Playing the Game of Hive: an Imitation Learning Approach

John Strachan McKenzie

28 August 2023

Contents

Abstract

felipe: Key points to make here. You need four sentences (give or take a couple if you need to make multiple arguments for each of the points below.). Try to avoid overly specialised jargon, e.g., talking about search algorithms and heuristics is OK, discussing the γ parameter of the temporal difference update in reinforcement learning is not.

- What is the problem you address? (automatic play of Hive)
- Why is this important/interesting? (no effective AI player of Hive so far)
- How do you aim to solve this problem? (Monte Carlo / Heuristics?)
- What follows from this? ("Make the world a better place.." err, actually, deeper insights into AI based players, since this game seems to be a challenge to current techniques.)

Acknowledgements

Chapter 1

Introduction

Hive is a two-player board game invented by John Yianni in 2001 [3]. The game was developed as an alternative to chess, with a simplified ruleset, easier to learn than chess and - most importantly - requires no board to play, making it highly portable. The game utilises a series of hexagonal tiles, with each flat edge representing a liberty to which other tiles (pieces) may moved or placed, with some pieces being permitted to stack atop others. Hive was further expanded from its original form to include three additional pieces, each adding a significant new complexity to the game and allowing for further play styles.

Hive, much like many strategy games of its nature, has an enthusiastic, sizeable community of players whom take part in frequent competitions, both online and in-person. Due to the novelty of Hive and how recent the game is compared to other competitive board games, the community of players is smaller than chess or go, meaning significantly fewer resources for the development of computer Hive (and AI therein) are available, and few studies into the game currently exist. At the time of writing, a google scholar search for 'hive board game' will return fewer than 10 unique papers, relevant to the game.

Conventional studies into the game of hive are limited to several approaches, largely focusing on Reinforcement Learning and Tree Search methods. Whilst all documented attempts are good, being based on solid research, the efficacy of their outputs are limited, with no attempt yet being able to challenge even a moderately competent player at the game, and all agents struggling to consistently beat a random move generator.

These studies also reveal much about the game of hive, demonstrating that its

hardness is underestimated, providing a challenging environment for agents to perform well due to the large search space and branching factor providing unique challenges perhaps as yet unmet within the field of game playing AI models chiefly due to its ruleset creating an inherently reward-sparse environment.

Lack of range of approaches leave much room for greater experimentation and novel research methods. One such field as yet unexplored in Hive is the novel approach of Imitation Learning, which is explored at length as a possibility for remedying some of the reward sparsity through having agents use proximity to expert human policy as a measure of progress and efficacy. Though unlikely to generate the best sequence of moves, this measure may provide an agent good enough to provide a challenging game of hive.

Due to the novelty of both the field of computer hive and imitation learning, a lack of existing infrastructure for the development of computer hive, and a series of technical issues arising from a mixture of these factors a pivot in approach from a pure implementation of an imitation agent in hive, to providing a series of tools, and some additional testing, leading to new discoveries about the nature and complexity inherent to the game of hive.

Whilst it was not possible to make any real conclusions to the original hypothesis, it was possible to generate a series of groundbreaking standards in the field of computer hive, namely a Stable Baselines3 Gymnasium environment for hive, and several respective state space representation systems for generating, validating and processing game states, as well as a series of tools which convert human gameplay string files to a series of expert policy rollouts essential for the development of future human gameplay imitation methods, both in hive and in other games.

We also take time to draw further comparisons with the game of hive and other frequently assessed strategy games of a similar nature, namely chess and go, comparing their branching factor, state space size and respective toolsets used in developing AI agents for these games, as well as drawing further recommendations for the future of AI research within the game of hive itself.

Chapter 2

Background

felipe: The paragraphs below feel like they belong in the introduction, since you are explaining why you talk about this. At this point assume the reader already understands why you are talking about Hive and RL.

Despite hive's popularity, in terms of AI development for Hive, there have been scant few attempts, though those published yield promising and useful results - over a range of approaches - though none have had any success in creating an agent that can best even novice players reliably, and most never manage to consistently beat a random move agent. Despite this, the game of hive produces an interesting challenge to AI due to a number of factors, the chief of which is is high branching factor, which varies wildly over the course of a typical game. Secondly its unique ruleset based entirely on relative moves creates a challenge for creating an efficient search environment, and a useful observation state for a computer without high computational overheads.

Although Reinforcement Learning (RL) approaches have been tried before[1][2], as well as approaches focusing on encouraging Monte-Carlo Tree Search (MCTS) to develop human-like play strategies [5], one approach not yet tried that has the potential to create an expert-level agent with lower training overheads is Imitation Learning (IL), a method which seeks to copy human or expert AI moves to generate a policy, which although isn't guaranteed to be the best policy, seeks to be as close as possible to the expert's policy.

2.1 The Game of Hive

The game of Hive, in full, contains 28 pieces, 14 black and 14 white hexagonal tiles, each with a depiction of an insect on the top surface. 22 pieces (11 each) are from the original or 'base' release of the game, and a further 6 (3 each)

are expansion pieces which are optional, though typically always played for all but beginners, and their inclusion in games is typically mandatory at competition level. Each piece type has its own unique move sequence, much like chess.

2.1.1 Rules

It is impossible to gain a full picture of the game without first understanding the rules. The rules update fairly infrequently, last changing for the inclusion of the pillbug piece in 2016[11]. There are a few basic game rules, plus each piece type having its own specific move sequence.

The game begins with no pieces in play. The player using the white pieces begins by placing one piece in the centre of the playing surface, whereby the player using the black pieces responds by similarly placing one of theirs touching this first piece. After this initial ply, players alternate placing pieces such that no new piece placed touches another player's pieces. No players may move until both queens have been placed. Each player must place their queen on, or before, their fourth move.

Once movement may begin, players may either move a piece or place a piece such that the placed piece only touches pieces of its own colour. Players may only move a piece on the grounds that it does not violate the 'one hive rule', whereby no piece may move if to do so at any stage would render any other piece disconnected from the hive, even if that means temporarily.

add diagram

The game ends when a player's queen is fully surrounded on all edges (not including the top surface of the piece). It is possible to draw a game of hive on two grounds:

1. Both players agree to a stalemate or,
2. The uncommon scenario whereby both queen pieces have a common neighbour as their only free edge, which is filled, surrounding both queens at once.

As for each piece type's move sequence, the 'base' game pieces move as follows:

Ant

x3 pieces The ant may move anywhere around the hive from its position to any other free position, provided the piece itself fits in that space without leaving the playing surface (being picked up).

insert image of pieces here

insert rules' demo image showing movement example

Grasshopper

x3 pieces The grasshopper may move any number of steps over the hive (as if jumping) provided there is a contiguous line of pieces from its origin to its destination, with no gaps. It must do so along the axis of one of its flat edges touching said line of contiguous pieces.

Spider

x2 pieces The spider may move like the ant piece around the hive, but exactly three edges per turn. It may not move more or fewer than three edges. Due to its highly restrictive move sequence, the spider is often considered to be the weakest piece in play, perhaps excepting the queen.

Beetle

x2 pieces The beetle may only move one edge at a time. It may also move one space on top of the hive at a time. Once it has been stacked atop the hive, the piece underneath is unable to move or use any abilities, and for the purposes of placing pieces (described subsequently), counts as the colour of the topmost beetle piece.

