University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Investigation and study on reinforcement learning for optimizing well path

Master's Thesis in Computer Science
by

Samiul Ehsan Chowdhury

Supervisors

Dr. Dan Sui

Dr. Tomasz Wiktorski

Andrzej Tadeusz Tunkiel

July 14, 2021

"Never discourage anyone... who continually makes progress, no matter how slow"

Plato

Abstract

Designing an optimal path has been considered one of the key challenges for drilling engineers. Even for a group of competent engineers, it takes many months to plan a well. A robust optimized path can influence total cost of wellbore, transport efficiency, and drilling speed. A proper optimized path can be called, if it is the shortest path, avoids collision, and has maximum contact with the reservoir.

The aim of this paper is to investigate the efficiency of machine learning algorithm to design an optimal path. In order to draw an optimal path, this thesis will apply QLearning of Reinforcement Learning in python and Path Tracing engine in Unity3D. The agent in both programs will interact with the environment, achieving maximum reward upon reaching goal or penalties with passing through the obstacles which are given. Numerous studies have been conducted on this subject recently. This thesis will show the behaviour of the algorithms to obey the main criterion of the trajectory design as an alternative solution.

Acknowledgements

I would like to thank my family and friends who have been supporting me and give courage to tackle all the problems.

I am very grateful to my supervisors, Professor Dan Sui, Tomasz Wiktorski and Andrzej Tadeusz Tunkiel for providing me this wonderful opportunity to pursue this thesis under them. Their guidance, ideas, encouragement and insightful and valuable feedback amidst their busy schedule has been invaluable during the thesis.

Contents

Abbreviations

ML	**M**achine **L**learning
AI	**A**rtificial **I**ntelligence
RL	**R**einforcement **L**learning
QL	**Q** **L**learning
TSP	**T**ravelling **S**alesmen **P**roblem
ACO	**A**nt **C**olony **O**ptimization
DE	**D**rilling **E**ngeering
WP	**W**ell **P**lanning
GA	**G**eneric **A**lgorithm
KOP	**K**ick **O**ff **P**oint
DLS	**D**og **L**eg **S**everity
NPV	**N**et **P**resent **V**alue
NCW	**N**on **C**onventional **W**ell
ANN	**A**rtificial **N**eural **N**etwork
HGA	**H**ybrid **G**eneric **A**lgorithm

Chapter 1

Introduction

Directional Drilling is an essential aspect of drilling Engineering and well planning. It is a practice of directing and deviating a wellbore towards a predetermined underground location. The whole drilling process may vary on engineering principles, corporate or personal philosophies; the result should be a safely drilled, minimum-cost hole that satisfies oil/gas production requirements.[1]

1.1 Motivation

Well planning is a tedious and time-consuming job for a drilling engineer. During the recent oil price slump, drilling companies have focused on costs reductions associated with drilling and well planning. In response, a concept of smart well planning complete with automation has developed. typical time spent for designing and planning a well in Norway is 2-3 months and predicts that it could be reduced by 80 percent if more automation and machine learning is involved in the process.[2]

1.2 Problem Definition

Due to so many complex and interacted drilling variables and design constraints, well bore trajectory optimal design is very demanding. In drilling engineering, well path optimization plays an important role, that could be based on the minimization of the total length of wellbore, minimization of drilling cost, improvement of cuttings transport efficiency, and increase of drilling speed [3]. When different optimization algorithms (genetic algorithm, particle swarm optimization etc.) are applied, they give some probable solutions.However, in the recent years, due to high density of drilled wellbores in the

mature fields, anti-collision has become a major concern. Considering this complex optimization problem, our attempt is to introduce AI techniques and algorithms for optimization. Reinforcement learning is able to solve a wide range of complex decision-making tasks that are previously unsolvable/challenging problems [4]. For this study, we aim to investigate reinforcement learning algorithm for optimizing 2D and 3D well path to see if the solution will be acceptable or reasonable or it performs better or worse than other existing optimization algorithms (genetic algorithm, particle swarm algorithm, etc.)

1.3 Research Questions

Following are the research questions for the undertaken research on Investigation and study on reinforcement learning for optimizing well path:-

1. Is it beneficial to use Machine learning for well path optimization?
2. Can Reinforcement Learning perform in path optimization?
3. Can Unity perform in path optimization?
4. Can RL or unity perform better or less than other existing optimization algorithms (genetic algorithm, particle swarm algorithm, etc.)?

1.4 Usecases

The related cases from the relevant and highly credible research on similar topics to well path optimization using reinforcement learning would be critically analyzed. So the results of the research can meet the compliance of given research questions in the best possible way.

1.5 Challenges

There are certain challenges that had been learned from the background research on reinforcement learning for well path optimization. It is a difficult task to manage a right or suitable framework for reinforcement learning. High reliance on reinforcement learning had led to overload of states that had adversely affected the results of desired objective. It is not recommended for a solution of simple problems. Reinforcement learning had been reckoned as a data hungry technique [4]. It needed a lot of data for computation

for taking decisions and it is a challenge to obtain huge amount of accurate and precise data. Integration of reinforcement learning with artificial intelligence for solving multiple tasks is a challenge. It is also challenging for reinforcement learning to perceive the environment in its true picture. Minimization of trial and error based learning is also a challenge [5].

1.6 Outline

Chapter 2 - Background: The first section in this chapter describes relevant theory behind the methods implemented in this thesis. The second section, I discuss some existing approaches to provide some context to this thesis.

Chapter 3 - Solution Approach: This chapter discusses related works and gives an overall idea of what is implemented in this thesis.

Chapter 4 - Implementation: This chapter shows what has been done in this thesis. It starts by discussing the architecture of the models and how they were trained and tested in 2D and 3D.

Chapter 5 - Results: The results after the implementation in Chapter 4 are shown in this chapter.

Chapter 6 - Discussion and Conclusion: This chapter discusses the results shown in Chapter 5.

Chapter 2

Background

As the name suggests, this chapter aims to provide a background to the work. It starts with short summaries of technical and theoretical information that is relevant to this thesis.

Due to the use of a Machine Learning Model, terms such as Machine Learning, Reinforcement Learning, Drilling Technique, and trajectory optimization are discussed, as well as more advanced terms such as Q Learning, Gridworld, and Unity3D.

The second section discusses some of the currently available approaches. A brief summary of some of the many wonderful pieces of work completed prior to the presentation of this thesis. Additionally, several popular commercial software applications are discussed.

2.1 Technical and Theoretical Background

As mentioned above, this section aims to provide a short summary of the technical and/or theoretical knowledge that is required to understand the work that has been done in this thesis and also by other similar and related works.

2.1.1 Machine Learning

"Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed" [6]. Experience can be some kind of input data (text, excel tables, images, sound clips, videos, etc). With Machine Learning a computer can find patterns in the data to make some kind of predictions that would normally be very complex for a human being [7].

2.1.2 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning[8].

Reinforcement learning differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge)[8].

2.1.3 Q Learning

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy is not needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward[9].

The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

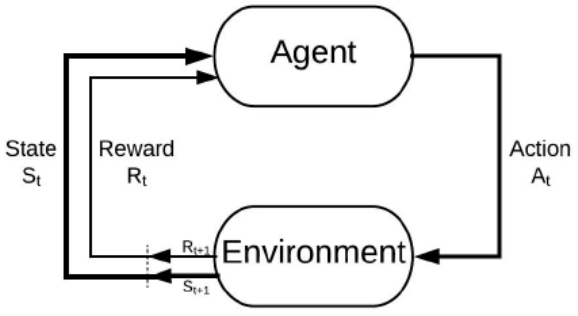

Figure 2.1: Basic components of Q Learning

The environment responds to the agent's actions by generating new observations, O(t), and scalar reward signals, R. (t). The subsequent action is determined by history, which is defined as the sequence of observations, actions, and reward signals at time t.

States are the information used to determine what happens next. There are three main types of states- Environment state (S), agent state (S), and the information/Markov state (S). The reward function r is summated over t (the time steps) which means the objective function calculates all potential rewards that can be attained through the game.

$$\sum_{t=0}^{t=\infty} \gamma^t r(x(t), a(t))$$

Figure 2.2: Cost/reward function of Q Learning

x represents the state at any given time step (denoted by t), and r represents the reward function for x and a.[10]

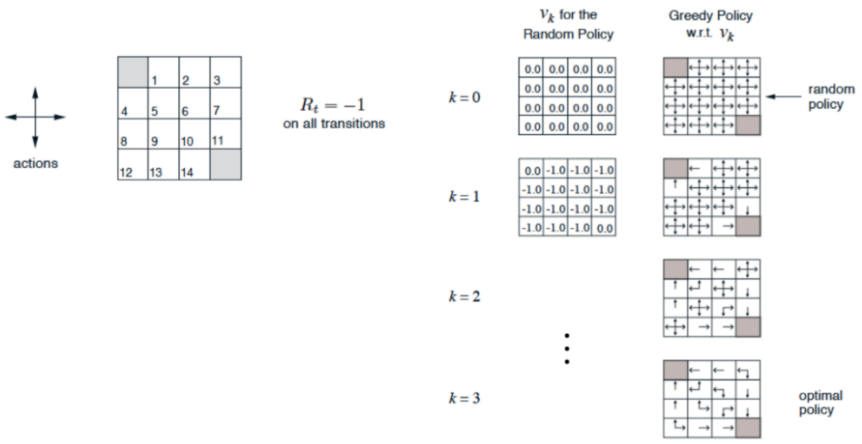

Figure 2.3: policy iteration of Q Learning

optimal policy-iteration re-defines policies after each step and computes the value in accordance to that new policy until the converged value is reached. This method will always reach convergence at the optimal policy and generally requires less iterations.

Implementation of Q-learning includes quantization and approximation of function. Q-learning at its least complex stores information in tables. This methodology flounders with the expanding quantities of action and states, the probability of the representative staying a specific state and playing out a specific activity is progressively little. [11]

Q-learning can be joined with work approximation, this makes it conceivable to apply the calculation to bigger issues, and in any event, when the state space is constant. [12]

learning rate, initial conditions, and discount factor are the influences of variables in Q-learning. In completely deterministic conditions, a learning pace is ideal. At the point when the issue is found to be stochastic, it is then known that the algorithm combines

under some any sort of technical situations on the LR (learning rate) that expect it to diminish to nothing.[11]

The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a).

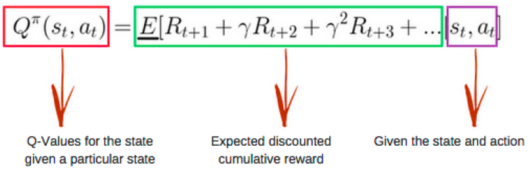

Figure 2.4: Bellman Equation in Q Learning[13]

2.1.4 Grid World

Grid World, a two-dimensional plane , is one of the easiest and simplest environments to test reinforcement learning algorithm. In this environment, agents can only move up, down, left, right in the grid, and there are traps in some tiles.The agent starts at the fixed start position and when it arrives at the goal or trap, episode ends.[14]

2.1.5 Unity3D

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005.The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms.The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences.[15]

NavMesh

Unity NavMesh is a package used to perform spatial queries such as path finding and walk-ability tests. This class also lets user set the path finding cost for specific area types, and tweak the global behavior of path finding and avoidance.[16]

2.1.6 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is the challenge of finding the shortest yet most efficient route for a person from a list of specific destinations. It is a well-known algorithmic problem in the fields of computer science and operations research.

There are many different routes to choose from, but finding the best one—the one that will require the least distance or cost—is what mathematicians and computer scientists have spent decades trying to solve.

TSP has commanded so much attention because it is easy to describe yet so challenging to solve. The problem can be solved by analyzing every round-trip route to determine the shortest one. However, as the number of destinations increases, the corresponding number of round-trips surpasses the capabilities of even the fastest computers. With 10 destinations, there can be more than 300,000 round-trip permutations and combinations. [17]

2.1.7 Ant colony optimization

The ant colony algorithm is an algorithm for finding an optimal path that is based on the behavior of ants searching for food.

At first, the ants wander randomly. When an ant finds a source of food, it walks back to the colony, leaving "markers" (pheromones) that show the path has food. When other ants come across the markers, they are likely to follow the path with a certain probability. If they do, they then populate the path with their own markers as they bring the food back. As more ants find the path, it gets stronger until there are a couple of streams of ants traveling to various food sources near the colony.

Because the ants drop pheromones every time they bring food, shorter paths are more likely to be stronger, hence optimizing the "solution." In the meantime, some ants are still randomly scouting for closer food sources. A similar approach can be used to find a near-optimal solution to the traveling salesman problem.

Once the food source is depleted, the route is no longer populated with pheromones and slowly decays. Because the ant colony works on a very dynamic system, the ant colony algorithm works very well in graphs with changing typologies. Examples of such systems include computer networks and artificial intelligence simulations of workers. [18]

2.1.8 Drilling Engineering

Directional drilling is an essential part of modern drilling operations. It can be defined as the practice of drilling non-vertical wells, or the practice of controlling the direction and deviation of the well to a predetermined underground target.

Types of Well Trajectories

Wells can be drilled using a multitude of different well profiles. The most common well trajectories are vertical, J-shape, S-shape and horizontal.

Vertical wells are simply vertical wells with no inclination, in reality minimal changes in inclination do occur in vertical wells.

