NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-04-29
January 20, 2004

A Strategic Game Playing Agent for FreeCiv

Philip A. Houk

Abstract

FreeCiv is an open source version of the game Civilization II (see www.freeciv.org). In this game, a player controls various attributes of a civilization as it grows over time. The program stability, open source code and developer commitment to continual improvement makes this game a good environment for performing research. This Masters Thesis project report describes the *QRG FreeCiv AI Player* (QRG FAP), which interfaces with the game and intelligently plays the initial expansion phase of the game. The project has included building a Lisp API to the game, designing game knowledge representations, implementing knowledge assertions about the game state and rules, building agents to handle specific areas of game reasoning and implementing an overall game playing agent that ties the sub-systems together. The QRG FAP was written in Lisp to take advantage of the Qualitative Reasoning Group's reasoning system FIRE. In the FreeCiv AI project, FIRE's Logic-based Truth Maintenance System (LTMS) maintains game information. The report discusses the QRG FAP design and tradeoffs associated with using a LTMS in a game environment. A qualitative terrain map was developed to simplify terrain reasoning tasks. The report explains how the map was helpful to the exploration agent and how it can be used to find strategic terrain masses. Several other programming techniques, described in the report, were adapted and used to help the QRG FAP explore and settle land during the game. Favorable comparisons are made between the QRG FAP, a human player and aspects of the FreeCiv AI. The entire system was designed and implemented with the goal of using the FreeCiv game environment as a rich domain for performing reasoning and learning research. It is hoped that this system will be used in further research.

Keywords: FreeCiv, Game Artificial Intelligence, Qualitative Terrain Representation, Qualitative Terrain Reasoning, Truth Maintenance System

Table of Contents

1.0 Introduction

FreeCiv is a wonderful open source version of the game Civilization (see www.freeciv.org). In this game, a player controls various attributes of a civilization as it grows over time. The program stability, open source code and developer commitment to continual improvement makes this game a good environment for performing research. FreeCiv is written in C and uses a client/server architecture. The server runs the game, controls the computer players and updates the human players. The human players interface with the game as clients. The clients and servers communicate through a socket interface.

My thesis project (the *QRG FreeCiv AI Player* (QRG FAP)) has included building a Lisp API to the game, designing game knowledge representations, implementing knowledge assertions about the game state and rules, building agents to handle specific areas of game reasoning and implementing an overall game playing agent that ties the project together and successfully plays the initial phase of the game. This work has focused on FreeCiv version 1.13, the version of the game that was current when I started this project. The QRG FAP was written in Lisp to take advantage of the Qualitative Reasoning Group's reasoning system FIRE. FIRE offers a number of computer problem solving tools in one system. In the FreeCiv AI project, FIRE's Logic-based Truth Maintenance System (LTMS) maintains game information. Several sets of rules were defined in FIRE to suggest game plans, needs and actions. In some cases, the AI queries the TMS for game information and makes game decisions accordingly. In many situations, functions were written to query and make decisions using the Lisp image of the game. Using the Lisp image can be faster when performing computationally intensive operations. The entire system was designed and implemented with the goal of using the FreeCiv game environment as a rich domain for performing reasoning and learning research. It is hoped that this system will be used in further research.

This report will explain in detail my Lisp-based, FreeCiv AI system. I will start by describing the Lisp interface to the game and the knowledge representations the API makes in the TMS. The remainder of the document will describe the QRG FAP that plays the initial expansion phase of the game. In this report I have used the term agent to describe a sub-system of code that handles a particular portion of the game reasoning. After explaining each portion of the QRG FAP, I will discuss the relevant tradeoffs associated with my approach to this portion of the game AI along with several extensions that would improve this portion of the game AI. Finally, where appropriate, I will discuss the results of comparisons between the QRG FAP, a human player and aspects of the current FreeCiv AI.

2.0 Lisp API to the Game

In the initial stage of this project, I learned the FreeCiv code and created a Lisp based API to the game. This involved developing an object oriented class structure for game

information, a socket based game information communication system and serialization/de-serialization functions for the game information and requests.

The class structure I developed for representing game information draws heavily from the structures in the game code. In most cases the class names I used for game objects were the exact same as the FreeCiv structure names. I only created fields in these classes for the information that could possibly apply to the QRG FAP client. Since we don't plan to implement a Lisp based, graphical interface to the game, I didn't bother saving sprite information or help text. I also omitted many of their AI fields along with fields that seemed to be deprecated. Most of the fields have documentation strings that explain my interpretation of how the field information is used in the game.

FreeCiv was designed as a socket based client/server game system. The server manages all the players, rule settings, the built-in AI players, and game states. Every player, aside from the game's AI players, must connect to the server via a socket. The Lisp client communicates with the game through this same socket interface. The QRG FAP client will look like a human player to the FreeCiv server. It operates with exactly the same information a human player has. I have written code that deserializes all of the rule and game state information. This system also includes functions that generate and serialize all of the game actions/requests a player can make. The system performs substantial error checking and maintains a game log that includes important program activities along with error notifications.

In order to maintain communication with the server while reasoning about the QRG FAP's game state, I designed and implemented a multi-threaded architecture for this client. One thread is dedicated to communication with the server. This thread blocks for either server data or game requests put on a request list. The second thread reasons with game state information and plays the game. Several gates and process-locks were implemented to ensure data integrity.

3.0 Updating the Game State

One of the most difficult design decisions involved with this project was deciding how to represent the game state in the Truth Maintenance System (TMS). What was the best way to keep the reasoner focused on the current game state? Ideally, I wanted to have past game state information available so the TMS could be mined for learning and improving the AI strategies. I devised two methods for representing game state information that met both goals, but found what I felt were big problems with both methods.