Queen Bee

x1 piece The queen bee has the smallest range of movement or ability in the game. It may move exactly one edge per turn, and may not move atop the hive.

The three expansion pieces move as follows:

Mosquito

x1 piece The mosquito is a unique piece in that it has no move sequence of its own. Rather, it copies the move sequence of any piece touching one of its edges. If it previously touched a beetle and has since moved atop the hive, it maintains the movement privileges of the beetle piece until it is no longer atop the hive.

Ladybug

x1 piece The lady bug makes two moves atop the hive then a third move to a free space on the playing surface, again on a free edge of a piece.

Pillbug

x1 piece The pillbug has the same move sequence as the queen, with one key difference: it may move any piece from one of its edges to any of its own free edges, provided this does not break the one hive rule.

- playing strategies
- complexity
- computational challenges

There is only one further move possible; a player may 'pass' if, and only if, there are no moves the player can legally make.

2.1.2 Formal Definition

At present, no formal definition written in a standard game grammar exists for the game of Hive. Moreover, common strategy game grammars can be difficult to implement, and many seek to implement gameplay logic using pseudocode. Despite the novelty of Hive, its full information, pure strategy, turn-based nature makes it an ideal candidate for representing as a formal game. Furthermore, the existence of a grammar for hive (and its competing titles) is useful in the comparison with other such pure-strategy games, both for examining relative hardness and exploring comparable solutions from said games.

A formal definition of hive can be written in a number of different respects depending, for example, on whether one considers there exists an invisible game board consisting of a hexagonal grid on which to play pieces, or maintain your perception of the game to be true to the original specification: the 'board-less board game', and instead consider the only board to be the set of tiles out of the players' hands and in play. This project chooses to model the state space faithfully to the game's ruleset, considering there is no board, only a state containing pieces in play, with their respective liberties as spaces. By using Riggins & McPherson's general game grammar [8] as a guide, it is possible to create a formal definition for hive thus:

felipe

Break this long sentence into two.

felipe

Is this based on Riggins & McPherson (but different), or their own formalization? Make this clear here

Let Hive be a game system $\mathcal{G}$ of a 8 tuple $\mathcal{G} =< \mathcal{P}, \mathcal{A}, \mathcal{S}, \mathcal{U}, \mathcal{L}, \mathcal{Q}, \mathcal{O}, \Omega >$ where:

- $\mathcal{P} = \{black, white\}$ is the set of two players (or agents) which can make actions in the game.

- $\mathcal{S}$ is the set of all possible game states with $\mathcal{S}_0$ representing the initial game state for every game, with no pieces placed.
- $\mathcal{A}$ is the set of all actions (from options $\{move, place\}$) a player can make.
- $\mathcal{U}$ is a function which calculates the utility of a game state to a player $\mathcal{P}$
- $\mathcal{L}$ is a function which yields a set of all legal moves given a player $\mathcal{P}$ and a game state $\mathcal{S}$.
- $\mathcal{Q}_{black,white}$ is the set of all pieces for each agent within the game.
- $\mathcal{O}$ is the set of all outcome states for a game, namely $\{blackwin, draw, whitewin\}$.
- Ω is a function to calculate the final outcome of the game.

johnny

Game state $\mathcal{S}_i$ can be viewed as a recursive function with each piece containing a call to the piece that came before it, with an edge reference (which should be considered another variable) eg $\mathcal{S}_i = Q_{white,grasshopper}(Q_{black}$

An algorithm in this grammar for a typical game would therefore follow:

1. Let the current state be $s' = s_0$ where $s_0 \in \mathcal{S}_0$.
2. **While** s' is not a terminal state ($\Omega(s') \notin \mathcal{O}$) repeat:

   Given player p, let a be a set of all valid actions given $\{a \in A \mid \mathcal{L}_p(s', a)\}$.

   Player p chooses to play a' from some a.

   Compute new game state $s'' = a_i(s')$. Let $s' = s''$.
3. Game ends. Calculate $\mathcal{U}_p(s')$

From this grammar, one point that may become obvious is that the above turn-sequence could be applicable to almost any turn-based pure strategy game. This is interesting for a number of reasons. First of all, this is consistent with current mathematical research, that suggests that any NP-hard computational problem can be represented as any other NP-hard problem. Secondly, it highlights the comparability of hive to other game computation problems more broadly, which allows for the possibility of direct application of successful approaches in such games to hive. Furthermore, it highlights the fundamental simplicity of the turn sequence of hive, with most of the game's complexity coming from its large state space.

2.2 Hive AI

Despite the relative popularity of Hive as a competitive game, Artificial Intelligence solutions herein are extremely novel, and the problem as a whole is sparsely researched. It is perhaps telling that one of the more prominent pieces of research for a Reinforcement learning Approach to hive is a bachelor's thesis from 2013[1], and that a google scholar search for 'hive game ai' at time of writing will yield 6 unique results related to the game of hive, all of which ate cited in this dissertation. Whilst it is important to understand the scarcity of academic research on Hive as a computing problem, this is not the only field currently developing the area of computer hive, with many practical and promising approaches coming from the broader software development community.

2.2.1 Current Approaches

There are at present, a number of notable attempts at developing strong hive agents, which can broadly be split into two categories:

1. Monte-Carlo Tree Search (MCTS) solutions, and
2. Reinforcement Learning (RL) solutions, typically attempting to apply the approach of AlphaZero to the problem of Hive.

Although both approaches trialed have yielded some promising results, they each have their respective drawbacks, with neither approach yet delivering an agent that can beat a competent player.

Monte-Carlo Tree Search

There are currently several working implementations of a game-playing agent for Hive using an MCTS approach, some still actively developed, each using some unique methods for planning moves, all of which being worthy of note.

- **Mzinga Engine**[10] is a simple demonstration agent written by Jon Thysell, initially in 2015 and has been actively been updated ever since. It uses simple Alpha-Beta pruning to determine the value of moves, utilising heuristics such as proximity to a win condition and number of possible moves an opponent may make in a position to calculate utility.
- **Nokamute**[9] is also a simple Alpha-Beta pruning agent like Mzinga Engine, but whose implementation is focused on the speed of computing the utility of positions. It achieves this in a number of ways, the first of which is by being written in rust[6] a machine-compiled

programming language focusing on memory safety and low computational overheads; secondly, it utilises machine concurrency to increase the search space computed over time. This approach comes with a trade-off, however, which is that in order to maintain computational speed, its utility function is extremely simple, much simpler than that of Mzinga.

explain what it lacks compared to mzinga, if anything

- **'Better AI for Hive: Mimicking human game-play strategies'**[5] Is a paper and respective implementation by academics at the University of Amsterdam and Leiden University, and although the name suggests a high relevance to this dissertation, the methodology detailed therein as an attempt to apply human-like play styles using certain heuristics in MCTS.

explain heuristics

Reinforcement Learning

Presently, all RL approaches to solving hive have come from an academic setting. This is hardly surprising due to the computational power typically required to generate a model with the ability to solve difficult computational problems with a high branching factor like hive, which unlike most universities, private persons typically lack the access to.

The noteworthy implementations of RL hive

- **AZHive**[2] is an attempt to implement an Alpha-Zero[**?**] approach to hive using deep reinforcement learning, convolution and extensive self-play.
- **Reinforcement Learning AI to Hive**[1] was an attempt by two Bachelor's degree students to apply a simple reinforcement approach to hive using modest computational resources.