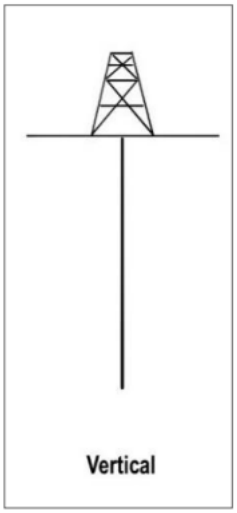

Figure 2.5: Bi-dimensional well trajectory of a vertical well

J-type wells are widely used and characterized by a straight vertical section, a kick-off point KOP with a build section, and a hold section.

S-shape wells are more complicated than J-shaped wells. The first section is a vertical section, followed by a kick-off point and a build section. When the desired angle is reached, the angle is held until the desired target is reached. The last part of the S-shaped well consists of a drop section. Where inclination of the well is dropped until the well is vertical. Then vertical drilling can proceed. S-shaped wells can be used for avoiding salt domes, hitting multiple targets, or to avoid faults.

Horizontal wells consist of a vertical section with a KOP. At the KOP, the well starts the build section until the well turns horizontal. At 90 $\circ$ there is no drop section, but minor variations may occur in the inclination. Horizontal wells are used for production in thin lateral reservoirs, avoiding gas and water coning and increasing production in a reservoir with low permeability. [19]

The 3D trajectory is usually planned after the preliminary design of the 2D well. The 3D project is essential due to the well not being drilling in a vertical plane consisting of only the wellhead and the target. Many factors should be considered, such as the geology and the BHA composition, making the drill turn and drill in different planes (Rocha, 2011).

Figure 2.6: Bidimensional well trajectory of a J-shaped well. Source: (Sunny, 2015b)

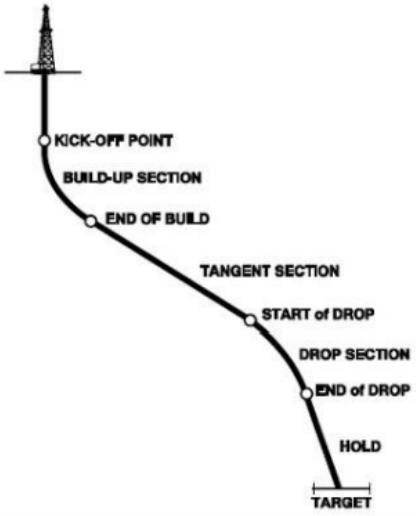

Figure 2.7: Bidimensional well trajectory of an S-shaped well. Source: (Sunny, 2015b).

Further, these 3D wells can also be applied in situations such as when it is impossible to place the wellhead in the same vertical plane as the target, in the case of restricted rig positioning, or when there is a necessity of reaching multiple targets, in a different plane, with the same well. The 3D trajectory is presented in a X, Y, Z plane.

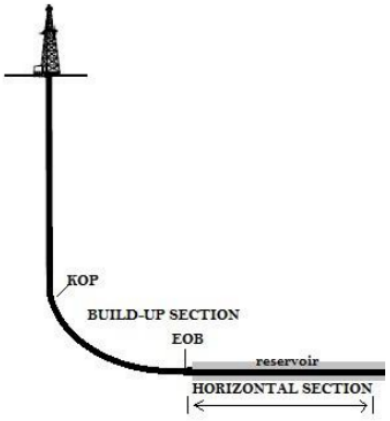

Figure 3.7: Bidimensional well trajectory of a horizontal well.
Source: (Sunny, 2015b).

Figure 2.8: Bidimensional well trajectory of a horizontal well. Source: (Sunny, 2015b)

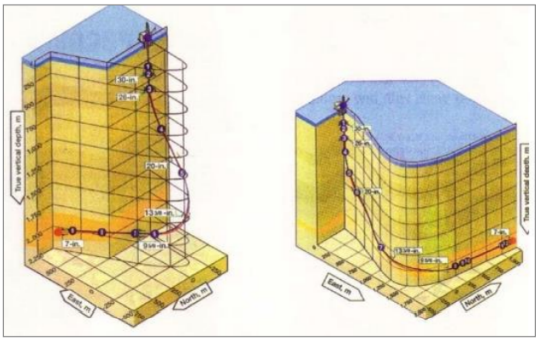

Figure 2.9: 3D Well Trajectories. Source: Eid E., 2020.

2.1.9 Well Planning

Well planning consists of several modules, the most basic of which are presented in Figure 1. Traditionally modules are completed in a predefined order and concerning the well purpose and constraints related to a geological area or/and technical limitations as well as the total budget.

However, The objective of well planning is to formulate from many variables a program for drilling a well that has 3 distinct characteristics: Safety, Cost and usability.

Unfortunately, it is not always possible to accomplish these objectives on each well because of constraints based on Geology,Drilling equipment,Temperature,Casing limitations,Hole sizing,Budget.

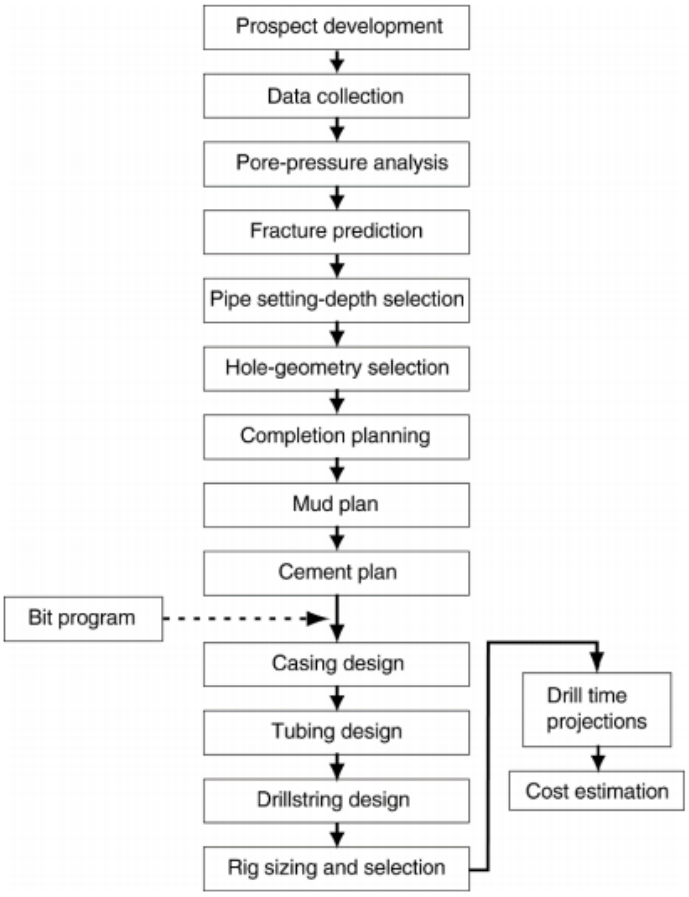

Figure 2.10: Traditional well planning procedure, (Adams, 1985)

2.2 Existing Approach

This section discusses existing approaches.It starts by discussing some existing methods that are used to well path optimization and then some existing commercial solutions.

Well number, location and trajectory design is one of the crucial aspects of Field Development Program (FDP).Well trajectory design is an arduous task to estimate the best ways to drill and perforate the desired reservoir layers to achieve pre-defined company objective(s). Besides, wells are an expensive part of the development phase and thus, they must be carefully studied and planned. Hence, designing a well trajectory demands an interdisciplinary team effort calling for inputs from geologists, geophysicists, reservoir engineers and drilling engineers. Many parameters affect the well trajectory placement including but not limited to field location (offshore/onshore), location of the rig, cost effectiveness, total drilling length, target zones, dogleg severity, step-out distance, basic well design, and the objectives of maximizing Recovery Factor (RF), economic values and/or production.

2.2.1 Gradient-based Well Trajectory Optimization

A method to automate the process of optimizing the trajectory of production well was first proposed by Vlemmix et al.[20]. Motivated by the advancements in the adjoint-based well location optimization, the authors extended the method of Handels et al. [21] to determine an optimal well trajectory in a three-dimensional reservoir model. Since optimal well trajectory is crucial to avoid gas cusping and water coning, the authors verified the method on a thin oil rim reservoir with a relatively large gas cap and aquifer.The Vlemmix et al. [20] method is based on surrounding each trajectory point with 'pseudoside tracks'. These fictional 'pseudo-side tracks' have very small perforations and thus production rates. The reason that this approach was chosen over placing vertical pseudo wells in each grid-block is that the effect of the side tracks on the total well behaviour including lift and well bore friction can be taken into account. The gradients of the objective function with respect to a dimensionless multiplication factor for perforations (interpreted as a 'pseudo valve representation' of an Inflow Control Valve) are used to find the improving directions for the trajectory points.

Two of the main disadvantages of the method were restriction of the trajectory movement to only two directions and restriction to optimize the well length. Additionally, the adjoint-based optimization technique may get stuck in a local optimal solution. The method employs drill-ability/smoothing algorithm to ensure dogleg severity is below the predefined limit. This ensures that the final well trajectory is drill-able and realistic.

This approach confirmed the scope for developing an algorithm to predict the well trajectory optimization and significantly improve the overall efficiency during FDP.

2.2.2 Gradient-free Optimization

Gradient-free optimization technique is another widely used method of optimizing well placements and trajectories. Although it is a computationally demanding methodology and numerous reservoir simulations have to be executed, it can in theory, capture the global solution. Several efficient algorithms have been established so far by means of Genetic Algorithm (GA), Artificial Neural Network (ANN), Simulated Annealing, etc. to compute the optimal well placement and trajectory.

Yeten et al.[22] presented a novel method for the optimization of the type (number of laterals), location and trajectory of non-conventional wells (NCWs). The authors presented a procedure of optimizing NCW using GA. GAs are stochastic search algorithms based on the general principles of Darwinian evolution and hence require numerous simulation runs. So as to moderate the actual number of simulations and improve the

efficiency, GAs are used in conjunction with three routines: ANN, hill climber and a near-well up-scaling technique. While ANN is used as a proxy to the reservoir simulator, hill climbing procedure enhances the search in the immediate neighbourhood of the solution and near-well up-scaling methodology speeds up the finite difference simulation runs. Besides, the authors also successfully attempted to account to the effects of reservoir uncertainty in few cases by including numerous plausible geo-statistical realizations of the reservoir in the calculations of the objective function.

Another method of optimizing well location was presented by Badru et al.[23] to maximize the NPV in an FDP. The paper presented the use of Hybrid Genetic Algorithm (HGA) in conjunction with a reservoir simulator. HGA includes GA, polytope algorithm and a kriging proxy. Although the algorithm optimizes the number and location of wells, the produced results were strongly dependent on process variables (completion, recovery process, production/injection rates, project life, etc.). Various field-scale example reservoirs presented in the paper demonstrate its worth of judging locations as good as engineering judgment.

Emerick et al.[24] proposed a robust algorithm of simultaneously optimizing the number, placement and trajectory of production/injection wells. To deal with well placement constraints, the developed software uses Genocop III – Genetic Algorithm for Numerical Optimization of Constrained Problems. The software is capable of dealing with realistic reservoirs with arbitrary well trajectories, complex model grids and linear and non-linear well placement constraints. Although initializing population randomly or using engineer's proposed base case are two possible ways of assigning initial population for the GA, the paper advocates the idea of starting the optimization process from the initial guess of engineers to achieve the "improved engineer's base case" which, in general, is the best possible solution to the problem. Based on the study of a synthetic case, the authors also concluded that the reservoir quality maps can be used for complex cases (or time-restricted scenarios) to define the initial well locations followed by optimization of type and number of wells.Full-field reservoir models were tested using the developed software to demonstrate the reliability of the algorithm.

Lee et al.[25] presented a paper on designing economically optimized wells by GA. The authors proposed using GA with a node-based configuration to get robust and more realisticwell-designs considering location, trajectories and interwell interference.The developed algorithm is capable of designing both vertical and horizontal wells (with several kick-off points) while improving the objective of the FDP. It was also duly presented that the interwell interference yields a comparatively lesser NPV than a non-interwell interference scenario.

Beckner et al.[26] proposed a cost-effective method to optimize the well schedule and placement plan for the FDPs using simulated annealing algorithm in conjunction with a reservoir simulator and an economic analysis module. Although the paper does not describe an algorithm to optimize the well location, the field-scale example reservoir presented in the paper demonstrated differing responses of well positions and scheduling for areal variations in reservoir properties and/or well costs and complete times. Moreover, a non-uniform well spacing is optimal for development to avoid effects on well productivity from well interference.

2.3 commercial solutions

In order to minimize drilling Challenges and Reduced costs and time associated with completion of wells, many companies have created their software for well design and real-time simulation. Some of this commercial software and its utilities will be presented in this section.

2.3.1 eDrilling

eDrilling AS is a world-leading supplier of AI, machine learning, and predictive analytics solutions to the oil and gas industry. The company has created a Life Cycle Drilling Simulation concept, diagnostics technology merged with a 3D visualization into a 'virtual wellbore' and advanced dynamic drilling models.All the models can interact with one another and be used in the whole drilling value chain from design and planning (wellPlanner) through scenario developments and training (wellSim) and then real-time operations/monitoring (wellAhead, wellBalance) and finally experience transfer and post-analysis.[27]

WellAhead

The wellAhead is a software for automated monitoring, live well support, and real-time optimization. It enables a better understanding of the well's status and the formations in its surroundings.

