Hierarchical Portfolio Search in Prismata

David Churchill, Michael Buro

[1 leave this spacer to make page count accurate]
[2 leave this spacer to make page count accurate]
[3 leave this spacer to make page count accurate]
[4 leave this spacer to make page count accurate]
[5 leave this spacer to make page count accurate]
[6 leave this spacer to make page count accurate]
[7 leave this spacer to make page count accurate]
[8 leave this spacer to make page count accurate]
[9 leave this spacer to make page count accurate]
[10 leave this spacer to make page count accurate]
[11 leave this spacer to make page count accurate]
[12 leave this spacer to make page count accurate]
[13 leave this spacer to make page count accurate]
[14 leave this spacer to make page count accurate]
[15 leave this spacer to make page count accurate]
[16 leave this spacer to make page count accurate]
[17 leave this spacer to make page count accurate]
[18 leave this spacer to make page count accurate]
[19 leave this spacer to make page count accurate]
[20 leave this spacer to make page count accurate]

1 Introduction

Many unique challenges are faced when trying to write an AI system for a modern online strategy game. Players can control groups of tens or even hundreds of units, each with their own unique properties and strategies, making for a gigantic number of possible actions to consider at any given state of the game. Even state of the art search algorithms such as Monte-Carlo Tree Search (MCTS) are unable to cope with such large action spaces, as they typically require the exploration of all possible actions from a given state in a search tree. In addition to the difficulty of dealing with large state and action spaces, other design features must be considered such as varying difficulty settings, robustness to game changes, and single player replay value.

In this article we will discuss the AI system designed for Prismata, the online strategy game developed by Lunarch Studios. For the Prismata AI, a new algorithm called Hierarchical Portfolio Search (HPS) was created which reduces the action space for complex strategy games, which helps deal with all of the challenges listed above. This results in a powerful search-based system capable of producing intelligent actions while using a modular design which is robust to changes in game properties.

2 AI Design Goals

In addition to creating an intelligent AI system for strategy games, other design decisions should also be considered in order to ensure an enjoyable user experience. When designing Prismata, the following design goals were laid out for the AI system:

- **New Player Tutorial**: Strategy games often have complex rules, many different unit types, and a variety of scenarios that the player must adjust to. All of this leads to a steep learning curve. Our primary goal with the Prismata AI was to aid new players as they learned to play the game, so that they would eventually become ready to face other players on the online ladder. This required the creation of several different difficulty settings so that players could continue to be challenged from beginner all the way up to expert play.
- **Single Player Replay Value**: Single player missions in video games are sometimes designed as rule-based sequences of events that players must navigate to achieve some goal. In Prismata, that goal is to destroy all of the enemy units, which can become quite boring if the AI does the same thing every time it encounters similar situations. Our goal was to build a dynamic AI system capable of using a variety of strategies so that it doesn't employ the exact same tactics each time.
- **Robust to Change**: Unlike games in the past which were finalized, shipped, and forgotten about, modern online strategy games are subject to constant design and balance changes. Due to their competitive nature, designers often tweak unit properties and game rules as players find strategies that are too powerful, too weak, or simply not fun to play against. We required an AI system that is able to cope with these changes and not rely on hand-crafted solutions that rely on specific unit properties which would be costly to update and maintain as units continue to change.
- **Intuitive / Modular Design**: Often times when creating a game, the behavior of the AI system, although intelligent, may not fit with the designer's views of how the AI should act. By designing the AI in such a way that its structure is modular and intuitive, designers are better able to understand the capabilities of the AI system and thus can more easily make suggestions on how behaviors should be modified. This leads to a much smoother overall design process than if the AI system was simply viewed as a magic black box by designers.

3 Prismata Gameplay Overview

Before diving into the details of the AI system, we need to understand the characteristics of the game it is playing. Here we will briefly describe the high-level game rules for Prismata.