2.2.2 Analysis

Although the above approaches are well researched an implemented, none have been able to best a competent player of hive except for a few limited circumstances[1], with the strongest agents at present being MCTS implementations, developed by enthusiastic community members with a software engineering background.

[1] An example of such a circumstance can be found here `https://www.youtube.com/watch?v=MuQkspg3AY0`. It is worth noting that the agent only managed to overcome the human player here by only allowing the human to play a specific, disadvantaging sequence of placements

The limited efficacy of the Reinforcement Learning agents' lack of efficacy is compounded by several factors; firstly, there is no standardised learning or training environment for hive as with chess - rather, both aforementioned RL projects developed separate approaches to state space implementation. AZHive sought to adapt the published specification for AlphaZero to fit the technical requirements of Hive, whereas the approach by Blixt & Anders implemented a non-standardised 'gym' environment for training purposes. Furthermore, this lack of a standardised approach to training and testing learning agents provides a significant challenge to making meaningful comparisons between methods and implementations, where limiting the number of dependent variables proves essential for drawing effective conclusions.

From reviewing the papers associated with these implementations, a number of relevant facts emerge. The first of which is that Hive has an extraordinarily high branching factor at between 60 and 65 on average[5], with an enormous standard deviation due to the ability of players to limit their opponents' moves greatly during the mid and endgame scenarios. This puts the branching factor of Hive somewhere between chess (≈ 35) and go (≈ 250). In Furthermore, due to the nature of hive, starting with few pieces, and only adding more as the game continues, this leads to the branching factor starting very low and rising exponentially over time - this is in stark contrast to chess, which follows a rough bell-curve, with few scenarios to explore nearer the endgame

cite

cite

cite
https://www.researchgate.net/figure/The-average-branching-factor-fig4_23751565

The second fact worth noting is that hive is a sparse reward environment. Both learning agents and symbolic agents struggle to develop useful metrics to evaluate the utility of a position. This is an issue that human players also share[4], with most players only utilising metrics specific to the endgame scenario such as 'tempo' - the number of turns until you lose subtracted from the number of turns until your opponent loses.

In comparable games such as chess and go, agents frequently make use of two metrics for intermediary goals: 'material' and 'territory'. Material represents the number of the opponents pieces captured, and specifically in chess, the perceived value of removing each piece. Territory represents the number of spaces on the board which an agent controls, or that cannot be utilised by the opposing player in any meaningful way.

Neither material nor territory are applicable to hive. Material is an obvious example, as one never captures a piece in hive, as no tiles are ever removed from play once added to the game. In the case of territory, it is possible to count the number of liberties, possible to place a new piece on, this is not essentially useful - firstly, this approach is only useful so long as there are

pieces to place still, which at the end of a game is unlikely. Secondly, not all liberties were created equally. Placing a spider, for instance, on one's own queen is considered to be a bad move in most cases[4], unlike placing a beetle within moving distance of the opponent's queen or pillbug piece.

As a consequence of the combination of a high search space and sparse intermediary reward opportunities, the game of Hive presents a unique challenge for computation, to which none have yet found an obvious solution; that is not to say there is no way to remedy the lack of intermediate reward, however. One solution, not yet tried that may provide a solution to this reward sparsity is imitation learning, wherein we treat expert moves (expert policy) as a goal in themselves for training an agent, and reward an agent on its proximity to those moves. This, in effect, converts a reward sparse environment into a reward dense one, with agents still learning how to complete an episode as a winner, without getting stuck in a local maxima of picking endless moves.

2.3 Imitation Learning

Imitation Learning (IL) is a relative newcomer in the field of machine learning, and may be seen as a subset or expansion to reinforcement learning[7], though it is worth noting that whilst most methods implement reinforcement techniques not every implementation of imitation learning uses reinforcement. Whereas the optimal goal of reinforcement learning is typically to find the best move given a state, the goal of imitation learning is to pick the move closest to an expert, typically a human, though often other agents. The approach has been applied in any number of fields, from self-driving vehicles to surgery.

cite

cite

2.3.1 A Brief Comparison of Imitation Algorithms

Within imitation learning, there are several algorithms each offering their own approaches, advantages and disadvantages - some being extremely domain-specific. At this point it is important to dissect the behaviours and limitations of these algorithms to determine their applicability to this research problem in order to arrive at the best fit.

- **Behavioural Cloning** (BC) is by far the most simple method for imitation and is most frequently demonstrated when talking about machine learning. In its simplest form, it trains on a key-value pair of states and actions that it has observed from an expert demonstration. In essence,

behavioural cloning methods typically assume that the expert policy demonstrates the optimal action in every possible state. Due to this simplicity, and a lack of an exploration component of any kind, when encountered with a new state, models trained in this way are likely to perform better than random.

In the context of hive, the ability to adapt in new states is important, as even with plenty of exploration, the chances of the agent to have encountered the same state twice is absurdly low after only 10 moves[1]. Furthermore, due to the tight link between demonstrations and output, behavioural cloning models are extremely prone to noise (i.e., adopting an expert's error).

A more advanced example of this algorithm is DAgger, which seeks to correct BC's penchant for adopting noisy inputs by using expert feedback to correct mistakes; this process is both labour and time consuming and assumes that there exists an error whom is omnipresent during training and can spot and correct mistakes, which must be done in real-time.

cite DAgger paper

- **Inverse Policy Learning** (IPL) is essentially RL done in reverse. Instead of an environment handing out rewards to an agent to teach the agent policy, the inverse policy agent observes other agents reward vectors in certain states in order to learn what the policy is, assuming the expert's actions are correct. IPL provides a powerful means to learn complex policies in difficult and novel reward spaces, and can learn how to generalise and imitate complex patterns of behaviour, though will struggle to do so with a limited number of observations. Because of this need for larger obervation sets, training an inverse reinforcement model is typically computationally expensive.

- **Adversarial Inverse Policy Learning** (AIPL) is an improvement on IPL and adds an 'adversarial' component in the form of a discriminator network who attempts to distinguish model-generated outputs from genuine expert ones. . This approach seeks to improve the accuracy and quality of the generated predictions from the model, and improves the diversity and accuracy of the learned policy. In a gym-like environment for training models in two player game environments, such as Hive, this approach helps the model both avoid being stuck in local maxima (as human demonstrators would not get stuck playing the same few legal moves in a loop to extract points) and produce a greater diversity of moves in a given state, accounting for more intricate strategies used by expert players.

cite AIPL paper

2.3.2 Comparable Approaches from Other Games

- Maia is an imitation learning neural chess engine which aims to make the most human move at a set ELO rating, rather than making the best possible move. Reasons for this approach to be expanded on obvs. Outcome is an agent that plays a lot like a human player - very good for training humans with, as well as a baseline for whether or not a move is 'human' as opposed to agreeing with stockfish etc.
- have there been approaches like this for GO? I think there have been. AlphaGo is famous but deliberately doesn't make human moves. I need to make more points on this.