The software uses all available drilling data combined with modeling and operational plans to monitor the well and provide counseling for more optimal drilling. Its graphic user interface provides real-time information visualization. [28]

wellPlanner

wellPlanner Uses dynamic simulations to address the industry's need to improve safety margins and reduce and rid themselves of risk, as well as the need to drastically reduce well planning time.[27]

Benefits of wellPlanner are increased drilling productivity, improved well planning accuracy,reduced drilling risks and uncertainty,improved drilling safety and quick well planning. [29]

WellSim

The wellSim™ is the software product family for engineering and training of all disciplines by improving insight and understanding of the dynamic well behavior; it has the potential to change the ways of planning and drilling several complex wells. [27] This software utilized an advanced downhole simulator, Intellectus, including a dynamic ROP model and coupled Flow and TD model. [30]

WellBalance

The wellBalance software improves any MPD control system to keep a better constant bottom hole pressure during MPD operations and perform planning with an offline model. It provides with real-time set points to the MPD control system, based on a dynamic real-time simulation calibrated against downhole measurements. The software is complemented with an offline simulation tool, the wellBalance™ Offline, used to test and analyze operations and procedures.[27]

2.3.2 Oliasoft Softwares

Oliasoft WellDesign, an ICE platform that remodels the process of building cost-efficient, secure and competent well design systems.WellDesign's advanced simulation engines automate well planning and operations, improving efficiency and safety; the open API allows users to build custom solutions by working any 3rd party application granting the user free range to design and automate their well planning ecosystems, and providing real-time collaboration between different virtual teams.

The main features from WellDesign are its integrated workflow, which allows all calculations to update according to the latest data, its comprehensive solution that makes sure all required well design calculations included are fully documented, reducing dependencies

on consultants, its digitalization, that allows complete control over the data potential, proving free to use data anywhere, anytime and for any application, and its integration system, allowing seamless integration with third-party applications and connecting to real-time data streams from operations to update designs and monitor critical parameters automatically.

Oliasoft's Well Design Simulator is divided into six modules: Trajectory Design, Casing Design, Hydraulics SS, Torque Drag, Tubing Design, and Blowout Kill. [31]

2.3.3 Schlumberger Softwares

Schlumberger has created a cloud-based environment called DELFI to compliment their large bank of data and different software solutions for the whole of the petroleum chain. It harnesses data, domain expertise, and scientific knowledge changing the way to perform in every part of the EP value chain. The DELFI environment creates workflows and applications attainable to every single user. It provides members with access to build shared workspaces for models, interpretations, and data while respecting proprietary information boundaries.

The DELFI environment puts the full scope of available cognitive technologies to work from AI to analytics. Robust cognitive systems recognize each user to deliver a uniquely personalized experience. Intelligently searching, proactively learning, and automating tasks, enabling the user to predict, prioritize, and advise. The new data is automatically shared between jobs across the DELFI environment; each live project is dynamically optimized with the latest information.

Apart from the DELFI environment, the company also provides software for drilling design,such as the DrillPlan and the DrillBench software.[32]

DrillPlan

The DrillPlan solution is a digital well construction planning solution that maximizes the results from shared teams by giving them access to all the data and science they need in a single, common system. The automation of repetitive tasks and validation workflows, enables better quality drilling programs to be produced quickly, and ensures entire plan is coherent.

DrillPlan solution includes circular workflows, plans are improved as new data is added—future programs learn from the experience of all wells planned before.

Designed for the cloud and accessible in the DELFI cognitive EP environment, the DrillPlan solution provides easy access to all of your well construction projects.[33]

DrillBench

The DrillBench Dynamic Drilling Simulation Software is a Schlumberger software that provides the user with dynamic simulations for pressure control, blow out control, managed and underbalanced operations, and well control.

The company's recent enhancements on this software include the integration of well paths directly from the Petrel platform, being able to read pore and fracture pressure from the Techlog platform, the support for dual gradient drilling with improved modeling of managed pressure drilling, the strengthened dynamic SS calculations, subsea pump, and the rig site Kick, which is a standardized kick sheet for operational rig site use, based on robust well control modeling and simulation. [34]

2.3.4 Halliburton Softwares

Halliburton is another well-known company in the industry, with a large bank of data and different software solutions for the whole of the petroleum chain. The company provides EP professionals with a software-driven life-cycle named Halliburton Landmark Solutions. Landmark Solutions is a hybrid cloud environment with seamless connectivity that uses a digital twin technology to provide the user with a faster, more open, and collaborative open industry platform. Two of the central systems in Landmark Solutions are the WellPlan software and the DecisionSpace® 365. The WellPlan software is the latest evolution in Halliburton's well construction information solutions. It is integrated with EDT and EDM applications, providing a complete well engineering software tool kit capable of designing complex well string operations and navigate different challenges found in the well construction.[35]

The DecisionSpace® 365 is a cloud-based subscription service for EP applications found on OSDU that provides high throughput, low latency, and self-cleaning solutions to large quantities of data from various sources into the OSDU. Once the data is loaded, the DecisionSpace® 365 cloud provides modular, open, and plug-and-play solutions with an intelligent workflow to provide efficiency and insight.[36]

Chapter 3

Solution Approach

This chapter discusses works that are related and gives an overall idea of what is done in this thesis in order to build a background and help understand the work done before this thesis was even possible. Finally, it discusses the approach that has been used in this thesis.

3.1 Trajectory optimization

Mansouri et al. [37] suggest a trajectory optimization model based on the genetic optimization algorithm (GA) with an objective function that maximizes the minimum separation factor along the wellbore. The algorithm seeks for the least distance between the planned well and the reference well in three dimensions. They applied the minimum separation factor constraint of 1.5. GA provides several random solutions, which are constrained in their study by KOP (kick-off point), DLS (dog-leg severity), inclination, and azimuth. The interval for calculations was selected to be 30 m. Busby et al. [38] point out that "manual optimization" of wellbore placement is a formidable task due to many possible solutions, uncertainties handling, and complex wellbore trajectories. Therefore, they propose a methodology for well placement based on the experiences organized as machine learning regression models using simulated data. In their model, geological parameters, including uncertainties and trajectory parameters, were used as inputs. Lu et al. [39] develop a bi-objective optimization technique to increase production based on StoSAG (stochastic simplex approximate gradient). Two primary objectives of the defined method are 1) search for optimal wellbore trajectory (including anti-collision) and control settings of injectors and producers to maximize the production and 2) minimize the risk related to achieving low net present value (NPV). The authors propose an iterative simultaneous procedure, where trajectory and control settings are updated for

each iteration because it outperforms more common sequential optimization as claimed in that paper. Yeten et al. [22] used GA in conjunction with an artificial neural network, a hill-climber, and a near-well up-scaling technique to determine the optimum well location and trajectory for non-conventional (multilateral) wells. Due to its stochastic nature, GA requires multiple iterations.

The artificial fish swarm algorithm is used to optimize the goal function, which is the smallest well length (AFSA) [40]. The calculations were completed using the Matlab environment. Compared to previously published data, AFSA optimization offers the best numerical results and the shortest route while also providing great stability and reliability. The algorithm has a basic structure and fast convergence, resulting in a global optimum in a short amount of time. As a result, AFSA can be utilized to determine the best drilling path. An article has shown that because of the difference between an actual trajectory and a planned trajectory, it is defined as a multi-objective optimization problem (MOP) with parameter uncertainties [41].

3.2 RL in Well Planning

There have been many attempts to incorporate Machine Learning with Drilling Engineering to reduce time and design cost. Reinforcement Learning has been used as the algorithm can learn and perform based on previous experiences on its own. Some of the works related to this thesis are discussed below.

3.2.1 RL for well location optimization

In this dissertation, They put the well location selection problem as a multi-stage sequential decision. Using reinforcement learning techniques and to formulate sequential well location selection as a dynamic programming problem. Within the reinforcement learning framework, geostatistical simulation techniques have been applied to characterize the uncertainty in reservoir models and determine the updates to the beliefs regarding reservoir rock properties resulting from alternative well location decisions and hypothetical observations at those locations. Because of the number of feasible sequences of well locations, the possible observations within each sequence, and the computational demands of simulating updated reservoir properties, traditional dynamic programming solution methods are not trackable. They show the application of reinforcement learning, a class of algorithms that combine Monte Carlo sampling methods with functional approximations of the objective function. These methods provide a computationally tractable approach to

exploring sequential well location selection with explicit consideration of the information value of initial well locations.

Using this approach, they test the hypothesis that a well location selection strategy is more robust to uncertainty in the initial data than single-stage optimization approaches. To test this hypothesis, they apply a novel reinforcement learning framework to select well locations accounting for information value. They have developed several proxy geostatistical models to reduce the computational time and effort and explore the effects of using a proxy model on the suggested well placement strategy. To find the solution to the well location problem, they used Q-learning and discuss its limitations. They build the framework on a more straightforward two-dimensional well location problem, using tabular and artificial neural networks as functional approximators. In the 2D case, they explored the sensitivity of the policy developed using reinforcement learning to the hyperparameters that control the exploration of the state-action space and convergence to the optimal policy.

Moreover, the deep reinforcement learning algorithm is also used for the Stanford V and SPE comparative solutions project model 2 reservoirs, which are more prominent three-dimensional reservoir cases. The results of the more significant reservoir cases demonstrate the benefits of sequential well location selection relative to single-stage optimization approaches. [42]

3.2.2 Training an Automated Directional Drilling Agent with Deep Reinforcement Learning in a Simulated Environment

In this paper, a method of training an automatic agent for motor directional drilling using the deep reinforcement learning approach is proposed.

In designing the method, motor-based directional drilling is framed into the reinforcement learning with an automatic drilling system, also known as an agent, interacting with an environment (i.e., formations, wellbore geometry, equipment) through choices of controls in a sequence. The agent perceives the states such as inclination, MD, TVD at survey points and the planned trajectories from the environment, and then decides the best action of sliding or rotating to achieve the maximum total rewards. The environment is affected by the agent's actions and returns corresponding rewards to the agent. The rewards can be positive (such as drilling to target) or negative (such as offset distance to the planned trajectory, cost of drilling, and action switching).

To train the agent, a drilling simulator in a simulated environment with a layered earth model and BHA directional responses in layers is currently being developed. The

simulator assumes that all other characteristics of the drilling system are constant and handles them automatically. During training, the agent is also provided with the planned trajectory.

The directional-drilling agent is trained for thousands of episodes. As a result, the agent can successfully drill to target in this simulated environment through the decisions of sliding and rotating. The proposed workflow is known as the first automated directional drilling method based on deep reinforcement learning, which makes a sequence of decisions of rotating and sliding actions to follow a planned trajectory.[43]

3.2.3 A Reinforcement Learning Based 3D Guided Drilling Method: Beyond Ground Control

This paper proposes a downhole self-steering guided drilling method based on a reinforcement learning framework to achieve the 3D well trajectory design and control in real-time. In every time interval of the drilling process, the proposed system evaluates the drilling status. It gives the adjustment action of the drill bit in 3D space according to the received data, guiding the drill bit to the target reservoir without the involvement of humans. The main module is a modified deep Q network using the Sarsa algorithm for online self-learning.

The experimental results show that after training, the drill bit is increasingly able to select control actions closer to the target reservoir. The frequency of effective actions is approximately 258 percent higher after the algorithm converges. The proposed system has the ability of online self-learning, which can automatically adjust the evaluation and decision models without manual monitoring. [44]

3.3 Proposed Solution

As refered to 1.2 Trajectory optimization is challenging task in drilling engineering. A number of Common methods such as Steepest ascent algorithm; conjugate gradient method; LBFGS method; Levenberg-Marquardt algorithm; Gauss-Newton method;SPSA; EnOpt; EnKF; SID-PSM; NEWUOA; QIM-AG. have been applied to sort out the trajectory optimaization as well as the different machine learning algorithms has been pit inot action as well, to present the more clear and comprehensive picture of path-finding. this paper has been used RL for number of reasons, While many commonplace ML algorithms use prior data for learning and training, RL does not use any sort of prior data. Moreover, The RL model is efficient to solve the problems when there is a need to

teach an AI agent to make decisions in a complex environment.Finally, training does not involve human intervention but instead allowing exploring rules based on self-play only.

the proposed RL model of this thesis is simple, compared to the Related approaches mentioned in the section 3.2. Since they attempted motor based and self steering drilling method,model of this thesis provides a simulation of navigating optimal path along with the bypassing of the existing obstacles. The obstacles can be defined as existing oil wells in that region.

This thesis has fabricated two different programs in Python and Unity3D to imitate real world problem of the trajectory optimization. simlutenously, python program will use to build the 2D scenerio of this case, In 2D case, the general Qlearning algorithm for grid-world environment is used where the agent determines next action based on QTable until it reaches its goal. Since Python does not provide the standard interactive 3D visualization. However, Unity being a game builder, provides vast range of in-game packages and assets that can replicate behaviour of Reinforcement Learning. Hence, popular packages are used to develop the simulation game in unity.

Chapter 4

Implementation

This chapter explains how each part of the system is implemented. The chapter begins by describing the problem and the resulting architecture design. The section 4.2 shows the implementation of 2D program by using q learning, and finally,the section **??** explains the 3D program in Unity.

This paper has described the key problem area in the previous sections, which primarily sheds light on the aspect of well trajectory optimization through the implementation of ML techniques. and The goal of this paper is to find an optimal path between two points. While, the shortest path, in most of the cases, has been considered as optimal path. this paper has theorized RL to present the possible solutions of proposed problem.

This paper is using a 2D program to show the effectively of the algorithm in this case. To some extent, 2D reflect the partial or shallow picture of path tracing of the well, which does not provide the comprehensive or substantial solution in the real life scenario. While it is possible to build more concrete picture through the application of a 3D program. As this paper is mainly concern to present the more efficient path tracing, where the use of 2d and 3D will reflect the more transparent and comparable picture of well trajectory.