Prismata is a two-player online strategy game, best described as a hybrid between a real-time strategy game and a collectible card game. Players take turns building resources, using unit abilities, purchasing new units, and attempting to destroy the units of their opponents. Unlike many strategy / card games, there is no hidden information in Prismata – no hands of cards or decks to draw from. Units that players control are globally visible and players can purchase additional units from a shared pool of available units which changes randomly at the start of each game (similar to the board game Dominion [Vaccarino 09]). The rules of Prismata are also deterministic, meaning that there is no possible way for the AI to cheat by magically drawing the right card from the top of the deck, or by getting some good 'luck' when most needed. In game theoretic terms, this makes Prismata a two-player, perfect information, zero-sum, alternating move game. This means that the AI does not need any move history in order to pick its next move – it can act strictly on the visible game state at any time.

Due to these properties, the Prismata AI was designed as a module that is separate from the rest of the game engine, accepts a current game state as input, and as output produces an ordered sequence of actions for the current player to perform. This architecture also gives the developer an option of where to run the AI calculations – a game state could be sent over a network to be calculated (if the game is being run on a platform with limited computational power), or run locally on a user's hardware (as they are in Prismata).

4 Hierarchical Portfolio Search (HPS)

The algorithm that was used to form the basis of the Prismata AI is hierarchical portfolio search [Churchill 15]. HPS was designed to make decisions in games with extremely large state and action spaces, such as strategy games. It is an extension of the portfolio greedy search algorithm, which is a hill climbing algorithm that has been used to guide combat in RTS games [Churchill 13]. The main idea behind these "portfolio-based" search systems is to reduce the branching factor of the game tree by using a portfolio of algorithms to generate a much smaller, yet hopefully intelligent set of actions. These algorithms can range from simple hand-coded heuristics to complex search algorithms. This method is useful in games where a player's decision space can be decomposed into many individual actions. For example in an RTS game in which a player controls an army of units, or in a card game where a player can play a sequence of cards. These decompositions are typically done *tactically*, so that each grouping in the portfolio contains similar actions, such as attacking, defending, etc.

HPS is a bottom-up, two level hierarchical search system which was originally inspired by historical military command structures. The bottom layer consists of the portfolio of algorithms described above, which generate multiple suggestions for each tactical area of the game. At the top layer, all possible combinations of those actions sequences generated by the portfolio are then iterated over by a high-level game tree search technique (such as alpha-beta or MCTS) which makes the final decision on which action sequence to perform. While this method will not produce the truly optimal move on a given turn it does quite well (as we will show in Section 5, below). Furthermore, the original problem may have contained so many action possibilities that deciding among them was intractable.

4.1 Components of HPS

HPS consists of several individual components that are used to form the search system. We define these components as follows:

- **State** s containing all relevant game information
- **Move** $m = \langle a_1, a_2, \ldots, a_k \rangle$, a sequence of Actions a_i
- **Player** function p [$m = p(s)$]
    - Takes as input a State s
    - Performs the Move decision logic

    - Returns Move m generated by p at state s
- **Game** function g [s’ = g(s, p_1, p_2)]
    - Takes as input state s and Player functions p_1, p_2
    - Performs game rules / logic
    - Implements Moves generated by p_1, p_2 until game is over
    - Returns resulting game State s’

These components are the same as those needed for most AI systems which work on abstract games.

In order to fully implement HPS, we will need to define two more key components. The first is a *Partial Player* function. This function is similar to a Player function, but instead of computing a complete turn Move for a player in the game, it computes a partial move associated with a tactical decomposition. For example, in a real-time strategy game if a player controls multiple types of units, a Partial Player may compute moves for only a specific type of unit, or for units on a specific part of the map.

- **Partial Player** function pp [m = pp(s)]
    - Takes as input State s
    - Performs decision logic for a subset of the turn
    - Returns partial Move m to perform at state S

The final component of HPS is the portfolio itself, which is simply a collection of Partial Player functions:

- **Portfolio** P = $\langle pp_1, pp_2, \ldots, pp_n \rangle$

The internal structure of the portfolio will depend on the type of game being played, however it is most useful if the Partial Players are grouped by tactical category or game phase. Iterating over all moves produced by combinations of Partial Players in the portfolio is done by the GenerateChildren procedure in Algorithm 1. Once we have created a portfolio, we can then apply any high level game tree search algorithm to search over all legal move combinations produced by the portfolio.