Chapter 3

Methodology

In this chapter, we demonstrate hte methodology used in this project, which aimed to improve development and understanding of AI methodologies for the game of Hive, and of imitation learning when applied to NP hard games. Despite encountering many unforeseen challenges and limitations, this project created great steps towards standardising the training of Hive AI agents, and creating tools for the conversion of expert games into game policies. This project initially intended to use Adversarial Inverse Policy Learning (AIPL) techniques to have AI agents to learn from expert Hive moves using the 'imitation' library for Python. However, compatibility issues between the 'imitation' library and the latest version of Stable Baselines3's Gymnasium posed significant obstacles. Despite these setbacks, the study's contributions to the small field of computer Hive are significant, particularly in the creation of a Gym environment tailored to Hive AI training, including movement validity testing etc. Further to this, we elaborate on specific methodologies applied, libraries used and created, and the strategies used to overcome the challenges and generate significant advancement into computer Hive.

3.1 Research Approach

The original research approach aimed to employ Adversarial Inverse Policy Learning (AIPL) techniques for the development of a Hive game agent capable of playing a game strong enough to give a good player a challenging game. The central methodology underlying this approach was the utilisation of AIPL to create agents which learn from competition-level Hive players' moves.

Training an AIPL agent in Hive would require using an extensive dataset of expert Hive moves, generated through human gameplay, from expert competition level matches. These expert moves would serve as the basis for training AI agents, allowing them to mimic the strategy, decision-making and tricks employed at a competitive level. This approach has the potential to yield an agent that is capable of making difficult and human-level decisions in the course of a game.

However, the realisation of this approach encountered a significant issue in the form of compatibility issues between the 'imitation' library for Python and the current version of Stable Baselines3's Gymnasium environment, which was developed specifically for this approach. These compatibility issues interrupted the original plan and necessitated a pivot in the research approach towards creating the toolset required for implementing IL for computer hive, in anticipation of the imitation library becoming more stable and contemporary. The subsequent sections of this methodology detail the revised research approach and the adaptations made in response to these challenges.

3.1.1 Data

-discuss datasets, how generated, how parsed etc I'll come back here

generate list of stats on the dataset used: number of players, in the set, number of games within, competition games, average game length etc.

3.2 Revised Implementation

Whilst the original research approach aimed to employ Adversarial Inverse Policy Learning (AIPL) for training Hive-playing AI agents, several challenges necessitated a significant pivot in the research methodology. The initial plan was to harness AIPL techniques to enable agents to learn from expert-level Hive moves, with the goal of creating highly skilled agents capable of emulating human-like strategic decision-making in the game.

The revised research approach centred around the development of a novel Gym environment tailored specifically for training RL and IL agents in Hive gameplay. This custom environment serves as a crucial, first of its kind contribution to the field of computer Hive, as it provides a standardised platform for training and evaluating Hive-playing agents. The environment incorporates game rules, two state representation systems, movement validation and an action space essential for model development, helping to overcome some of the current problems that hinder computer hive development.

The project also sought to meet technical specifications and compatibility with existing standards for computer Hive - namely 'Universal Hive protocol',

a standard for having hive agents communicate with other agents, with human players, and with viewer interfaces, for which two existing agents are compatible.

Furthermore, the revised approach involved the processing and utilisation of a large expert dataset generated at competition level, converting games into expert policy rollouts compatible with the 'imitation' library's expert policy rollout parsing methods, which will enable human policy learning for AI agents in future, not only for Hive, but for other pure strategy games of this nature also.

These implementations in spite of the challenges faced seek to underscore the research commitment to advancing computer Hive and imitation learning for strategy games as a whole, to enable hive to be attempted as a computing problem on equal footing with games such as Chess and Go.

3.2.1 Dependencies

Over the course of development, several libraries and frameworks were utilised. These dependencies encompass a set of tools that have been incorporated to enable aspects of the research and development process and improve overall functionality and further the research objectives.

The first library worth mentioning is - in fact - the first dependency inherited by the research. Namely the '**hive-ai**' library from github user 'yoderw'. As the library does not implement an installable package, and is not complete for the purposes of this research the project was simply forked and forms the basis of the source code generated in the implementation of this project. Furthermore, the project at time of writing was last updated in February of 2015 and was written in an early version of python 3, as well as having several failing unit tests. As such, extensive work had to be carried out to make the library functional, discussed subsequently in this section.

cite & include footnote about it also being a fork etc

The next dependency used extensively in this project is the Stable Baseline3 (SB3) library, a framework central to the implementation and development of this research. SB3 implements a framework for creating and testing both standardised agents and environments for the development of RL models, and in our case, IL models. It is extremely reliable, well tested and extensively documented, implementing a number of pre-written algorithms such as Deep-Q Learning Networks (DQNs), Proximal Policy Optimisations (PPOs) etc. It also implements the interface standard for gymnasium environments, the aforementioned environments used for training and testing models - a standard critical to the output of this research.

The final dependency of note is the '**imitation**' library for python - itself something of a fork of the Stable Baselines3 project. It implements several algorithms within its framework, including the aforementioned Adversarial Inverse Policy Learning algorithm, and several methods of basic Behavioural Cloning. Further to this, it also implements a framework for generating observations from 'expert' AI models, from which represent expert policies, all within a 'gym' environment. These are then in-turn used to create expert 'rollouts', a series of expert trajectories (actions given a state) from which imitation learning agents may learn. This is only true, at present, for gymnasium-generated trajectories and a method for turning human moves in games into expert trajectories had to be developed as part of this research.

3.2.2 Gymnasium

As previously demonstrated, the current state of computer Hive is fragmented, niche and disparate. Up until now, there was no standard environment or format for hive state space representation nor action space representation - especially for the purposes of training and testing; that is until now. One of the key outputs of this project has been to create a gymnasium environment to standardise the training and testing of learning models in this space, including providing the means to provide objective comparisons to other methodologies applied to the same problem.

Observation Space Implementation

Hive has no board. This is a key pillar of the game's design. It is designed this way such that it can be played anywhere. AI agents typically need a numerical representation of a game state such that they might make informed actions. Typically, a grid environment would be employed here for representing the game state, however this representation is too naïve for computing various aspects of the game logic. Hive is very much a 3-dimensional game with its ruleset very much wedded to real-world physics, for example deciding if pieces may move through a space is decided on whether or not it physically fits.

In the aforementioned hive python library, the game state is represented in a dict, with a list of python game tile objects acting as a value, and a set of coordinates representing the key. The coordinates in question are Cartesian in nature, starting at $(0, 0)$ and extending infinitely in both positive and negative axes. For each value in the dict, of type 'list', the piece at index -1 is always the upper-most tile (that is to say the one visibly on top of the stack).

mention how representation of coords is done

Although this implementation is both fast and programatic, lending itself to extensibility, it is much too complex a representation for a learning agent to interpret. Furthermore, this representation is tightly linked to python builtin objects and types, making training in other languages impossible. A compromise observation state, therefore, is needed.

Converting a 3D game state to a 2D one is in itself a compromise, though one that might be done effectively enough such that a learning agent will not suffer from a decrease in useful information. As such, the design chosen settles on a 28×28 grid, representing the max number of pieces in both x and y coordinates.[1] Further to this, only the piece at the -1 position (the top) of each hexagon will be represented in the grid state, re-appearing if/when the top piece moves away.

The coordinates stored in the 'hive' object are converted from their Cartesian state to 2D coordinates required by a numpy array by adding the (x, y) to the centre point index of the 3D array at $(13, 13)$, the results of which represent the absolute coordinates, where $(0, 0)$ represents the first (upper-left-most) index in the array.

