became very time consuming, despite structuring my day to challenge these complex problems that concerned the code and required through research:

One aspect that required attention is the integration of the chosen programming language, Python, and its Pygame interface. While Python is a versatile language and Pygame is a familiar tool for game development, I recognized the need for a more detailed plan to synchronise these tools effectively for improvement on my project. This included addressing compatibility issues, refining the scope of AI implementation within the game structure, and ensuring a synergy between the user's gaming experience and with the AI algorithms.

During the term, I combatted this issue through research on other projects and specific algorithms that suit the usage of AI in python and in pygame. I made notes and expanded on code snippets or texts that efficiently use AI in my programming language, to gain an understanding on their functions, and how it can be represented in my code. AI manipulation is common when creating video games, thus plenty of projects contained extensive usage of AI techniques to better understand how they are could be implemented in my code. Moving ahead, when anticipating projects that involve specific requirements and various programming languages, I must conduct further research into these requirements before tackling challenging algorithms, to ensure a smooth software development experience.

Furthermore, the oversight in the initial planning is highlighted by the realization that the project lacks a detailed exploration of potential difficulties and challenges associated with the chosen AI methodologies, leading to misunderstandings especially in its the implementation earlier in the term. I mitigated this through productive critical analysis of the Constraint Programming techniques limitations in backtracking, branch and bound, and further analysis in other strategies such as the heuristic search algorithms like A* search and heapq within my Sokoban solver. This provided a clearer understanding of the application and efficiency of these methods, enabling informed adjustments and adaptations to the project's approach. Similar to the previous setback, further research into understanding concepts, enhances the smoothness of creating an AI solver. This could lead to clearer knowledge of tasks, better composition of tasks thus, better understanding of the task and the project overall.

Despite the setbacks to my project, I felt that there was suitable progression to the project's final deliverables and what I gathered from this project experience:

Throughout the project, I was fairly adaptable to the tasks given at hand. When dealing with a requested subtask, I would analyse how I would solve this problem through concurrent thinking and understanding logic to conquer the problem. This would help me better understand the hierarchy task and work myself towards to solving the bigger task. Whilst logic and concepts may be wrong or cause disrupt, I would adapt by creating print statements or extensive debugging of the task at hand to solve the problem and help reach the project's aims and objectives.

Time management was also a key factor to me completing majority of the project's final deliverables. Counteracting my limited start back in the interim, I felt that I better structured my timetable to better understand my project, by spending efficient and valuable time on each task or sub-task encountered to me.

To conclude, this reflection underlines the need for an iterative and adaptive planning process. Referring to my interim, I acknowledged there were gaps in my initial strategy that was not seen as a setback but an opportunity for refinement and improvement for my project and towards successful progression. The experiences this term, allowed me to be more proactive in my project, adaptable and consistent in finding solutions to problems. Moving forward, I will address these efficiently issues, incorporating a more comprehensive understanding of the tasks ahead and refining my project plans to ensure a successful implementation of software within future projects.