4.1 Initial Experimentation

There are several ways to detect the shortest paths, this thesis has been used several tools to exhibit the results. Sequentially, this paper has used TSP and ACO tools before determining the results thorough applying reinforcement learning.

4.1.1 Start and Target Points

The program contains two points: a start point and a target point. These points correspond to the positions of oil deposits on the surface and underground. That is why, in both programs, the start point is located at the top of the diagram and the target point is located at the bottom.

4.1.2 Obstacles

Both programs use obstacles that are similar to real-world oil paths that have been drilled in that area. For both programs, the primary constraint is a soft or a hard obstacle.

Hard obstacle

The hard obstacle is analogous to the existing oil path. In the real world, we cannot drill through an existing oil path while drilling a new one. Thus, when optimizing a trajectory, avoiding previous paths is critical. In both programs, the algorithm determines the optimal path using cost estimation. As a result, the cost/weight of a hard obstacle is significantly greater, preventing the new path from intersecting.

Soft Obstacle

The soft obstruction is analogous to the outer layer of the existing oil path. In ideal cases, drillers avoid drilling new paths adjacent to existing ones, but this can be done if there is no other option. Similarly, the cost/weight of soft obstacles is moderate, allowing the algorithm to perform an overall cost estimation and intersect with it only when the cost of avoiding is significantly greater than the cost of passing.

4.1.3 Cost Reward Calculation

Because RL determines the optimal path based on the total cost, cost estimation is a critical component of the algorithm.

RL Program

Because the program is written in a grid-world environment, each grid has a cost of -1. Additionally, because the program is designed to avoid obstacles whenever possible, the cost of obstacle grids is higher than the cost of a standard grid. The cost of a hard obstacle is set to -1000 because the desired path should avoid it; additionally, a small cost of -3 has been specified for a soft obstacle. Additional obstacles can be added at a different cost.

For reaching the goal, a reward of 100000 is given.

Unity

Unity's path tracing engine behaves similarly to RL. As a result, different prices are assigned to various objects. Due to the engine's similarity to RL, a lower cost of -10 for difficult obstacles is specified. During training, a reward value of one is assigned.

4.2 2D Program

This section discusses the two-dimensional implementation of the program. The primary objective of the 2D program was to find the optimal path while avoiding obstacles (existing oil paths).

This flow chart 4.1 figured the 2D program in python. The flow chart is divided into two parts of the program, where the main flow refers to the initial conditions, and qTables are drawn. It also defines the parameters of the Greedy algorithm and passes to the training episode flow. Max epoch 10000 means maximum training episodes, after completion of the training, the algorithm finds the shortest path from the QTable, and if it reaches to terminal state, the program ends and shows the result.

In training flow diagram, The algorithm gets starting and goal location for the path. If the probability factor is greater than the given random number in the training function, which starts the training procedure. Based on the condition, the movement kicks-on (left,right,up,down). Based on the reward template, the algorithm updates the Q table and sends the information to the shortest path function.

Due to the complexity of the underground rock formations, we determined that a grid-world environment would be optimal for the experiment. The entire program will be discussed in parts below.

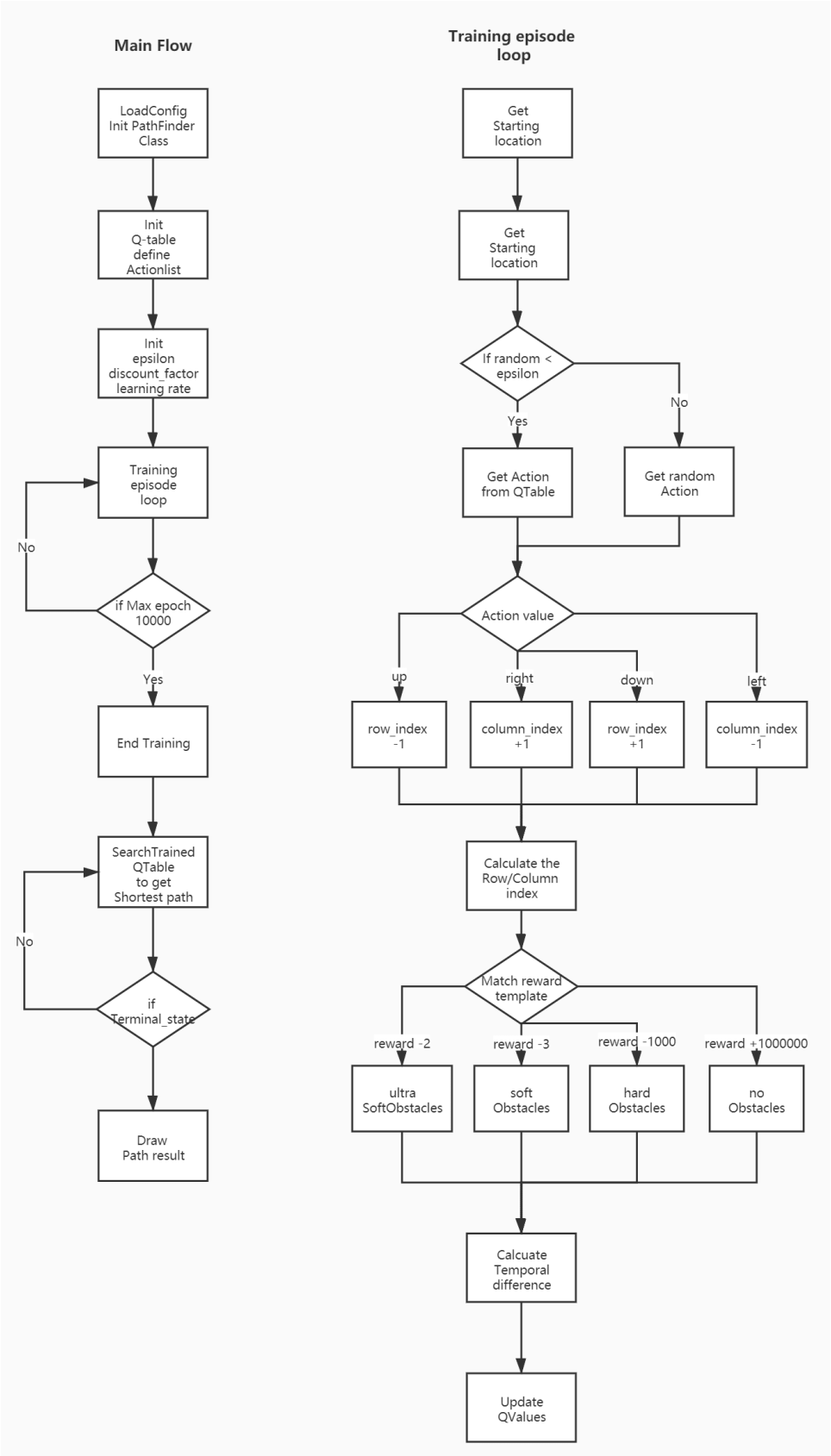

Figure 4.1: Flowchart of QLearning program in 2D

4.2.1 Adding configuration

The configuration begins with the program's initial conditions, such as start and endpoints for the drill path. Then, obstacle coordinates are added. In the real world, a well path is composed of several layers and constraints. first and foremost, there is the existing drilled well path. this is named as the hard layer. consecutively, there is a protective outer layer which is denoted as soft layer that surrounds the hard layer.

```
#Heights for the window are given. coordinates for
start,end points and obstacles. Weight for each layer
and reward for reaching the goal.
config = {
    #simulator window size
    "height": 30,
    "width": 30,
    # coordinate for start position
    "startLoc": [2, 4],
    # coordinate for goal position
    "endLoc": [27, 20],
    #coordinate for soft obstacle
    "softObstacles": [
        {"topLeft": [9, 7], "bottomRight": [10, 9]},
        {"topLeft": [11, 8], "bottomRight": [12, 8]},
        {"topLeft": [12, 9], "bottomRight": [12,10]},
        {"topLeft": [10, 10], "bottomRight": [10, 10]},
        {"topLeft": [11, 11], "bottomRight": [14, 11]},
        ],
    #weight for the hard obstacle
    "softWeight": -4,
    #coordinate for hard obstacle
    "hardObstacles": [
        {"topLeft": [9, 8], "bottomRight": [10, 8]},

        {"topLeft": [11, 9], "bottomRight": [11, 10]},
        {"topLeft": [12, 11], "bottomRight": [12, 11]},

        {"topLeft": [13, 12], "bottomRight": [14, 12]},

        {"topLeft": [15, 13], "bottomRight": [15, 14]},
        {"topLeft": [13, 25], "bottomRight": [13, 25]},
        ],
     #weight for the hard obstacle
    "hardWeight": -1000,
    #Reward for reaching the goal
    "reward": 1000000
}
```

Listing 4.1: Initial configuration for QLearning in Python

The role of the obstacles defines as that no new path can converge through the hard layer. In an ideal scenario, the new path should not converge through the soft layer; however,

if the cost of going around the soft layer is greater than the cost of going through, the algorithm will pass through the soft layer.

Height has been referred in code listing 4.1 is the window size for the environment. Weight/cost of each layer and reward of end point is given as well.

4.2.2 Initializing the Environment and Training

After Declaring the Configuration, the environment is initialized in code listing 4.2. QTable and Actions at each point are also defined.

```
def __init__(self, config):
        self.environment_rows = config['height']
        self.environment_columns = config['width']
        self.q_values = np.zeros((self.environment_rows, self.environment_columns, 4))
        self.actions = ['up', 'right', 'down', 'left']
        self.rewards = np.full((self.environment_rows, self.environment_columns), -1)
        for i in config["semiSoftObstacles"]:
            self.rewards[i["topLeft"][0]:i["bottomRight"][0]+1,
            i["topLeft"][1]:i["bottomRight"][1]+1] = config["semiSoftWeight"]
        for i in config["softObstacles"]:
            self.rewards[i["topLeft"][0]:i["bottomRight"][0]+1,
            i["topLeft"][1]:i["bottomRight"][1]+1] = config["softWeight"]
        for i in config["hardObstacles"]:
            self.rewards[i["topLeft"][0]:i["bottomRight"][0]+1,
            i["topLeft"][1]:i["bottomRight"][1]+1] = config["hardWeight"]
        assert self.rewards[config["startLoc"][0],config["startLoc"][1]]
        != config["hardWeight"]
```

Listing 4.2: Environment configuration for QLearning program

There are some stop/terminal conditions mentioned below:

1. Start point can not be inside hard obstacles.
2. Crushed into hard obstacle
3. Reach End point

```
def is_terminal_state(self, current_row_index, current_column_index):
    return (self.rewards[current_row_index, current_column_index] ==
    config["hardWeight"]) or ((self.rewards[current_row_index, current_column_index]
        == config['reward']))
```

Listing 4.3: Termination condition for QLearning program

Listing 4.4 shows how the agent enumerates many times starting from random position,trying to maximize its reward(reach start-point). It gets a random row and column index and continues choosing random row and column indexes until a non-terminal state is identified.

```
def get_starting_location(self):
        current_row_index = np.random.randint(self.environment_rows)
        current_column_index = np.random.randint(self.environment_columns)
        while self.is_terminal_state(current_row_index, current_column_index):
            current_row_index = np.random.randint(self.environment_rows)
            current_column_index = np.random.randint(self.environment_columns)
        return current_row_index, current_column_index
```

Listing 4.4: Staring condition for QLearning program

Then it chooses next action from given [row,column] state with the highest probability.The epsilon is for adding some randomness to our agent.

```
def get_next_action(self, current_row_index, current_column_index, epsilon):
        if np.random.random() < epsilon: #epsilon=maximum probability
            return np.argmax(self.q_values[current_row_index, current_column_index])
        else:
            return np.random.randint(4)
```

Listing 4.5: choose best action for QLearning program

once the action is chosen,the agent returns the location and updates the action index after making that action.

```
def get_next_location(self, current_row_index, current_column_index, action_index):
        new_row_index = current_row_index
        new_column_index = current_column_index
        if self.actions[action_index] == 'up' and current_row_index > 0:
            new_row_index -= 1
        elif self.actions[action_index] == 'right' and current_column_index
        < self.environment_columns - 1:
            new_column_index += 1
        elif self.actions[action_index] == 'down' and current_row_index
        < self.environment_rows - 1:
            new_row_index += 1
        elif self.actions[action_index] == 'left' and current_column_index > 0:
            new_column_index -= 1
        return new_row_index, new_column_index
```

Listing 4.6: choose next location for QLearning program

The shortest path function in listing 4.7 takes start location as its parameters. Agent will immediately return if this is an invalid starting location. If the start location is valid then continue moving the path until reaches end location. Moreover, while it is moving toward the goal, it updates the shortest path index.After training episode is finished, it accepts the end location,and returns the shortest path of the both locations.

```
def get_shortest_path(self, start_row_index, start_column_index):
        shortest_path = []
        if self.is_terminal_state(start_row_index, start_column_index):
            return shortest_path
        else:
            current_row_index, current_column_index = start_row_index,
            start_column_index
            shortest_path.append([current_row_index, current_column_index])
        while not self.is_terminal_state(current_row_index, current_column_index):
            action_index = self.get_next_action(current_row_index,
            current_column_index, 1.)
            current_row_index, current_column_index = self.get_next_location(current_row_index,
            current_column_index, action_index)
            shortest_path.append([current_row_index,
            current_column_index])
        return shortest_path
```

Listing 4.7: Get shortest Path for QLearning program

Action table of the path taken for shortest path for one condition is given below: There

```
[[27, 28], [27, 27], [26, 27], [26, 26], [26, 25], [25, 25], [24, 25], [24, 24], [24, 23], [24, 22], [24, 21], [24,
20], [24, 19], [24, 18], [24, 17], [23, 17], [23, 16], [22, 16], [21, 16], [21, 15], [20, 15], [19, 15], [19, 14],
[19, 13], [18, 13], [18, 12], [17, 12], [17, 11], [16, 11], [16, 10], [16, 9], [16, 8], [16, 7], [15, 7], [14, 7],
[13, 7], [12, 7], [11, 7], [11, 6], [10, 6], [9, 6], [8, 6], [7, 6], [6, 6], [5, 6], [4, 6]]
```

Figure 4.2: Action table of shortest path QLearning program

are some Training parameters required for the algorithm. These hyper parameters are used in Bellman equation for computing Q values:

1. Epsilon: As agent begins the learning, we would want it to take random actions to explore more paths. But as the agent gets better, the Q-function converges to more consistent Q-values. Now we would like our agent to exploit paths with highest Q-value i.e takes greedy actions.