The pieces in question are represented in the 2D array by a series of integers, ranging from 11 to 180 where the units column represents piece number, tens column is piece type, and hundreds column is the piece colour (+100 for black, +0 for white). The method for +100 for black pieces Initially the pieces were to be unsigned 8 bit integers to save on memory allocation, though Stable Baselines3 does not support unsigned integers in observation spaces.

finish this algo

compare this action space to chess and talk about how this results in a sparse grid. calculate how sparse this grid is.

Action Space Implementation

The action space developed implements a discrete action space. Discrete action spaces in a Stable Baselines3 gym environment is a range of integers from one to n, with each integer in sequence representing a possible action. Initially, an action space of $28^2 \times 28^2$ was designed, representing a move from any (x, y) to any other (x', y'). This resulted in an action space comprising

[1] It is technically possible in this instance that the board state may extend beyond this limit, as the game starts at the centre of the board using Cartesian coordinates; it is therefore possible if each piece were to extend in one direction in sequence. This, however is an extreme outside possibility and not possible in a typical game. In over 15,000 parsed games, not one ever exited this game state

of 614,656 moves, an absurd number, and a typical run of millions of steps would arrive at only a few valid moves, not to mention good moves.

It was possible to slim this number down significantly however, to a mere 10,977 moves. This was done by replacing the initial (x, y) space to 14, possible moves, each representing one piece in the current player's hand, plus one extra move, accounting for the player to pass their move if and only if they have no legal moves. This figure is still problematically high however, as the chance of an agent making a valid opening move is $\frac{1}{784}$, though there is little way to condense this action space further without limiting the agent's ability to actually play a game of hive. This action space seems larger still when compared to both a chess gym environment, where the action space is a significantly smaller 4,101 possible actions and a go environment utilising a mere 362 possible actions, making the action space for a hive gym uniquely broad.

cite chess gym env

cite go gym

3.2.3 Reward Space

The reward space in the Hive gym environment is fundamental to shaping the learning processes of learning agents engaged in Hive gameplay. Designed with consideration of the game's extreme action and observation sparsity, and with the game's objectives, this reward structure serves as a loose guiding principle to steer agent behaviour. The components of the reward space in the base case are implemented thus:

- `TRUE_ACTION_REWARD`: This reward component incentivizes the agent to take moves that are legal within the rules of the game, regardless of how advantageous the move is. It yields a reward of 1.
- `FALSE_ACTION_REWARD`: This reward disincentivizes the agent from making moves that are not legal and is an obvious penalty. It yields a reward of -1.
- `GAME_WIN_REWARD`: The reward total for this state is scaled to be significantly higher than the other rewards to indicate a magnitude for winning, which may be useful in avoiding having an agent stuck in a local maxima.
- `GAME_LOSS_REWARD`: In the exact opposite scenario to the above, this state seeks to encourage the training agent to protect one's own queen from attack by using the same magnitude to disincentivize losing a game.
- `GAME_DRAW_REWARD`: The nature of the game of hive occasionally leads

to draws. Though rare, these are considered to be much favourable to losing and therefore ending the game in this state is rewarded, such that agents may choose this outcome over a loss.

It is worth note that there is no reward associated specifically with attacking the opponent's queen, nor liberating spaces around one's own queen. The reason for this is twofold:

1. In a paper which sought to imitate human decisions using MCTS[5], researchers discovered that rewarding agents for covering the opposing queen's liberties would result in a the algorithm being unable to defend, naïvely throwing pieces at the opponent's queen rather than stopping opponent's pieces who may attack the agent's own queen, even in scenarios where the agent was one move from defeat.

2. Contemporary competitive strategy for hive typically asserts that surrounding the opponent's queen is not the object of the game, but merely how one wins[4]; that is to say, attacking the opposing queen is of secondary importance to the game as a whole, and can only be done when one can be assured that doing so won't lose the game. It is therefore not a rewardable strategy in and of itself.

Expert Rollouts and Reward Adjustments

In the context of expert rollouts, a notable modification was made to the reward space: the removal of `TRUE_ACTION_REWARD` and `FALSE_ACTION_REWARD` from expert rollouts. This is due to the inherent nature of competition game notations; in such notations, actions always legal, as illegal moves cannot be made as the game software generating moves will not allow them to be made, rendering the evaluation of the legality of expert moves redundant. Therefore, these reward components were removed from the expert rollout reward structure to optimise learning from expert policies effectively, as seeing reward for every move becomes meaningless.

Step Sequence

Over the course of play, the environment makes a number of complicated calculations at each step, which are worthy of further discussion. Whereas the agent picks an integer in the action space and stores the observations resulting, making assumptions about the correlation between action and subsequent observation, the environment must complete a complex sequence of calculations to validate the action and generate the resulting output. The procedure in plain text is as follows:

johnny

I need to remind the markers about how hive uses relative placing, the board uses absolutes and I had to convert one to the other

1. Get `ACTION` from agent.
2. If `ACTION` in set of all legal moves:
    1. `REWARD` $+1$.
    2. Convert *action* to move tuple of format (`PIECE_ID`, (x, y)).
    3. Convert `PIECE_ID` to respective python object
    4. If piece on board, get coordinates of piece, and move from location to (x, y)
    5. Else, place python object at coordinates (x, y)
    6. `OBSERVATION_STATE` = Convert Hive object pieces dict to 2D numpy array.
    7. If terminal state (Ω) reached:
        - If Agent is the winner: `REWARD` $+100$.
        - If Agent is the loser: `REWARD` -100.
        - If game is drawn: `REWARD` $+50$.
        - `TERM` $= True$.
    8. Else, `TERM` $= False$.
3. Else:
    1. *Reward* -1.
    2. Since no update to game state, `TERM` $= False$.
4. Return (`OBSERVATION_STATE`, `REWARD`, `TERM`, `TRUNC`)

Need to implement trunc method

3.3 Universal Hive Protocol

Over the course of implementation, efforts were taken to include interoperability with the only-existing computer hive standards: Universal Hive Protocol (UHP). This standard was created by Jon Thysell in 2017 as a means of implementing a computer chess style of protocol to the game of Hive, and is documented in a GitHub repository[2]. Universal Hive Protocol implements two standards: UHP Engines - representing hive engines or players, and UHP

[2] `https://github.com/jonthysell/Mzinga/wiki/UniversalHiveProtocol`

Viewers - describing interfaces that interpret engine moves and provide an interface that humans may view or play in games.

UHP operates by sending and receiving strings of text in a REPL (Read, Evaluate, Print loop), using a set of fixed commands with limited arguments.

UHP Engines

A UHP engine represents a game-playing agent, which implements the standardised protocol. The engine has the most responsibilities in the two-part system. As per its responsibilities, the UHP Engine must:

- Validate moves and return an error state where an invalid move is played.
- Store the current game state.
- Know whose turn it is currently.
- Be able to produce the entire game's move set up until the present moment.
- Implement the ability to undo or redo a move.
- Recognise an end-game state, and report win/loss/draw condition per player

UHP Viewers

A UHP viewer represents an interface between two players, agent or human. It provides a graphical interface that human users may interact with, and displays critical information about the game, such as current game state, whose turn it is, and alerts users to erroneous actions. At present, only one such viewer has been implemented; namely, MzingaViewer.

An exhaustive list of the mandatory responsibilities of a UHP viewer is as follows:

- Render a graphical representation of the game state
- Display the pieces in both players' hands
- Display whose turn it is
- Display the movement history
- Visually declare that the game is in a terminal state, if it is so.