Chapter 8: Bibliography

1) Chesani, F., Mello, P. and Milano, M., 2017. Solving mathematical puzzles: a challenging competition for AI. *AI Magazine*, *38*(3), pp.83-96.
2) Zhuang, Y.T., Wu, F., Chen, C. and Pan, Y.H., 2017. Challenges and opportunities: from big data to knowledge in AI 2.0. *Frontiers of Information Technology & Electronic Engineering*, *18*, pp.3-14.
3) Feng, D., Gomes, C.P. and Selman, B., 2020. A novel automated curriculum strategy to solve hard sokoban planning instances. *Advances in Neural Information Processing Systems*, *33*, pp.3141-3152.
4) Kumar, V., 1992. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, *13*(1), pp.32-32.
5) Math Is Fun. Sokoban. Math Is Fun. https://www.mathsisfun.com/games/sokoban.html
6) Rendl, A., Miguel, I., Gent, I.P. and Gregory, P., 2009, January. Common Subexpressions in Constraint Models of Planning Problems. In *SARA*.
7) Rossi, F., Van Beek, P. and Walsh, T., 2008. Constraint programming. *Foundations of Artificial Intelligence*, *3*, pp.181-211.
8) Pereira, A., Ritt, M. and Buriol, L., 2013. Finding optimal solutions to Sokoban using instance dependent pattern databases. In *Proceedings of the International Symposium on Combinatorial Search* (Vol. 4, No. 1, pp. 141-148).
9) Van Rossum, G., 2007, June. Python Programming Language. In *USENIX annual technical conference* (Vol. 41, No. 1, pp. 1-36).
10) Srinath, K.R., 2017. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), pp.354-357.
11) Agruss, C. and Johnson, B., 2000. Ad hoc software testing. *Viitattu*, *4*, p.2009.
12) Hart, P.E., Nilsson, N.J. and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), pp.100-107.
13) Pearl, J., 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc..
14) Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), pp.484-489.
15) Simonis, H., 2005, October. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems* (Vol. 12, pp. 13-27). Citeseer.
16) AlmaBetter. (n.d.). A* Algorithm in AI (Artificial Intelligence). [online] Available at: https://www.almabetter.com/bytes/tutorials/artificial-intelligence/a-star-algorithm-in-ai [Accessed 12 Apr. 2024].
17) Sharma, S.K. and Kumar, S., 2016. Comparative analysis of Manhattan and Euclidean distance metrics using A* algorithm. *J. Res. Eng. Appl. Sci*, *1*(4), pp.196-198.
18) Moon, N.W., Baker, P.M. and Goughnour, K., 2019. Designing wearable technologies for users with disabilities: Accessibility, usability, and connectivity factors. *Journal of Rehabilitation and Assistive Technologies Engineering*, *6*, p.2055668319862137.
19) AKTAY, S., 2022. The usability of images generated by artificial intelligence (AI) in education. *International technology and education journal*, *6*(2), pp.51-62.

Chapter 9: Appendix

APPENDIX A

DIARY:

```
# 27th September 2023
First look at my final year project in terms of how I will approach my
project.
The two things I want to establish before my meeting with my supervisor on
Friday.

* What are the constraints and how effective are they for solving puzzles.
* What programming language will I use with its documentation.

Tomorrow I also want to start on my plan, but I will further investigate my
plan after speaking to my supervisor.

# 28th September 2023
Looking back on my objectives yesterday I have made decision on what
programming language I would like to implement.

I looked to use Python since it has a specific framework for making games
called pygame

## Pygame
Is a set of modules designed to write video games that include video graphics
and sound libraries.

With the use of pygame, I will be able to make Sokoban a lot easier and insert
constraint programming techniques with.

Today I want to research on constraint programming and how well it will work
with python.

# 29th September 2023
I have met with my supervisor today to discuss some of the techinques and the
game I want to do.

My supervisor mentioned making Sokoban the game choice for my puzzle.

## Sokoban
Sokoban originated from Japan and it's a game revolves around pushing boxes to
different areas of the level to solve the puzzle.
```

```
We also discussed some of the techniques I could implement such as
Backtracking which I will research more into when I do my project plan.

# Week 6
This week I will like to design a playable level for the user to solve.

To do this I will need to:

* Customise and design the sprites used for this project
* Draw the grid out for the level
* Handle user input

It may be unrealistic to get all 3 of these tasks done this week but, I would
like to get at least two of these tasks completed this week so that I can
focus my attention on implementing CSP techniques for that level.

# Week 7
Due to troubles with assignments and family concerns, the previous week has
been difficult to fit in final year project work.

However, now that assignments have been cleared, I can now focus on week 6 and
start on AI.

I will be doing some of my project during the holiday due to my setbacks in
the past two months so progression will be recorded as holiday progression as
the subtitle.

# Week 8
Focus on this week is completing the report and presenting my work thus far.

# Christmas Break
Focused on research and got more information on how to implement AI
algorithms.

The code works now so will be looking into improving it with sprites and new
levels.

# Week 9-11
Implement AI algorithms for the basic code.

Improve aesthetics of the game, add the sprites to make it look sophisticated.

# Week 10
Complete bug fixes for the simple level and doing more research into AI
algorithms in this code.

# Week 11
```

```
Do more levels, complete the interface of the game. Allow great accessibility
for the user to access the game.
To achieve this, I will be looking into implementing a hash map, to allow
levels to transverse between eachother on each level completion.

# Week 12
Fix the bug of my code where the game does not end when told to. Implement the
AI aspect of everything this week with thought of knowledge from planning.

# Week 13
Implement the A* Search given from the testing one made externally to this
project, figure out the logic of moving the box to a target. Start final
report.

Complete the heuristics.

# Week 14
Complete documentation, final debug changes and complete final report.
```