4.2 Portfolio Creation

An important factor in the success of HPS is the creation of the Portfolio itself, since only actions generated by partial players within the portfolio will be considered by the top level search. Two factors are important when designing the portfolio: the tactical decomposition used to partition the portfolio and the variety of Partial Players contained within each partition.

In Table 1 we can see an example tactical decomposition for the portfolio of partial players in Prismata, which is broken down by game phase. The Defense is the ‘blocking’ phase of the game, and contains partial players that decide in which order to assign blocking units. The Ability phase involves players using the abilities of units to do things such as gather resources or attack the opponent. The Buy phase involves purchasing additional units to grow the player’s army. Finally, the Breach phase involves assigning damage to enemy units in order to kill them. Each of these Partial Players only compute actions which are legal in that phase of the game – so in order to generate a sequence of actions which comprises the entire turn we must concatenate actions produced by one of the Partial Players from each phase.

This 'game phase' decomposition works well for games that can be broken down temporally, however not all games have such abstract notions. Depending on the game you are writing AI for, your decomposition may be different. For example, in a real-time strategy game setting categories may involve different types of units, or a geometric decomposition of units placed in different locations of the map. In strategy card games these categories could be separated by different mechanics such as card drawing, card vs. card combat, or spell casting. It is vital that you include a wide variety of tactical Partial Players so that the high level search algorithm is able to search a wide strategy space, hopefully finding an overall strong move for the turn.

Table 1: A sample Portfolio used in Prismata, organized by tactical game phase.

Defense	**Ability**	**Buy**	**Breach**
Min Cost Loss	Attack All	Buy Attack	Breach Cost
Save Attackers	Leave Block	Buy Defense	Breach Attack
	Don't Attack	Buy Econ	

4.3 State Evaluation

Even with the aid of an action space reducing method such as HPS, games that go on for many turns produce very large game trees which we cannot hope to search to completion. We therefore must employ a heuristic evaluation on the game states at leaf nodes in the search. Evaluation functions vary dramatically from game to game, and usually depend on some domain specific knowledge. For example, early heuristic evaluations for Chess involved assigning points to pieces, such as 1 point for a Pawn and 9 points for a Queen, with a simple player sum difference used as the state evaluation.

These formula-based evaluations have had some success, but they are outperformed by a method known as a *symmetric game playout* [Churchill 15]. The concept behind a symmetric game playout is to assign a simple deterministic rule-based policy to both players in the game, and then play the game out to the end using that policy. Even if the policy is not optimal, the idea is that if both players are following the same policy then the winner of the game is likely to have had an advantage at the original evaluated state. The Game function is used to perform this playout for evaluation in HPS. We can see a full example of the HPS system using NegaMax as the top level search in Listing 1.

4.4 HPS Algorithm

Now that we have discussed all of the components of HPS, we can see a sample implementation of HPS in Listing 1, which uses the NegaMax algorithm as the high-level search algorithm. NegaMax is used here for brevity, but could be replaced by any high level search algorithm or learning technique (such as MCTS, alpha-beta, or evolutionary algorithms). The core idea of HPS is not in the specific high level search algorithm that you use choose, but rather in limiting the large action space that is passed in to the search by first generating a reasonable-sized set of candidate moves to consider.

Listing 1: HPS Using Negamax.

```
procedure HPS(State s, Portfolio p)
     return NegaMax(s, p, maxDepth)

procedure GenerateChildren(State s, Portfolio p)
     m[] = empty set
     for all move phases f in s
          m[f] = empty set
          for PartialPlayers pp in p[f]
               m[f].add(pp(s))
     moves[] = crossProduct(m[f] : move phase f)
     return ApplyMovesToState(moves, s)

procedure NegaMax(State s, Portfolio p, Depth d)
     if (d == 0) or s.isTerminal()
          Player e = playout player for evaluation
          return Game(s, e, e).eval()
     children[] = GenerateChildren(s, p)
     bestVal = -infty
     for all c in children
          val = -NegaMax(c, p, d-1)
          bestVal = max(bestVal, val)
     return bestVal
```