Although the implementation of this project did provide some rudimentary support for interfacing machine learning agents with other UHP engines, the output from Stable Baselines3 agents is somewhat different to that of a typical ML model, and would require an interface layer to convert the action state to the tuple required by the currently written UHP layer. Due to the lack of a working implementation of a competent hive agent from these methods, however, the need for testing versus other agents is, at present, irrelevant. The UHP implementation provided, therefore, is only slightly relevant to the outcome of this research.

Chapter 4

Results

Here the relevant outcomes from experiments conducted within the context of the Hive gymnasium environment are presented. Whilst the analysis represents a valuable step, both in the evaluation of this approach, and toward the development of computer hive, it is important to acknowledge the constraints of the experiments and their subsequent results. Specifically, due to resource limitations and the expansive nature of the Hive gameplay environment, only one full run of 1,000,000 steps could be achieved in the timescale of the project, yielding a limited dataset. Consequently, these results represent a basic exploration of the training process within this gym environment.

In subsequent sections, the shortcomings and limitations of this study are presented in greater detail, their causes and contributions discussed relative to their impact on the outcomes and approaches of the study as a whole.

In spite of these short comings, these results still provide a valuable insight into the learning process in the environment though the statistics generated. Subsequently, we discuss in-depth the results and draw limited conclusions on what evidence is available, and further examine which future steps may be taken to improve upon this environment.

4.1 Training

4.1.1 Hardaware

The experiment in question was carried out on a 2015 Lenovo Thinkpad X1 Carbon, with an Intel Core i5 vPro processor at a clock speed of 2.7GHz, and has 8GB of DDR3 Memory. Further, the computer has an Intel HD

5500 integrated graphics chip, however PyTorch is unable to make use of this hardware for the purposes of acceleration.

This use of hardware represents a method of last resort due to the only two instances of compute-as-a-service possible within the financial bounds of this project being temporarily out of service during the time of running (near the end of the project), or because of software compatibility issues involving private repositories, as is described in fuller detail in the discussion of this dissertation.

4.1.2 Agent

The model chosen to provide a test case for the Hive gymnasium environment is Stable Baselines3's Deep Q-Network (DQN) model, chosen due to its simplicity as a model. A DQN is a rudimentary learning model, especially when compared to more recent approaches, implementing a basic and inefficient recall function, requiring large training sets or many environmental interactions to be able to learn enough to make accurate predictions. Furthermore, its exploration strategy is extremely limiting, focusing on short term reward with only random exploration subsequently, making it prone to local maxima, and its performance is tightly linked to its epsilon (ϵ) and gamma (γ) values, which require much experimentation in each case to find an optimum specific to each environment and test case. This inefficiency was considered a valuable requirement in order to test how poor models may cope in the environment, and to have a good worst-case baseline to compare future models to. Furthermore, because of its rudimentary specifications, tends to have a slightly higher learning rate than its newer counterparts, especially on older hardware without adequate graphical accelerations.

The DQN model used from Stable Baselines3 was initiated with the default settings. There were several reasons for this, chiefly of which was the issue of replicability. Using the default settings as a base case would ensure greater replicability of the experiments. It establishes a standardised starting point that other researchers can easily replicate, making it a reliable benchmark for comparison. Furthermore, default settings are typically designed to provide a balanced and generalised performance across a wide range of tasks. This choice not only simplifies the initial setup but also ensures that any enhancements or modifications to the model can be built upon an obvious baseline, making the research more transparent and easier to interpret.

The parameters default to the DQN model are the following:

- Exploration rate (ϵ): 0001

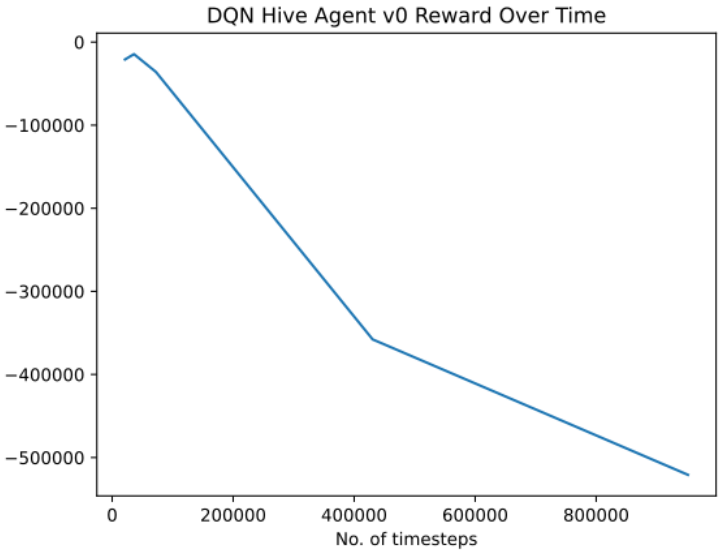
- Gamma (γ): 0.99
- Buffer size: 1000000
- Batch size: 32
- Learning starts: 50000

Furthermore, the default agent has no replay buffer class.

rewrite this if time allows, otherwise JUST DELETE

4.2 Performance

The model was initialised as described above and was allowed to run within the environment for 1,000,000 iterations. On the hardware listed, this took a little over 9 hours. Over the course of the run, the agent completed a mere 5 games (episodes), with a loss value of 6.5e+03, and had a final reward value below -500,000.



The statistical results from this run are as follows:

```
---------------------------------
| rollout/            |         |
|    ep_len_mean      | 1.9e+05 |
|    ep_rew_mean      | -1.9e+05 |
|    exploration_rate | 0.05    |
| time/               |         |
```

```
|    episodes         | 5        |
|    fps              | 28       |
|    time_elapsed     | 33016    |
|    total_timesteps  | 951684   |
| train/              |          |
|    learning_rate    | 0.0001   |
|    loss             | 6.5e+03  |
|    n_updates        | 225420   |
-----------------------------------
```

Chapter 5

Discussion

before moving onto conclusion, make sure hypothesis is clear and obvious throughout!

5.1 Findings

5.1.1 Hive Complexity

Throughout the this dissertation, we make consistent comparisons to the games of Chess and Go, and their respective computer implementations. We further make specific reference to their assigned branching factor, a habit shared by other papers on computer hive. Nevertheless, the implementation of project shows that branching factor is only one small part of the story of complexity. Our approach shows that the observation space for hive is orders of magnitude greater than that of chess, and more than double that of go, and the action space is again, vastly larger than the aforementioned games. This demonstrates one of the many facets of hive that prove hive is a challenging computational environment, and will likely remain so for years to come.

5.1.2 DQN Experiment

With regards to the implementation of the Deep Q-Network (DQN) agent, which was used to test and explore the created Hive environment, the outcomes of this experiment have yielded valuable insights that merit extensive consideration. As the agent's training progressed, it became evident that the action space of Hive is vast - too large, in fact, for a simple agent such as this to make valid generalisations about the possible move set. Notably, the average episode length reached roughly 190,000 iterations, further highlighting the vastness and complexity of the game's state space. Further to

this, the corresponding average episode reward demonstrated a disappointing trend, showing that the DQN agent struggled to make meaningful progress within the gymnasium environment. With a final reward of approximately -550,000, the agent's performance seemed to mirror the challenges associated with Hive's computational aspects. Furthermore, these results may demonstrate the need for a non-punitive reward network, as the results seem to suggest the agent spent much of its time trying to minimise its loss - understandable when considering that the chances of randomly arriving at a correct move are, in the best case, $\frac{1}{56}$ (simply put, even in the best of circumstances, for ever 1 point of reward the agent receives, it will receive an average of 56 points of penalty). Subsequently, the results, and the agent's inability to learn much of the game of hive, provide evidence for our hypothesis that even with valid move rewards, the game of hive is an extremely sparse reward environment, with the agent going hundreds of iterations on average without reward.