2. Learning rate: How fast the algorithm learns. 0 means low and 1 is High

3. Discount factor:It quantifies how much importance we give for future rewards. It's also handy to approximate the noise in future rewards. Gamma varies from 0 to 1. If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

```
def train(self, epoch):
        #hyperparameters used in Bellman equation for computing Qvalues
        epsilon = 0.8
        discount_factor = 0.99
        learning_rate = 0.4

        for episode in range(epoch):
            row_index, column_index = self.get_starting_location()

            while not self.is_terminal_state(row_index, column_index):

                action_index = self.get_next_action(row_index, column_index, epsilon)

                old_row_index, old_column_index = row_index, column_index
                row_index, column_index = self.get_next_location(row_index, column_index, action

                reward = self.rewards[row_index, column_index]
                old_q_value = self.q_values[old_row_index, old_column_index, action_index]
                temporal_difference = reward + (discount_factor * np.max(self.q_values[row_index

                new_q_value = old_q_value + (learning_rate * temporal_difference)
                self.q_values[old_row_index, old_column_index, action_index] = new_q_value


try:
    short = PathFinder(config)

    short.train(10000)
except AssertionError as e:
    print(e)
```

Listing 4.8: Training function for QLearning program

The code listing 4.8 refers to the training function. The function performs similar action to shortest path.The agent chooses action,goes to new state,stores old and new [row,column] indexes,receive reward,calculate temporal difference and finally update Qtable according to Bellman equation. Currently, the code does 10000 experimental runs.

4.2.3 Visualization

Final part of the program is visualizing the path the agent chose along with obstacles in a grid based window.Python OpenCV library is used for visualization.

```
def visualize_(boardSize, path, start, end):

    rows = boardSize[0] * 20
    cols = boardSize[1] * 20
    img = np.ones((cols + 1, rows + 1, 3), dtype = "uint8") * 250 #3channel

    stepX = int(rows / 20) + 1
    stepY = int(cols / 20) + 1
    x = np.linspace(start = 0, stop = rows, num = stepX)
    y = np.linspace(start = 0, stop = cols, num = stepY)

    v_xy = []
    h_xy = []
    for i in range(stepX):
        v_xy.append([int(x[i]), 0, int(x[i]), cols-1])
    for i in range(stepY):
        h_xy.append([0, int(y[i]), rows-1, int(y[i])])


    for i in range(stepX):
        [x1, y1, x2, y2] = v_xy[i]
        cv2.line(img, (x1,y1), (x2, y2), (255,255,255), 1)
    for i in range(stepY):
        [x1_, y1_, x2_, y2_] = h_xy[i]
        cv2.line(img, (x1_,y1_), (x2_ + 1, y2_), (255,255,255), 1)

    for obs in config['softObstacles']:
        cv2.rectangle(img,(obs['topLeft'][1]*20, obs['topLeft'][0]*20),
        ((obs['bottomRight'][1]+1)*20, (obs['bottomRight'][0]+1)*20), (100,0,100),2)

    for obs in config['hardObstacles']:
        cv2.rectangle(img,(obs['topLeft'][1]*20, obs['topLeft'][0]*20),
        ((obs['bottomRight'][1]+1)*20, (obs['bottomRight'][0]+1)*20), (255,0,255),2)

    color = (255,0,0)
    for ind in range(len(path) - 1):
        cv2.line(img, (path[ind][1]*20 + 10,path[ind][0]*20 + 10),
        (path[ind + 1][1]*20 + 10, path[ind + 1][0]*20 + 10), color, 3)
    cv2.circle(img,(start[1]*20+10,start[0]*20+10), 3, (0,255,255), -1)
    # for end in endPoints:
    cv2.circle(img,(end[1]*20+10,end[0]*20+10), 3, (255,0,255), -1)

    cv2.namedWindow('Optimal path', cv2.WINDOW_NORMAL)
    cv2.imshow('Optimal path', img)
    cv2.waitKey(0)
def visualize(path):
    for p in range(len(path),-1,-1):
        visualize_((config['height'], config['width']), path[p:], config['startLoc'],
        config['endLoc'])
    cv2.destroyAllWindows()
```

Listing 4.9: Visulization function for QLearning program

In the code listing 4.9, CV2.line and CV2.linespace are used to draw the gridshape window. cv2.rectangle Creates the rectangle shaped obstacles for each obstacle. Moreover, CV2.line and CV2.circle are used to create the path and start,end locations. The program draws the path by pressing Space bar in the keyboard.Once the drawing is finished, the window closes.

4.3 3D program

The 3D program is built using Unity3D. While the initial plan was to write RL algorithm in python, visualization in unity and connect them using an APi, the execution time would be long and complex. Moreover, Unity has stand alone functionality and packages that provides solution to find optimal path avoiding obstacles. The program is built using c language. A basic workflow diagram is shown below: The Program revolves

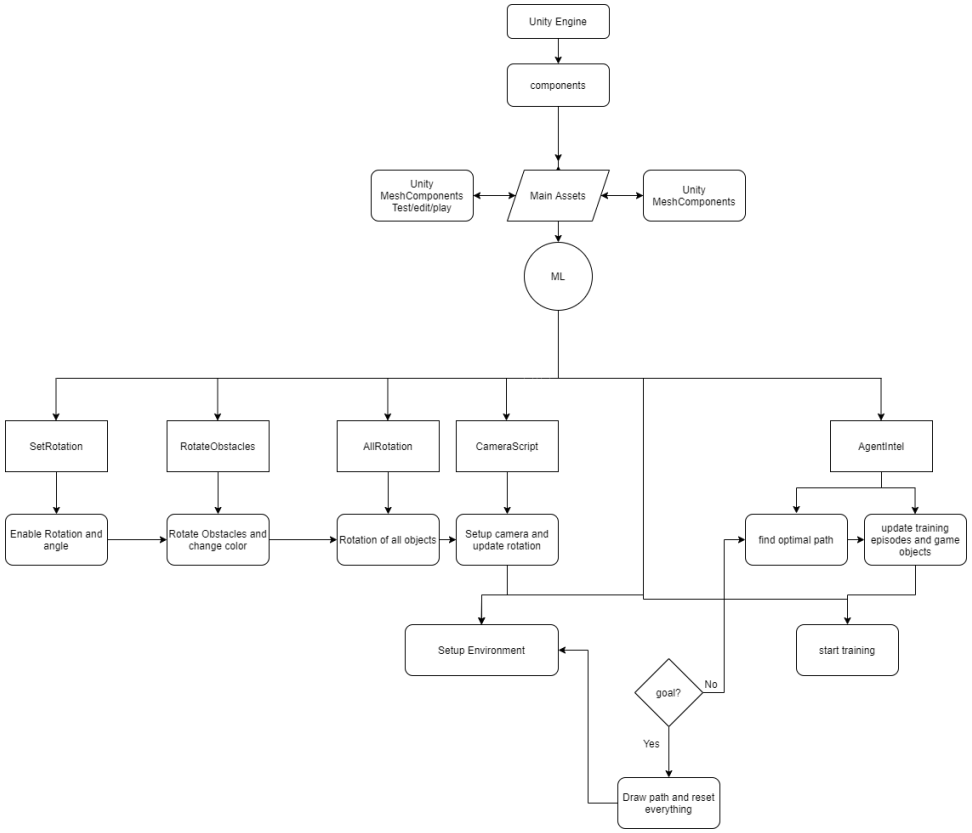

Figure 4.3: Flowchart of Unity program in 3D

around the Unity Engine as Unity needs supporting packages and assets to run. Mesh components refers to Unity path finding AI tools while main asset refers to written

scripts. Six scripts are written in the program. ML is the main Script, which sets up the environment, enables the objects, the functions and finally gives output. AgentAI mainly gets object information from the scripts, runs training episodes and finds an optimal path. Rest of the scripts are built to make games objects and camera.

4.3.1 Game Components and Assets

There are several Game components and assets used to build the program. Built-in Unity components are explained and Custom scripts will be explained next section.

Unity UI

Unity UI is a UI toolkit for developing user interfaces for games and applications. It is a GameObject-based UI system that uses Components and the Game View to arrange, position, and style user interfaces.

Unity Physics

Unity helps developers to simulate physics in the Project to ensure that the objects correctly accelerate and respond to collisions, gravity, and various other forces. Unity provides different physics engine implementations which can be used according to Project needs: 3D, 2D, object-oriented, or data-oriented.

Unity's built-in 3D physics engine, which you can use in object-oriented projects. It includes the following:

1. Main physics-related concepts: Rigidbodies, Colliders, Joints, physics articulations and Character Controllers.
2. Specific physics contexts: Continuous collision detection and Multi-scene physics.

Unity Engine.AI

The AI module implements the path finding features in Unity.theere are number of tools are being prevailed to define the path finding. But this thesis applies NavMesh, as it is catered more detail and holistic view .

NavMesh has several classes that are used in this program. including the following:

1. NavMesh Link: NavMesh Link creates a navigable link between two locations that use NavMeshes.

2. NavMeshModifiers: NavMesh Modifiers adjust how a specific GameObject behaves during NavMesh baking at runtime.

3. NavMeshModifierVolume: NavMesh Modifier Volume is useful for marking certain areas of walkable surfaces that might not be represented as separate geometry, for example danger areas.It to make certain areas non-walkable. in this program, obstacles have this behaviour.

4. NavMeshSurface: The NavMesh Surface component represents the walkable area for a specific NavMesh Agent type, and defines a part of the Scene where a NavMesh should be built.

Unity MonoBehaviour

MonoBehaviour is an API that is a base class from which every Unity script derives. When using C, It must explicitly derive from MonoBehaviour. some of the MonoBehaviour functions are : Start(),Update(),FixedUpdate(), LateUpdate().

4.3.2 Custom Assets

Apart from built-in assets, there are also some scripts to build the program. They are discussed below.

CameraScript

The CameraScript is used for 360 degree camera rotation and movement of the objects using Arrowkeys and speed. Code listing 4.10 refers to the code:

```
void Update()
{
    /*
            yaw += speedH * Input.GetAxis("Mouse X");
            pitch -= speedV * Input.GetAxis("Mouse Y");

            transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);*/
    if (!TranslationAllowed)
        return;

    if (Input.GetKey(KeyCode.UpArrow) && transform.position.z < -20f)
    {
        transform.Translate(0,0,0.2f);
```

```
        }
        if (Input.GetKey(KeyCode.DownArrow) && transform.position.z > -120f )
        {
            transform.Translate(0, 0, -0.2f);
        }

        if (Input.GetKey(KeyCode.RightArrow) && transform.position.x < 20f)
        {
            transform.Translate(0.2f, 0, 0);
        }
        if (Input.GetKey(KeyCode.LeftArrow) &&  transform.position.x > -20f)
        {
            transform.Translate(-0.2f, 0, 0f);
        }

    }
}
```

Listing 4.10: Camera Script for Unity program

All Rotation

Code listing 4.11 is a helper script related to movement and rotation for all objects, camera and mouse. It updates animation, screen surface,camera view and toggles rotation of the objects.

```
        void Start () {
                View360.color = Color.gray;
        }
        public Animator ToggleAnim;
        private bool IN = true;
        public void ToogleBut()
        {
                if (!IN)
                {
                        ToggleAnim.SetInteger("Toggle", 2);
                        IN = true;
                        return;
                }
                if (IN)
                {
                        ToggleAnim.SetInteger("Toggle", 3);
                        IN = false;
                }
        }
        public Transform Platform;
        // Update is called once per frame
        void FixedUpdate () {


                if (AllRotationAllowed &&  Input.GetMouseButton(0) &&
                Input.GetAxis("Mouse X") != 0 )
```

```
                {

                        transform.RotateAround(Platform.position,Vector3.up,5*
                        Input.GetAxis("Mouse X"));
                        transform.RotateAround(Platform.position, Vector3.left,
                        5 * Input.GetAxis("Mouse Y"));
                }

                mPrevPos = Input.mousePosition;
        }
        public void TurnOnView()
        {
                if (!AllRotationAllowed)
                {
                        AllRotationAllowed = true;
                        View360.color = Color.blue;

                        return;
                }

                if (AllRotationAllowed)
                {
                        AllRotationAllowed = false;
                        View360.color = Color.gray;


                        return;
                }
        }
        public void StopRotationn()
        {
                AllRotationAllowed = false;
                View360.color = Color.gray;

        }
        public void ResetView()
    {

                GameObject.Find("Platform").transform.eulerAngles = Vector3.zero;
                transform.position = Vector3.zero - Vector3.forward * 100;
                transform.eulerAngles = Vector3.zero;
        }

}
```