4.5 Creating Multiple Difficulty Settings

In most games, it is desirable to have multiple difficulty settings for the AI that players can choose from so that they can learn the game rules and face an opponent of appropriate skill. One of the strengths of HPS is the ease with which different difficulty settings can be created simply by modifying the Partial Players contained in the portfolio, or by modifying the parameters of the high-level search. There are many difficulty settings in Prismata which were all created in this way, they are:

- Master Bot – Uses a Portfolio of 12 Partial Players and does a 3000ms MCTS search within HPS, chosen as a balance between search strength and player wait time
- Expert Bot – Uses the same Portfolio as Master Bot, with a 2-ply Alpha-Beta search, typical execution times are under 100ms.
- Medium Bot – Picks a random move from Master Bot's Portfolio
- Easy Bot – Same as Medium, but with weaker defensive purchasing
- Pacifist Bot – Same as Medium, but never attacks
- Random Bot – All actions taken are randomly

An experiment was performed which played 10,000 games between each difficulty setting pairing, the results of which can be seen in Table 2. The final column shows the average scores of each difficulty setting (100 meaning unbeatable, 0 meaning never wins), from which we can see that the difficulty settings perform in line with their intuitive descriptions. The modular design of HPS allowed us to make slight changes to the portfolio and search settings to create multiple difficulty settings, which satisfied our design goals of creating both a new player tutorial for beginners, and strong opponents for expert players.

Table 2: Results of a 10,000 rounds of round robin between each difficulty setting. Score = win% + (draw% / 2) for row difficulty vs. column difficulty. UCT100 and AB100 refer to UCT (MCTS with UCB-1 action selection) and AlphaBeta each with 100ms think times. Pacifist Bot was omitted since it is designed not to attack and therefore cannot win.

	UCT100	**AB100**	**Expert**	**Medium**	**Easy**	**Random**	**AVG**
UCT100	-	52.1	67.3	96.4	99.7	99.9	83.1
AB100	47.9	-	68.0	94.7	99.5	99.9	82.0
Expert	32.7	32.0	-	90.7	98.9	99.8	70.8
Medium	3.6	5.3	9.3	-	85.9	97.4	40.3
Easy	0.3	0.5	1.1	14.1	-	86.3	20.5
Random	0.1	0.1	0.2	2.6	13.7	-	3.3

5 Evaluation of HPS Playing Strength

To test the strength of the AI system in Prismata in an unbiased fashion, an experiment was run in which the AI secretly played against human players on the ranked Prismata ladder. Prismata's main competitive form of play is the "Ranked" play mode, where players queue for games and are auto-matched with players of similar ranking. Player skill is determined via a ranking system that starts at Tier 1 and progresses by winning games up until Tier 10. Once players reach Tier 10, they then ranked using a numerical system similar to those used in chess.

To test against humans, a custom build of the client was created in which the AI queued for a ranked play match, played the game against whichever human it matched against, and then re-queued once the match was finished. The AI system was given randomized clicking timers in order to minimize the chances that the human players would suspect that they were playing against an AI. The AI used was the hardest difficulty setting, "Master Bot," which used MCTS as its top level search with a think time of 3 seconds. After 48 hours and just over 200 games played, the AI had achieved a rank of Tier 6 with 48% progression toward Tier 7, and stayed at that rank for several hours. This placed the AI's skill level within the top 25% of human players on the Prismata rank ladder, the distribution of which can be seen in Figure 1.

Since this experiment was performed, many improvements have been made to the AI, such as improved tactical decision making in the blocking and breaching phase, an improved playout player, and fixing some obvious blunders that the bot made in its attack phase. Master Bot is estimated to now be at Tier 8 skill level, which is stronger than all but the top 10-15% of human players.