While the DQN agent's performance may not have yielded significant breakthroughs, the results are indicative of potential for future advancements in computer Hive. This experiment lays a solid foundation for future research endeavours, setting the stage for the development of innovative algorithms and strategies that can successfully navigate the game of Hive. Additionally, these findings highlight the need for tailored approaches that can address the unique challenges posed by Hive's extensive state space, some suggestions for this documented further below.

johnny

Perhaps our approach might have worked better had we simply done it in thing with existing implementations such as chess or go.

Despite the significant contributions made in this study, it is essential to acknowledge certain shortcomings that prevented a comprehensive evaluation of our hypotheses - these shortcomings are acknowledged further in this discussion. In spite of these shortcomings, this research represents a large step in the field of computer Hive, having developed a greater understanding of the games complexities overall. Whilst it was not possible to test the initial hypothesis, the outcomes from a revised methodology yield a significant body of progress for Hive as a modern computing problem.

5.2 Contributions

Although the findings of this dissertation are slight due to the many limitations of the approach attempted, the outputs from the project represent a significant contribution to the field of computer hive. Whereas the following methods all exist for typical computer implementations of pure strategy games, the novelty of hive meant that none existed before now, and enable

a standardised development of machine learning in the game of hive.

this sentence makes no sense fam

5.2.1 Gymnasium

Foremost of these contributions is the advent of a Stable Baselines3 Gymnasium Environment for Hive, which provides developers and researchers a standardised environment for training, testing and evaluation of machine learning approaches to the game, and promotes future development of state-of-the-art methods in the context of Hive, opening the door further to future development in this space.

Further to this point, the observation space and action space developed for this implementation go some way to prove the hardness of hive in relation to the games of Chess and Go, showing that further research into the hardness of the game, looking far beyond branching factor, before further improvements in computer hive can be made.

5.2.2 State Space Representation & Validation

This project undertook the task of developing a comprehensive game validation system and state space representation for the game of Hive which is both complex and heavily wedded as a rule set to the physical constraints of the real world. Inheriting and improving upon existing libraries and projects, this project refined validation mechanisms from previously abandoned open source computer hive projects. Said validation system plays a comprehensive role in ensuring the legality of moves during AI training and gameplay. It provides foundations for assessing the quality of AI model generated moves, promoting more precise learning processes.

In addition to validation, this research introduced a means to convert complex object-based state space representations to efficient integer-based grid representation of game states. This representation is not only compatible with neural networks but also significantly lightens the computational overheads of processing the game. This innovation helps agents to efficiently process game states, ultimately enhancing their overall performance. Further, it bridges the gap between Hive's unconventional hexagonal grid structure and the neural network architectures commonly employed in AI research.

5.2.3 Relative to Absolute Positioning Conversions

Due to the boardless nature of the game of Hive, all positions in the game are noted based on their position relative to another piece, with the exception of

the first piece, which is noted in isolation. This is in stark contrast to Chess and Go, where all positions are marked with grid coordinates (e.g., F5 etc.). The nature of computer state representations of games, including hive, Hive means that an absolute coordinate system for the purposes of moves and placements is absolutely essential,

5.2.4 Notation Files to Game State Representation Conversions

Another key contribution of this work resides in the development of a system for converting gameplay files (written in string based hive notation) into meaningful game state representations. By taking gameplay files as an input, the functions robustly parse the move strings, which are inherently relative, and transform these into game states, providing several helper functions for understanding and manipulating the current board state. This conversion process underpins the work for various subsequent tasks and functionalities within the AI ecosystem developed for Hive.

These subsequent tasks are, of themselves, extremely noteworthy within the context of developments in computer hive. The pipeline, for example, allows human matches to serve as expert policy rollouts for the context of the 'imitation' library for python, and likely for several other imitation learning environments. It also allows for the training of Neural Networks using these games as training policies in supervised learning. Finally, the conversions provide methods to convert said moves into SB3 gymnasium actions and observations.

It is worth noting that with the inclusion of the UHP wrapper class, these conversions can work both ways, although the implementation of conversions in the other direction are poorly tested and less robust, though there are several implementations of UHP engines that may be extended or inherited, rather than meriting further new work.

johnny

I NEED TO MENTION THIS MORE IN METHODOLOGY

Finally, the inclusion of a formal grammar of the game of hive represents a new observation into have as a mathematical and computational problem, and may prove useful to future development into mathematical models of the game in future study.

5.3 Implications

Over the course of this study, we have uncovered several key features. Chiefly of which, is discovering the vast state space of the game of Hive, especially in comparison to its competing game environments such as chess and go.

Further to discovering the complexity and vastness of the state space of hive, this study reveals novel, new attempts at simplifying these complex action and observation spaces, into smaller state spaces, symbolically resolved by the environment. This could have ramifications far beyond hive, and may serve as useful examples for simplifying much many other complex computational environments in future.

Finally, the outcome of this paper has been to create a standardised series of tools for the field AI development for computer Hive. Specifically, we have created a standardised environment for the training and evaluation of agents in the game, and expanded on this with a further toolset that enables the conversion of state space representations from formats amiable to gym environments, to those of other implementations. Furthermore, we have created a set of utilities that can parse and translate human player games (with their respective relative text movements) into accurate state representations, which will go some way to promoting the development of learning agents in this space.

5.4 Limitations

As stated throughout, this project encountered many roadblocks during research and implementation that severely hindered the ability of this study to answer its key research question: that of whether Imitation Learning would provide a better game playing agent in the game of hive, with copying expert (human) moves representing a good intermediate reward state for training machine learning models, in lieu of the otherwise sparse reward environment Hive would otherwise offer. The reasons for this are multiple, spanning many aspects of both research and implementation, as will be discussed further below. Due to these limitations, it is currently impossible to say whether the human imitation approach can yield any results for the development of hive-playing AI, let alone ones better than existing implementations. In spite of this however, this study pivots to providing research in state space size and implementation strategies for future developments.

5.4.1 Lack of Previous Research

The novelty of research in the game of hive cannot be understated, with fewer than 10 published papers on hive as a computing or mathematical problem, most of which originate from two universities, the University of Leiden and the University of Amsterdam; the papers in question being student-led. In comparison to other strategy games, this is an incredibly niche research space with little-to-no previous work on which to base subsequent study. Additionally, few - if any - of these papers provide promising results in the creation of machine learning models in hive, owing to the computational time required to train a model on a search space as large as the one presented by hive[2], and a lack of further academic interest in hive when compared to go and chess causing a scarcity in the resources available to solve the problem by comparison.

5.4.2 Lack of Existing Infrastructure for Basis of Study

On this particular point, it is important to once again draw a comparison to go or chess; when compared to these games, hive has essentially no existing AI infrastructure due directly to its novelty and recency. Whereas a simple google search will yield a wealth of resources for training AI in chess such as gym environments, adversarial gym environments movement parsers etc., none of these tools existed before now for the game of hive. As such, significant efforts had to be made in the implementation of these tools for the development of AI in hive, which resulted in much less time dedicated to the implementation and development of the AI models in question. Furthermore, with no standardisation on how to represent the game space (again, unlike the aforementioned go and chess), significant efforts had to be taken to develop this space into a standardised representation.