Listing 4.11: AllRotation Script for Unity program

RotateObstacles

Code listing 4.12 is another helper script for rotation of the obstacles. There are two types of cylindrical obstacles used. Vertical and Horizontal. The update function updates the movement of the obstacles using Arrowkeys and rotation using mouse movement.

```
void Update()
{
    if (gameObject.tag != "Player")
    {
        if (gameObject.tag != "agent" && Selected &&
        Input.GetAxis("Mouse X") != 0 &&   gameObject.tag
        != "Finish" && transform.eulerAngles != Vector3.zero)
        {

            transform.RotateAround(transform.position, Vector3.up,
            5 * Input.GetAxis("Mouse X"));

        }
    if (gameObject.tag != "agent" && Selected &&
    Input.GetAxis("Mouse X") != 0 && gameObject.tag == "Finish")
        {

            transform.RotateAround(transform.position,
            Vector3.right, 5 * Input.GetAxis("Mouse X"));
            transform.RotateAround(transform.position,
            Vector3.forward, 5 * Input.GetAxis("Mouse Y"));

        }

        if (Input.GetKeyDown(KeyCode.Delete) && Selected &&
        gameObject.tag != "Player")
        {
            Destroy(gameObject);
        }
        if (Input.GetKey(KeyCode.W) && Selected &&
        gameObject.tag != "agent")
        {
            transform.localScale -= Vector3.up * 0.1f;
        }
        if (Input.GetKey(KeyCode.S) && Selected)
        {
            transform.localScale += Vector3.up * 0.1f;
        }
        if (Input.GetKey(KeyCode.LeftArrow) && transform.position.z
        < 55f && Selected)
        {
            transform.Translate(0, 0, 0.1f);
        }
        if (Input.GetKey(KeyCode.RightArrow) && transform.position.z
        > -35f && Selected)
        {
            transform.Translate(0, 0, -0.1f);
        }
```

```
            if (Input.GetKey(KeyCode.DownArrow) && transform.position.x
            < 22f && Selected)
            {
                transform.Translate(0, -0.1f, 0);
            }
            if (Input.GetKey(KeyCode.UpArrow) && transform.position.x >
            -22f && Selected)
            {
                transform.Translate(0f, 0.1f, 0f);
            }
            if (Input.GetKey(KeyCode.A) && transform.position.x < 22f
            && Selected)
            {
                transform.Translate(-0.1f, 0f, 0);
            }
            if (Input.GetKey(KeyCode.D) && transform.position.x > -22f
            && Selected)
            {
                transform.Translate(0.1f, 0f, 0f);
            }
        }
        else
        {
            if (Input.GetKey(KeyCode.DownArrow) && transform.position.x
            < 22f && Selected)
            {
                transform.Translate(0, -0.1f, 0);
            }
            if (Input.GetKey(KeyCode.UpArrow) && transform.position.x
            > -22f && Selected)
            {
                transform.Translate(0f, 0.1f, 0f);
            }
        }
    }
```

Listing 4.12: Update function in RotateObstacles Script for Unity program

It also has color function to distinguish the color of the selected cylinder. Functions in listing 4.13 changes obstacle color pink when the object is selected for rotation or movement.

```
    public void Selecteded()
    {
        AllRotation.AllRotationAllowed = false;
        Selected = true;
        FindObjectOfType<AllRotation>().StopRotationn();
        GetComponent<MeshRenderer>().material.color = Color.magenta;
        CameraScript.TranslationAllowed = false;
    }
    public void ResetmColor()
    {
        Selected = false;
```

```
        GetComponent<MeshRenderer>().material.color = DefColor;
        CameraScript.TranslationAllowed = true;

    }
```

Listing 4.13: color change in RotateObstacles Script for Unity program

SetRotation

Code listing4.14 is helper script to define rotation angle and enable rotation of the objects.

```
private void OnEnable()
    {
        if (gameObject.tag == "pit")
        {

            //    transform.eulerAngles = new Vector3(0, 90, -90);

        }
        if (gameObject.tag == "goal")
        {

                transform.eulerAngles = new Vector3(90, 0, 0);
        }
    }
```

Listing 4.14: SetRotation Script for Unity program

AgentIntel

This Script is creates training episodes and agent intelligence. The agent tries to find optimal path here in below code listing 4.15. Once the path is found the function sends message to Update function to draw the path.

```
private void ST()
    {
        body = GetComponent<Rigidbody>();
        //   agent = GetComponent<NavMeshAgent>();
        Status = GameObject.Find("Status").GetComponent<Text>();
        transform.position = InitialPos;
        GetComponent<TrailRenderer>().time = 8000;
        Status.text = "Status : Finding Path";
        FisrtCheck = false;
        Check = false;
        StopPositionaing = true;
        Target = GameObject.FindGameObjectWithTag("goal").transform.position;
        GameObject e = GameObject.FindGameObjectWithTag("value");
        if (e != null && transform.position.y < -12.0f)
```

```
        {       Target = Target+ Vector3.up * 3.5f;
            Invoke("S",1.0f);
        }   Begin = true;
    }
```

Listing 4.15: find optimal path function for Unity program

ML

This is the main Script of the program. This script initialises the environment and objects.It enables start,end positions, obstacles.It also resets all previous activities.Starts training episodes to find optimal path in code listing 4.15. Training function is given below

```
public void StartTraining()
    {
        // perform check
        if (c < 2)
            return;
        TraceText.text = "Retrace Path";
      Transform target =   GameObject.FindGameObjectWithTag("Player").transform;
        Instantiate(Agent, new Vector3(target.position.x, target.position.y, target.position.z),
        GameObject.FindGameObjectWithTag("agent").SendMessage("ST");
        GameObject.Find("FP").SetActive(false);
    }
```

Listing 4.16: Training function in ML Script for Unity program

Once the optimal path is found, The update function in listing4.17 draws the path and ends program.

```
private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
        Ray raye = cam.ScreenPointToRay(Input.mousePosition);
                RaycastHit hite;
                if (Physics.Raycast(raye, out hite))
                {
                    if (hite.transform.tag == "pit")
                    {
                        hite.transform.parent.GetComponent<RotateObstacles>().Selecteded();
                    Debug.Log("Hitting .....");
                    return;
                }
                if (hite.transform.tag == "Player" )
                {
                    hite.transform.GetComponent<RotateObstacles>().Selecteded();
                    Debug.Log("Hitting .....");
                    return;
                }}
```

```
if (PlacedEnd)
    {
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            dummy = GameObject.FindGameObjectWithTag("goal");
            if (dummy != null)
                Destroy(dummy);
            Instantiate(EndStar, new Vector3(hit.point.x,
            Box.position.y + 1f, hit.point.z), Quaternion.identity, Box);
        }}
if (PlacedStarter)
    {
        dummy = GameObject.FindGameObjectWithTag("agent");
        if (dummy != null)
            Destroy(dummy);

        dummy = GameObject.FindGameObjectWithTag("Player");
        if (dummy != null)
            Destroy(dummy);
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            Instantiate(Starter, new Vector3(hit.point.x,
            Box.position.y + 1f, hit.point.z),
            Quaternion.identity, Box);

        }}
if (PlaceHobs)
    {
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            Instantiate(HObsPrefab, new Vector3(hit.point.x,
            Box.position.y + 1.3f, hit.point.z),
            HObsPrefab.transform.rotation, Box);
        } }
if (PlaceVobs)
    {
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            Instantiate(VObsPrefab, new Vector3(hit.point.x,
            Box.position.y + 4.3f, hit.point.z),
            VObsPrefab.transform.rotation, Box);
        } }
ResetAll();
resetObstacle();}
```

Listing 4.17: Update function in ML Script for Unity program

Chapter 5

Result

This chapter presents the results from the Implementation done in Chapter 4. It starts by showing and explaining the results from both Python and Unity programs.

5.1 Results from QLearning in Python

Fig 5.1, shows a path using two existing obstacles. As the randomness factor is closer to 0, the algorithm bend as much as possible to look like a curved path. The result is found after 10000 training episode. Moreover, in all the figures start point is coloured as yellow, end/goal is pink. Hard obstacle is pink rectangles, soft obstacle is chocolate colour and path is represented in blue.

Its should be noted that, the result is not unique. With same condition, different path/result can be found due to the fact that all the paths/results has same total cost.

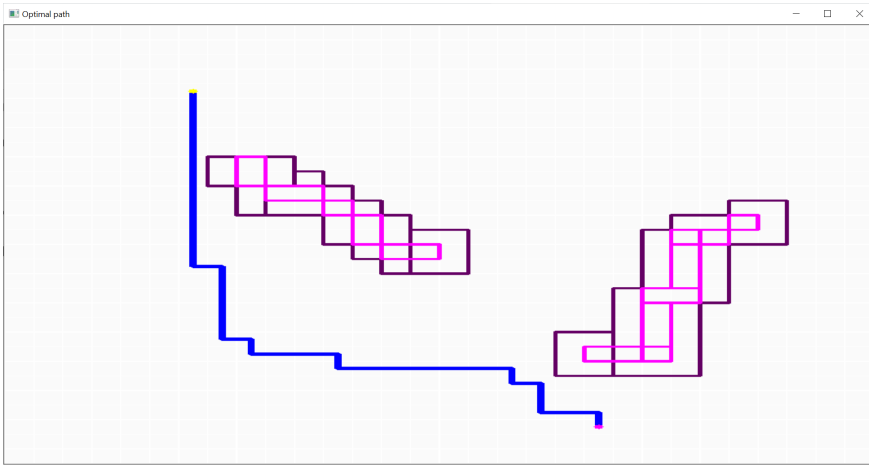

Figure 5.1: QL result using 2 obstacles(after training 10000 times)

Fig 5.2, shows a path after training 50000 times with same criteria. As qLearning uses greedy approach, it always tries to find simpler and shortest path to reach its goal. Hence, it gave more flat and straight path as much as possible. This also indicates that, if the start point can be moved closer towards goal on the surface, more drilling cost can be saved.

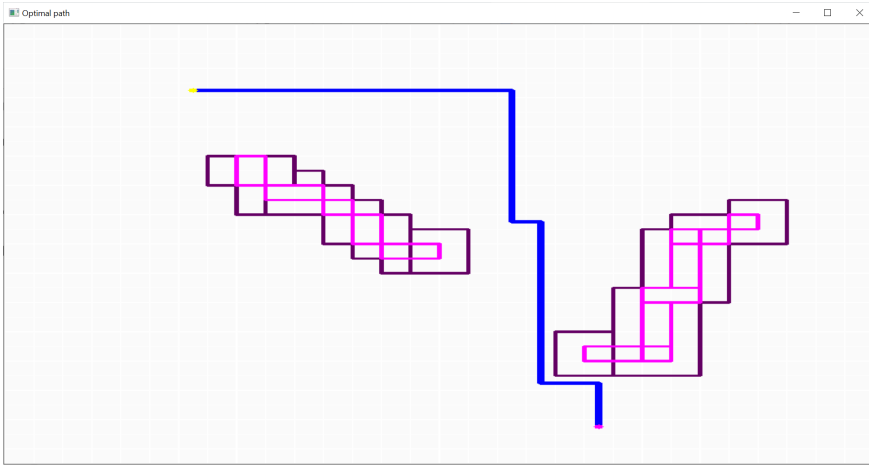

Figure 5.2: QL result using 2 obstacles (after training 50000 times)

Fig 5.3, shows similar path to 5.1 when a horizontal obstacle is added.it should be noted that, The start and target points are in the same position.