APPENDIX B

YOUTUBE VIDEO OF PROJECT: https://youtu.be/UkNeuGlUXLk

APPENDIX C: Professional Issues

INTRODUCTION

Usability in technology focuses on the simple accessibility of which people can use a system to interact with a system or application. In the context of the Sokoban game project, usability is a multifaceted concern that directly influences the effectiveness of the game and its underlying AI systems. This report section explores the professional issues, one of which is usability, and its practical or ethical impact on my Sokoban project, focusing on accessibility, the implications of AI potentially replacing human decision-making causing large ethical reasoning of AI in video games.

ACCESSIBILTY

Accessibility refers to the simple providence of products, devices, or services. The practical goal of accessibility in Sokoban is to ensure that the game is playable enjoyable and transferrable for everyone, including individuals with disabilities such as visual impairments, limited motor skills, or cognitive challenges [18]. This could further expand the extensions of my project and allow the code to be further accessible to impaired or disabled players.

Visual Accessibility: Sokoban's interface utilizes distinct colours to differentiate between walls, boxes, and targets. For players with colour blindness, this could pose a challenge. Implementing symbols or patterns on these elements can enhance visual accessibility. Additionally, allowing users to customize colours or providing high contrast modes ensures that all players can distinguish game elements clearly.

Motor Accessibility: Players with limited motor skills, like typing or sitting might find precise movements or tracking challenging. To address this, the project could allow for various input methods, this includes keyboard customization, single-switch input, or even eye-tracking technologies, enabling players to use the most comfortable and effective control handlers.

Cognitive Accessibility: The challenging nature of Sokoban requires extensive planning and problem-solving, which might be a limitation for players with cognitive disabilities. Therefore, providing adjustable difficulty levels or if users are reluctant for AI to automatically solve the puzzle, hints could make the game more accessible, ensuring that it remains challenging but not discouraging.

REPLACING HUMANS WITH AI

The integration of AI in Sokoban could raise ethical questions about the role of AI in gaming and whether AI could or should replace human players. In Sokoban, AI is used to solve levels and demonstrate the optimal paths by animation, which disregards the strategic aspect of puzzles but also poses the risk of diminishing the gaming experience.

The AI solver in Sokoban can enhance the user experience by providing hints or solve challenging levels. Using AI to demonstrate strategies or solve puzzles can be educational, helping players learn new techniques and improve their problem-solving skills. This educational aspect underlines the role of AI as a complementary tool that supports human players or new players [19]. However, if players rely too heavily on AI, it might reduce the satisfaction derived from solving the puzzles independently. Balancing AI involvement ensures that the is utilised to assist and teach rather than replacing the human player.

CONCLUSION

The usability of Sokoban involves ensuring that the game is accessible to all players and balancing the role of AI in enhancing versus replacing human interaction. By focusing on these aspects, the Sokoban project not only enhances player experience but also obliges to its ethical standards that respects player boundaries and data privacy. This approach ensures that the game remains a tool for enjoyment and learning, rather a reason for ethical and social conflict.