5.4.3 Time Constraints

Due to the extensive precursory work that was required in order to carry out the most rudimentary of experiments in this study, the scope of this project was much too ambitious for the time allotted (a little over 2 months). As such, this work is only part-complete and would require further study in order to test the initial hypothesis.

5.4.4 Limited Access to Required Resources

The singular test run of a DQN model in the gymnasium environment took place on an under-powered laptop with no hardware acceleration, which was a platform of last resort. This was caused by two main factors, both of which would have both been accounted for:

1. The High-Performance Computing for Research platform (Maxwell) was down during a large portion of the project's timeframe where a test run would have been possible. Owing to the time taken to develop a working iteration of the gymnasium environment, little time remained to make test runs on the high performance computing environment provided to students by the university.

2. Subsequently, efforts were made to use Google's Colab platform as a backup option, though this prove impossible too, due to frequent versioning and compatibility issues within python, causing runtime errors during execution. Further knock-on consequences included financial costs to the author associated with running such a computationally expensive simulation on a paid compute-as-a-service platform such as Google Colab, which eventually became impractical after several failed runs.

These issues meant that further research and experimentation with the gymnasium environment, and with subsequent model types, was not possible within the timeframe of the project.

5.4.5 Other Difficulties

A final stumbling block worth noting here is that of the 'imitation' library for python itself. Compatibility issues with the current version of Stable Baselines3's gymnasium framework were encountered, and these issues were largely undocumented in the library's resources. This incompatibility significantly hindered the project's ability to provide a demonstration run of the previously mentioned human-generated expert policy rollouts, further necessitating a change in approach for the development of this study. These issues were difficult, if not impossible to rectify, short of becoming a contributing developer on the library itself. This further speaks to the novelty of imitation learning itself, as there is no other general purpose imitation learning library compatible with Stable Baselines3's gymnasium environments currently being actively developed in python.

5.5 Recommendations & Future Work

5.5.1 Future Developments

The Hive Gymnasium is designed with an eye toward extensibility and future enhancements, including the possibility of creating a true adversarial gym environment supporting multiple agent, and a human-compatible interface for training and testing vs human players. Future versions may improve upon this state space representation somewhat, and include invalid move masking, creating hybrid symbolic/machine learning approach to reducing the overall search space a model would have to compute to interact successfully with the game environment such that it may learn generalisations about strategy in the game by removing the time needed for the agent to learn the most basic rules of the game, which represent the most expensive part of training. This is especially possible when we consider that movement validation and valid move generations are already handled within the software developed as part of this project.

Further to this, improvements to the 'imitation' library for python are essential to be able to continue this research further. This may include actively developing the library, or contributing through the submission of bug reports and pull requests on the project's source repository.

An overall improvement to the gymnasium environment as it stands may include the creation of a true adversarial multi-agent environment as documented in the python 'pettingzoo' framework[1] which would enable the direct self-play of developed agents, and direct evaluation of the competitive abilities of models.

In summary, more time is required on this topic to fully evaluate both the original hypothesis of the study - imitation learning's ability to conquer hive as a reward-sparse learning environment - and to test the efficacy of the observation and action states of the environment developed as a key part of this project

[1] `https://pettingzoo.farama.org/index.html`

Chapter 6

Conclusion

In summary, it was not possible within the confines of this study to determine whether or not imitation learning is a suitable approach for attempting to create a strong game playing agent in the game of Hive, nor if it would be useful in remedying the issues of reward sparsity within said game. This was due in part to a number of features, namely a lack of broad previous research into either imitation learning for games, or into hive as an AI problem on the whole. Further to this, the project was beset by numerous technical issues at key points in the research, hampering a full analysis into the original methodology.

In spite of these shortcomings, this dissertation was still able to provide a number of significant contributions to the field of computer hive as a whole, chiefly of which was the advent of a standardised training environment for hive AI agents through the creation of a gymnasium environment specifically for hive. Further to this, a number of tools were created which allow for the full representation of a game state in the game of hive, simplified until comprehensible by machine learning agents. Finally, we delivered a series of helper methods which allow for the conversion of expert human games into compatible policy observations, all of which is groundbreaking to the field of hive.

Through the development of these tools, this study gained a new and deeper understanding of the complexities of the game, demonstrating that a mere comparative analysis of the branching factor of hive, when compared to other games, is not enough to tell the whole picture. Through implementing an action space and observation space necessary to enable AI agent training, it was discovered that the total state space of hive is vast, greater by some mar-

gin than that of chess or go. Furthermore, we demonstrated some exciting developments in the reduction of the action space representation, reducing it by orders of magnitude - a process which may have further-reaching consequences into other games and search environments.

It is once again important to highlight the limitations of this research, due to limited access to resources such as computing time, and a range of version compatibility issues with existing software, it was not fully possible to evaluate all elements developed to their fullest potential given the subsequent time constraints afforded to this project. Furthermore, the novelty of both imitation learning and computer hive as fields of study means that all work broached in this dissertation are extremely novel and merit extensive further testing.

The work in this dissertation represents a strong foundation for AI development, both IL and RL, and was developed with an eye towards future developments and expansion in this field, namely improvements to existing imitation learning infrastructure in python (which is essential to continued research in this topic), multi-agent extensions to the current gymnasium environment, and subsequent analysis in the complexity of hive compared to its pure strategy counterparts to develop a full mathematical understanding of its hardness.

felipe: The bibliography has a number of items that are not really "citable" stuff, like Github repositories (which should just be footnotes), and websites (same). You should also complete the bibliographic info for items like [1] Is this a book? A paper?

Bibliography

[1] Rikard Blixt and Anders Ye. Reinforcement learning ai to hive, 2013.

[2] Danilo de Goede, Duncan Kampert, and Ana Lucia Varbanescu. The cost of reinforcement learning for game engines: The az-hive case-study. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*, pages 145–152, 2022.

[3] Gen42 Games. About us. `https://www.gen42.com/about-us`, 2022. [Online; accessed 13-April-2023].

[4] R. Ingersoll. *Play Hive Like a Champion, Second Edition: Strategy, Tactics and Commentary*. CreateSpace Independent Publishing Platform, 2014.

[5] Duncan Kampert, Ana-Lucia Varbanescu, Matthias Müller-Brockhausen, and Aske Plaat. Mimicking the human approach in the game of hive. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2021.

[6] Nicholas D Matsakis and Felix S Klock II. The rust language. In *ACM SIGAda Ada Letters*, volume 34, pages 103–104. ACM, 2014.

[7] Dean A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*, 3(1):88–97, 03 1991.

[8] Paul Riggins and David McPherson. Formal game grammar and equivalence. In *2020 IEEE Conference on Games (CoG)*. IEEE, aug 2020.

[9] E Roshan-Eisner. Nokamute. `https://github.com/edre/nokamute`, 2021.

[10] J Thysell. Mzinga. `https://github.com/jonthysell/Mzinga`, 2015.

[11] John Yianni. Hive: A game crawling with possibilities. https://www.gen42.com/download/rules/hive/Hive_English_Rules.pdf, 2016. [Online; accessed 13-April-2023].