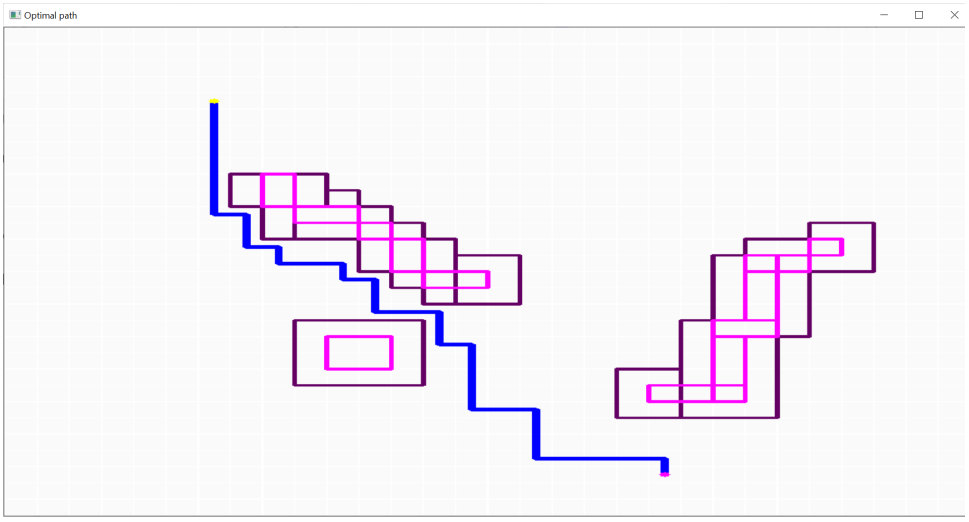

Figure 5.3: QL result using 3 obstacles

In Fig 5.4, position of one obstacle is changed but the algorithm achieved similar result as before. In Fig 5.5, position of both obstacles are changed. Position of goal point is

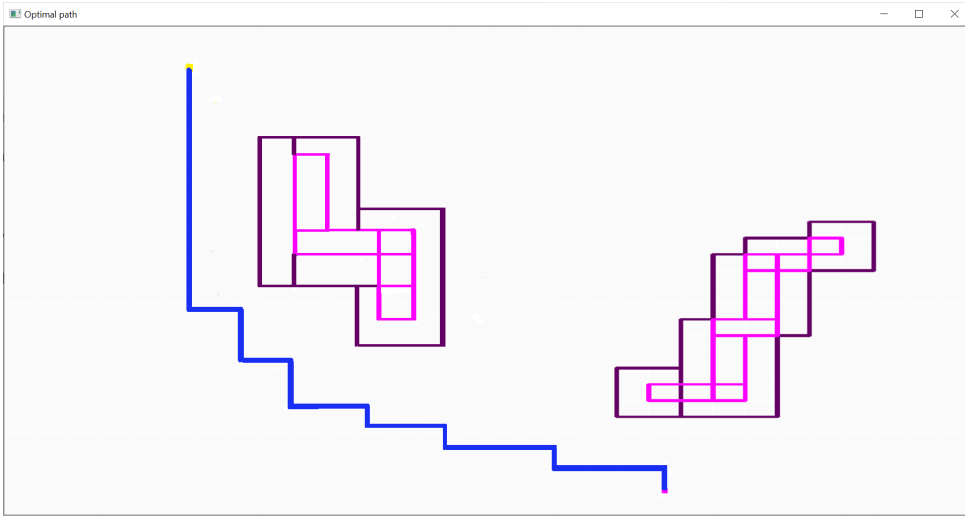

Figure 5.4: QL result using 2 obstacles

changed as well. It can be seen that, as there is little space to find path in the bottom, the algorithm showed a similar path as Fig 5.2.

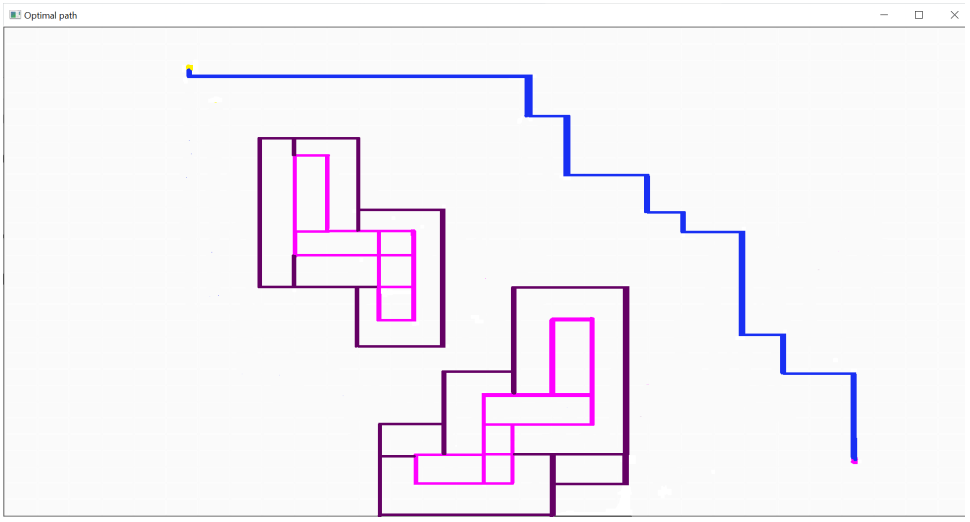

Figure 5.5: QL result using 2 obstacles

5.2 Results from Unity program

As there are only two directions(vertical and horizontal) used for the obstacles, making a existing curvy path is challenging. Therefore, more obstacles are added so that the path has more angles. As it is a 3D program, 5.6 to 5.10 shows results from different angles.

There are two layers of the obstacle. the inner layer is hard obstacle and outer padding is soft obstacle. Start point is shown as green ball where the goal is represented by a star. The blue ball represents the path tracing agent or driller.

The grid plane refers to lowest level of the simulation to help the user position the objects.

Fig 5.6, shows a simple path between two obstacles. As the conjunction points are pretty straightforward, the path showed in almost straight line.

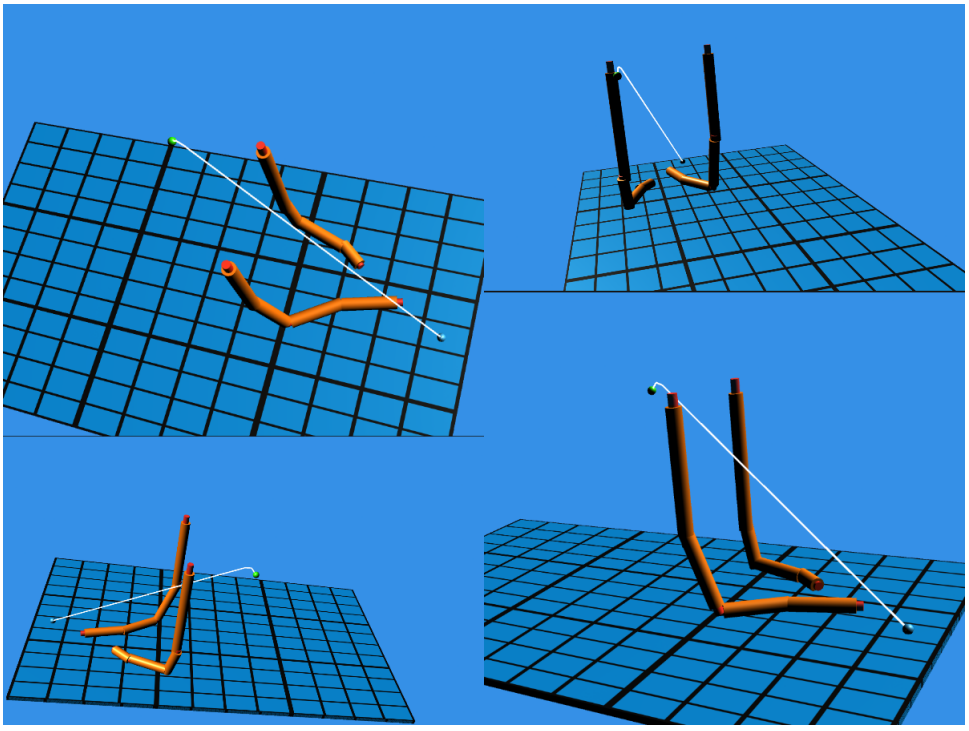

Figure 5.6: Unity result using 2 obstacles

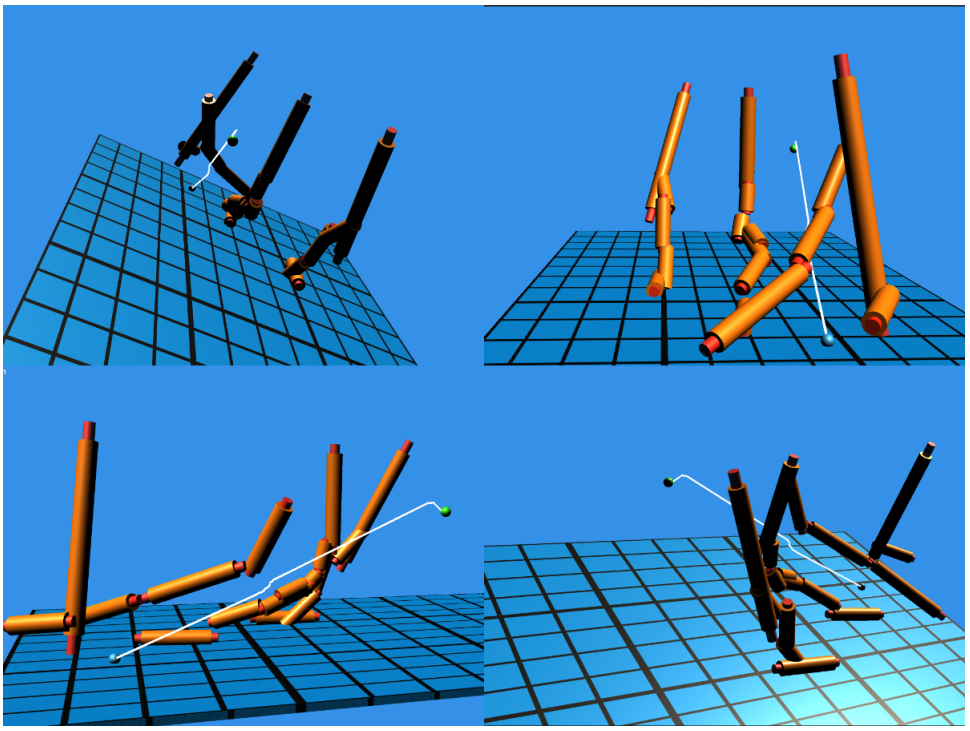

Figure 5.7: Unity result using 4 obstacles

Fig 5.7, shows again a simple path although there were 4 obstacles tightly packed to each other. Although the path is mostly straight, there was little bend and curve to avoid the obstacles and reach target.

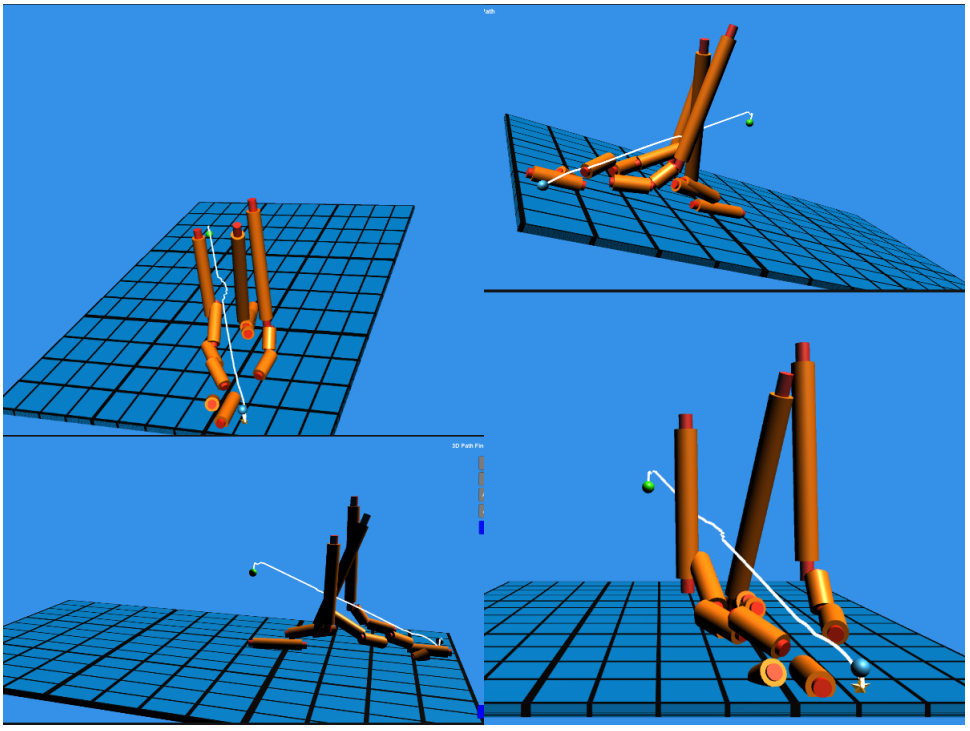

Figure 5.8: Unity result using 3 obstacles

Fig 5.8, shows more bended path than previous ones. the path keeps distance from the obstacles and makes little tweaks and turns to reach its goal.

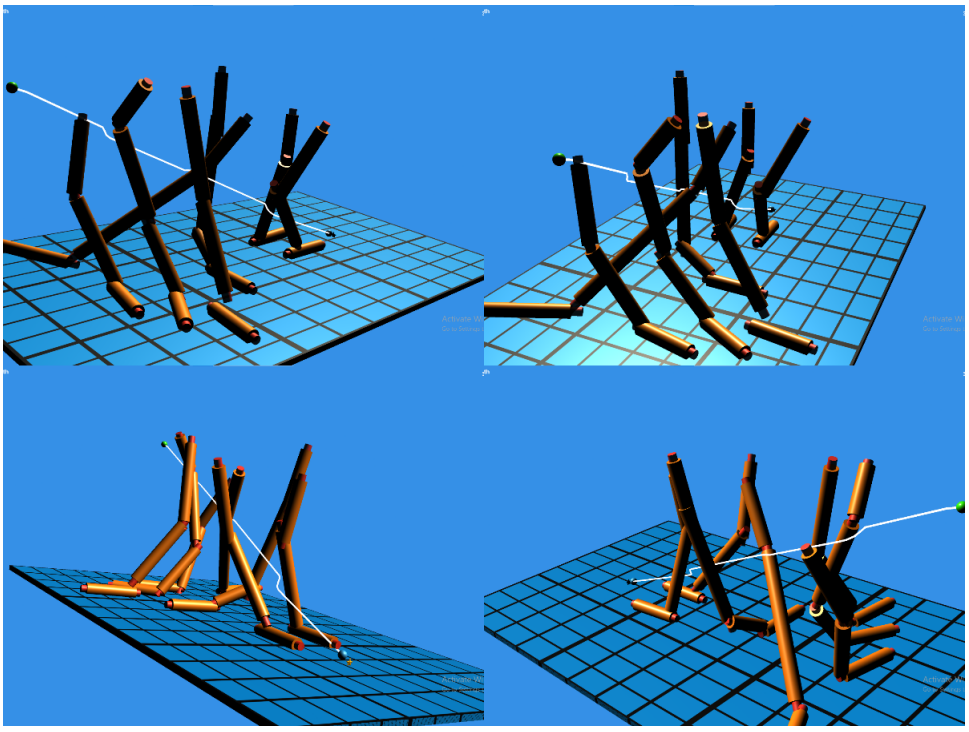

Figure 5.9: Unity result using 7 obstacles

Fig 5.9, rather simple path although 7 complex obstacles are added. It can be seen that, the path makes necessary turns to reach its destination. The path may appear to pass through the obstacles from one angle due to low graphics.