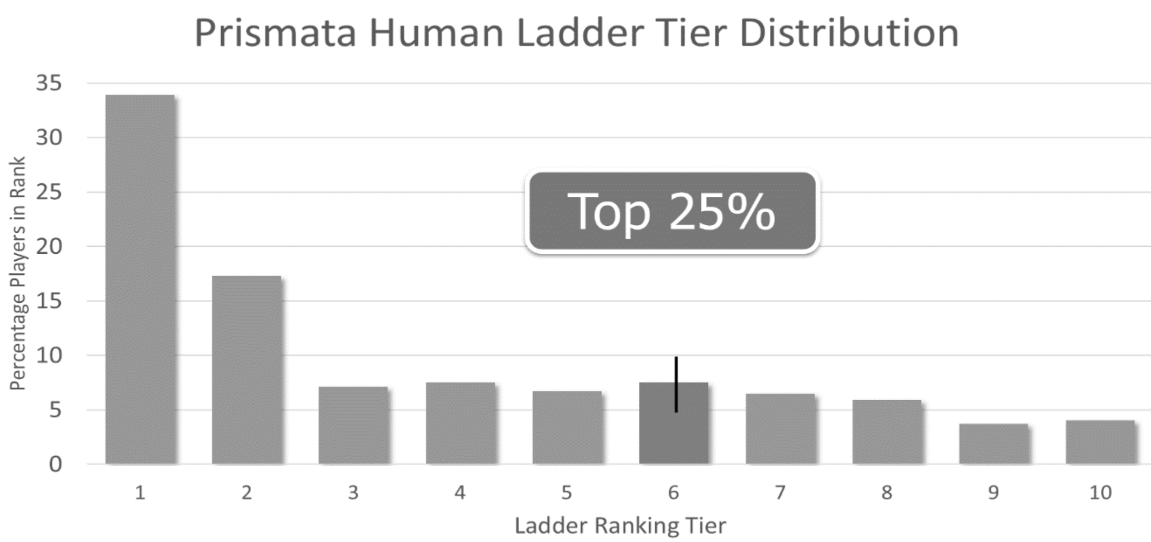

Figure 1: Distribution of player rankings in “Ranked” play mode in Prismata. After 48 hours of testing, Master Bot had achieved a rank of Tier 6 with 48% progress toward Rank 7, which placed its skill level in the top 25% of human players.

6 Conclusion

In this paper we have introduced Hierarchical Portfolio Search (HPS), a new algorithm which was designed to make strong decisions in games with large state and action spaces. HPS has been in use for over two years as the basis of the Prismata AI system, with nearly a million games played vs. human opponents. Because of its modular design, search-based decision making, and intuitive architecture, it has been robust to over 20 game balance patches, producing intelligent actions even with major changes to many of units in Prismata. Creating different difficulty settings using HPS was merely a matter of changing the algorithms in the underlying portfolio, which resulted in a total of 7 different difficulties - from pacifist punching bag to the clever Master Bot. The hardest difficulty of the Prismata AI, Master Bot, was played in secret on the human ranked ladder and achieved a skill within the top 25% of human players, showing that HPS is capable of producing strong moves in a real-world competitive video game.

6 References

[Churchill 13] Churchill, D., Buro, M. 2013. Portfolio Greedy Search and Simulation for Large-Scale Combat in Starcraft. CIG 2013.
[Churchill 15] Churchill, D., Buro, M. 2015. Hierarchical Portfolio Search: Prismata's

Robust AI Architecture for Games with Large Search Spaces. AIIDE 2015.
[Vaccarino 09] Donald X Vaccarino (2009) Dominion. Rio Grande Games.

7 Biography

David Churchill is the Lead AI Programmer for Lunarch Studios on the online strategy game Prismata. He completed his PhD in Computing Science at the University of Alberta in the area of artificial intelligence for video games, specifically in real-time heuristic search techniques for StarCraft. Since 2011 he has been the organizer of the AIIDE Starcraft AI Competition, and won the competition in 2013 with his entry UAlbertaBot.

Michael Buro is a professor in the computing science department at the University of Alberta in Edmonton, Canada. He received his PhD in 1994 for his work on Logistello - an Othello program that later defeated the reigning human World champion 6-0. His current research interests include heuristic search, pathfinding, abstraction, state inference, and opponent modeling applied to video games and card games. In these areas Michael and his students have made numerous contributions, culminating in developing fast geometric pathfinding algorithms, creating the World's best Skat playing program, and one of the strongest StarCraft Broodwar bots.