Fig 5.10, also provides similar result although more obstacles are added. It should be noted that, as there is space, the algorithm does not try to go through soft obstacles.

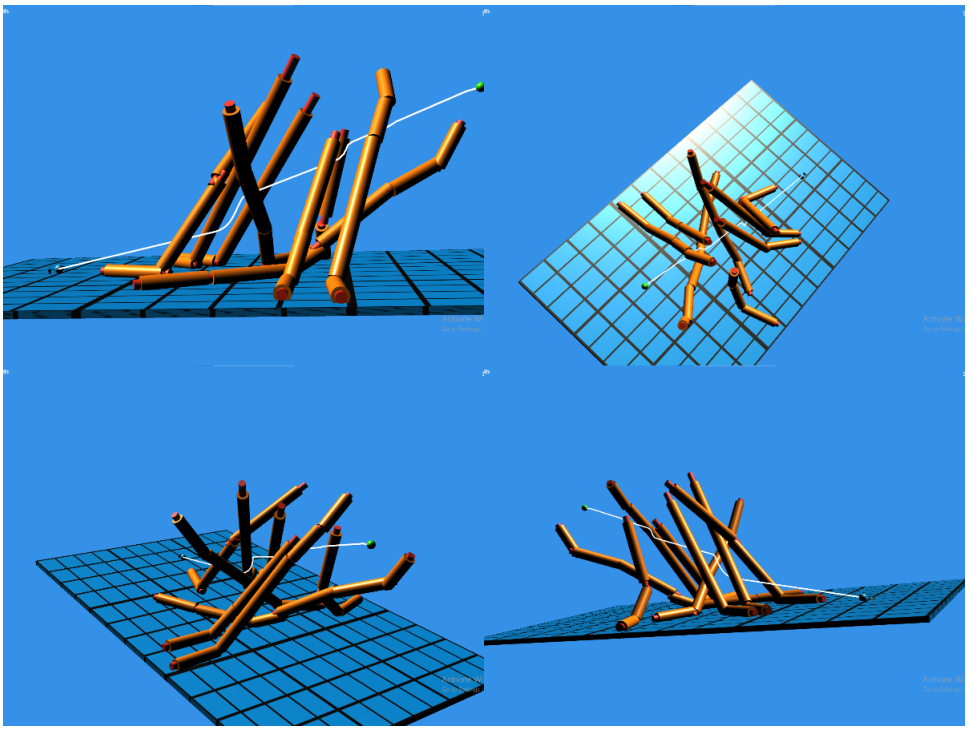

Figure 5.10: Unity result using 8 obstacles

Chapter 6

Discussion and Conclusion

6.1 Discussion on the results

According to the results presented in Chapter 5, both the RL and unity programs behave similarly in terms of performance.

As only obstacles are given as a condition, both programs find a path avoiding the obstacles. It should be noted that as there are only two dimensions in RL program, the result is simpler than unity result.

If more conditions are added, their behaviour will change. If condition of mud layer is given, they may try to make a path close to obstacles or go through soft obstacle.

6.2 Limitations

Both program has limitation in term of performance. With less condition such as rock formation, mud layer or steering condition, the algorithm provide less realistic result.

Also, in Unity program, there are some build issues. As the pixel density is low, the image from one side can be misleading. A larger variety of obstacle shape can affect the algorithm.

6.3 conclusion

The objective of this thesis was to investigate and simulate Reinforcement learning for well path optimization. There are similar works going on for well path optimization using different machine learning algorithms and enhancing existing optimization techniques.However, the result shows that Reinforcement learning can be good option for path optimization.

At the very first chapter, in section 1.3, there were some research questions that this thesis was going to answer. The answers are as follows:

1. Is it beneficial to use Machine learning for well path optimization?

Ans: Yes. Machine learning is proven method to complement existing optimization techniques.

2. Can Reinforcement Learning perform in path optimization?

Ans: Due to its behaviour of self learning and no prior data requirements, Reinforcement Learning performs better to simulate path optimization.

3. Can Unity perform in path optimization?

Ans: With vast amount of ML and visualization packages, Unity is a great tool to simulate reinforcement learning for 3D environments.

4. Can RL or unity perform better or less than other existing optimization algorithms (genetic algorithm, particle swarm algorithm, etc.)?

Ans: With the improvement of existing algorithms, it is hard to compare performance with Reinforcement Learning to guide a drill. However, Reinforcement Learning can achieve good results.

List of Figures

Appendix A

Downloadable content

A.1 2D program

The code of the 2D program can be found here. link.

A.2 Unity program

The source code of the unity code can be found here. link.

A.2.1 Game

The game version of the program can be found here. link.

The game Control is given below:

To move the obstacle:

-A,D key X direction. -Up, Down key Y direction. -Left, Right Key Z direction.

M,S key To scale the obstacle.

Bibliography

[1] SPE. Directional Drilling. https://petrowiki.spe.org/Directional$_d$*rilling*.

[2] Angelsen S. Sollie. O.K. Suyuthi A. Mistry R. Myrseth P. Tveiten Ellingsen, H.P. Study on machine learning in the norwegian petroleum industry. 2020. URL https://www.og21.no/en/strategy-and-analyses/og21-studies-and-analyses/previous-years-studies/.

[3] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. *CoRR*, abs/1808.03196, 2018. URL http://arxiv.org/abs/1808.03196.

[4] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039, 2018. doi: 10.1109/ICRA.2018.8461233.

[5] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A research framework for deep reinforcement learning. *CoRR*, abs/1812.06110, 2018. URL http://arxiv.org/abs/1812.06110.

[6] Expert System. What is machine learning? a definition. URL https://expertsystem.com/machine-learning-definition.

[7] Tom Mitchel. *Machine Learning*. McGraw-Hill, illustrated edition, 1997. ISBN 0071154671, 9780071154673. URL http://www.cs.cmu.edu/~tom/mlbook.html.

[8] Wikipedia contributors. Reinforcement learning — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1029168114, 2021. [Online; accessed 19-June-2021].

[9] Andre Violante. Simple reinforcement learning: Q-learning. https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56, 2019.

[10] Suraj Bansal. Rl explained- reinforcing the intuition and math. https://medium.datadriveninvestor.com/rl-explained-reinforcing-the-intuition-and-math-fd1185369186, 2020. [Online; accessed 28-June-2021].

[11] Van-Hai Bui, A. Hussain, and Hak-Man Kim. Double deep q -learning-based distributed operation of battery energy storage system considering uncertainties. *IEEE Transactions on Smart Grid*, 11:457–469, 2020.

[12] Chi Jin, Zeyuan Allen-Zhu, Sébastien Bubeck, and Michael I. Jordan. Is q-learning provably efficient? *CoRR*, abs/1807.03765, 2018. URL http://arxiv.org/abs/1807.03765.

[13] Chathurangi Shyalika. A beginners guide to q-learning. https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c, 2019. [Online; accessed 28-June-2021].

[14] Team-Saida. Gridworld. https://teamsaida.github.io/SAIDA$_RL/GridWorld/$.

[15] Wikipedia contributors. Unity (game engine) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1026656533, 2021. [Online; accessed 27-June-2021].

[16] Unity Technologies . Navmesh. https://docs.unity3d.com/ScriptReference/AI.NavMesh.html, 2021. [Online; accessed 28-June-2021].

[17] Suzanne Ma. Travelling salesman problem. https://blog.routific.com/travelling-salesman-problem, 2020.

[18] Macura, Wiktor K. Ant colony algorithm. https://mathworld.wolfram.com/AntColonyAlgorithm.html, 2020.

[19] Tom Inglis. *Directional Drilling*. Springer Netherlands, Netherlands, 1987. ISBN 978-94-017-1270-5.

[20] Joosten Gerard J.P. Brouwer Roald Vlemmix, Stijn and Jan-Dirk Jansen. Adjoint-based well trajectory optimization. 2009. doi: https://doi.org/10.2118/121891-MS.

[21] Zandvliet M.J. van Essen G.M. Brouwer D.R. Handels, M. and J.D. Jansen. Adjoint based well-placement optimization under production constraints. 2007. doi: https://doi.org/10.2118/105797-MS.

[22] Durlofsky Louis J. Yeten, Burak and Khalid Aziz. Optimization of nonconventional well type, location, and trajectory. *SPE J*, 2003. doi: https://doi.org/10.2118/86880-PA.

[23] O. Badru and C. S Kabir. Well placement optimization in field development. 2003. doi: https://doi.org/10.2118/84191-MS.

[24] A. Emerick, Eugênio Silva, B. Messer, L. F. Almeida, Dilza Szwarcman, M. Pacheco, and Marley M. B. R. Vellasco. Well placement optimization using a genetic algorithm with nonlinear constraints. 2009.

[25] Park C. Kang J. M. Lee, J.W. and C. K Jeong. Horizontal well design incorporated with interwell interference, drilling location, and trajectory for the recovery optimization. 2009. doi: https://doi.org/10.2118/125539-MS.

[26] B.L. Beckner and X. Song. Field development planning using simulated annealing - optimal economic well scheduling and placement. 1995. doi: https://doi.org/10.2118/30650-MS.

[27] eDrilling. Products. https://edrilling.no/products/, 2021. [Online; accessed 28-June-2021].

[28] petromehras contribution. drilling-completion-software:wellahead. https://petromehras.com/petroleum-software-directory/drilling-completion-software/wellahead, 2019. [Online; accessed 28-June-2021].

[29] petromehras contribution. drilling-completion-software:wellplanner. https://www.petromehras.com/petroleum-software-directory/drilling-completion-software/wellplanner, 2019. [Online; accessed 28-June-2021].

[30] petromehras contribution. drilling-completion-software:wellsim. https://www.petromehras.com/petroleum-software-directory/drilling-completion-software/wellsim, 2019. [Online; accessed 28-June-2021].

[31] oliasoft. Welldesign. https://www.oliasoft.com/oliasoft-welldesign/, 2021. [Online; accessed 28-June-2021].

[32] Schlumberger press release. Schlumberger announces delfi cognitive ep environment. https://www.slb.com/newsroom/press-release/2017/pr-2017-0913-delfi, 2017. [Online; accessed 28-June-2021].

[33] Schlumberger . Drillplan. https://www.software.slb.com/delfi/delfi-experience/drillplan, 2018. [Online; accessed 28-June-2021].

[34] Schlumberger . Drillbench. https://www.software.slb.com/products/drillbench#sectionFullWidthTable, 2015. [Online; accessed 28-June-2021].

[35] Halliburton . H012161 datasheet. https://www.landmark.solutions/Portals/0/LMSDocs/Datasheets/WellPlan_Software_DATASHEET-.pdf, 2016. [Online; accessed 28-June-2021].

[36] Halliburton . Decisionspace®365. https://www.landmark.solutions/Portals/0/LMSDocs/PDF/DecisionSpace365.pdf?ver=2021-04-13-194707-547, 2020. [Online; accessed 28-June-2021].

[37] Khosravanian R. Wood D.A. Mansouri, V. Optimizing the separation factor along a directional well trajectory to minimize collision risk. *J Petrol Explor Prod Techno*, 10: 2113–2125, 2020.

[38] Daniel Busby; Frédérik Pivot; Amine Tadjer. Use of data analytics to improve well placement optimization under uncertainty. *Abu Dhabi International Petroleum Exhibition Conference*, 2017. doi: https://doi.org/10.2118/188265-MS.

[39] Ranran Lu; Fahim Forouzanfar; A. C. Reynolds. Bi-objective optimization of well placement and controls using stosag. *SPE Reservoir Simulation Conference*, 2017. doi: https://doi.org/10.2118/182705-MS.

[40] Zhang H. Gao D Sun, T. Application of the artificial fish swarm algorithm to well trajectory optimization. *Chem Technol Fuels Oils 55, 213–218*, 2019. doi: https://doi.org/10.1007/s10553-019-01023-7.

[41] Wendi Huang, Min Wu, Luefeng Chen, Jinhua She, Hiroshi Hashimoto, and Seiichi Kawata. Multiobjective drilling trajectory optimization considering parameter uncertainties. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–10, 2020. doi: 10.1109/TSMC.2020.3019428.

[42] Kshitij Dawar. Reinforcement learning for well location optimization.

[43] Chen Wei Liu Qiuhua Chau Minh Vesselinov Velizar Yu, Yingwei and Richard Meehan. Training an automated directional drilling agent with deep reinforcement learning in a simulated environment. *SPE/IADC International Drilling Conference and Exhibition*. doi: https://doi.org/10.2118/204105-MS.

[44] Hao Liu, Dandan Zhu, Yi Liu, Aimin Du, Dong Chen, and Zhihui Ye. A reinforcement learning based 3d guided drilling method: Beyond ground control. In *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, ICNCC 2018, page 44–48, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450365536. doi: 10.1145/3301326.3301374. URL https://doi.org/10.1145/3301326.3301374.