In the first method, I planned on making game assertions. Then I could create implications where the current turn enabled these assertions. The biggest problem with this approach was handling information updates during the turn. The current turn can enable different states during the turn as game actions are taken. I couldn't find a good way to get around this problem.

In the second method, I planned on adding a turn field to each relationship/predicate I used. One problem with this approach was that there is a lot of game information that doesn't change from turn to turn, for example the game rules or terrain type information. I didn't want to assert this information over and over for each turn. It may have been okay to ignore turn information for things like game rules in the knowledge base. However most of the game state information can change during the course of the game. Keeping all of these turn number indexed game assertions around in the TMS would quickly bloat memory usage. I knew that late in the game the system would run out of memory.

So instead of either approach, I decided that the TMS would only maintain a representation of the current game state. If we wanted to mine the TMS for learning, we would have to devise a way to save the pertinent game state information over the period of several turns. Reasoning that required information from previous turns could be carried out on representations using the second approach above.

The game information updates happen via calls to the overloaded `update-game-property` methods. These methods handle all types of game updates in the TMS and the LISP object representations. A list of specific assertions can be found in the documents folder with the code. In some special cases, the updates trigger special actions. For example, when the LISP client receives information, during its turn, about a newly discovered tile, the tile location is placed on a global `*NEW_TILES*` list. This list signals the AI to incrementally update the information it uses for some of the reasoning tasks. In other cases, the TMS updates automatically impact the AI reasoning via FIRE rules and updated game state assertions.

4.0 The Expansion Phase of FreeCiv

In the beginning of the game, the main goals for a player are to rapidly uncover terrain and settle/claim as much of the discovered land as possible. In a typical game, the players settle land until they meet boundaries that force them to change their initial expansion strategy. The changes are driven by the need to either interact with an expanding neighbor or the inability to cross water tiles. In either case, the first expansion phase of the game remains the same. These initial cities provide the resource base for the civilization during the rest of the game. The effectiveness of a player in the remainder of the game is greatly determined by this expansion phase. The most interesting part of this project is the AI that effectively plays this expansion phase of the game. Specific aspects of this AI are detailed in the remainder of this report.

5.0 Exploration

The ability for an AI to explore and maintain current knowledge of the world is the key to a fair computer player. Most computer players in strategy games need map omniscience in order to compete reasonably well. I believe there are times where omniscience hurts the player's strategy rather than helping it. Certainly AI player omniscience leads to unrealistic game play. When a game AI has map omniscience, exploration is not a great

concern. In order to have a good AI player without map omniscience, the player must have an effective exploration capability.

Effective exploration entails rapidly discovering tiles that are of strategic interest to the player. In the early phases of the game, interesting tiles include potential city sites, defensive positions and mobility corridors. As the game continues, the explorer will focus on exploring terrain inhabited by other civilizations. The AI determines the strategic interest of unknown tiles by reasoning about the currently known map information. The explorer must discover land in a way that continues to support these strategic concerns.

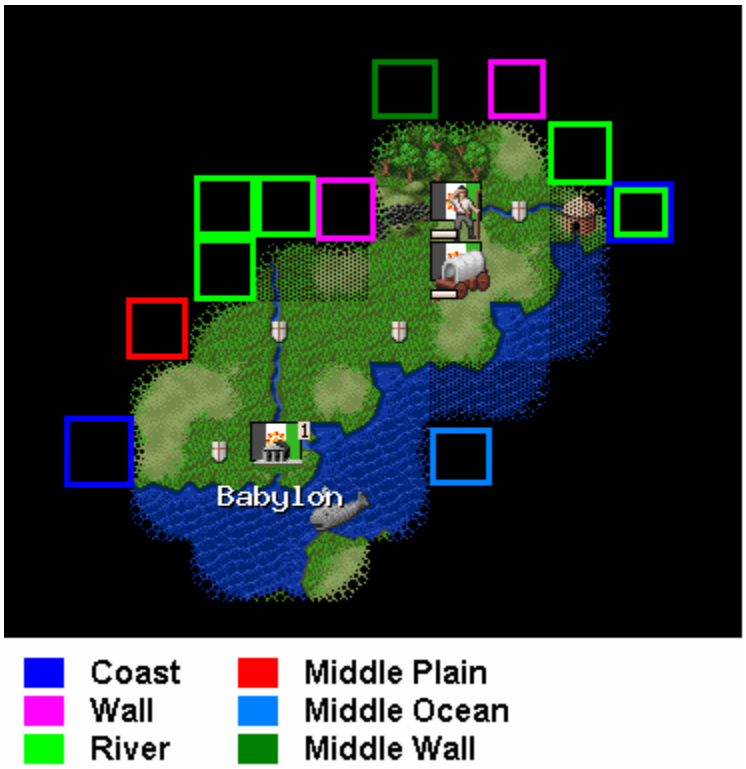
The major uses of terrain reasoning are city site planning, defensive positioning and offensive maneuvering. Reasoning about terrain at a micro (individual tile) level can lead to intractable algorithms. Regions of tiles provide much better strategic information and reasoning leverage to a player. The QRG FAP uses terrain regions to support its game playing efforts. In the case of exploration, the agent uses an *influence map* [Tozour, 2001] [Woodcock, 2002] to make decisions. This influence map data structure (`*EXPLORE_MAP*` in the code and explore-map in this paper) combines exploration derived from two types of terrain regions, described in the following sections, the qualitative terrain map and potential city sites.

5.1 Qualitative Terrain Map

As the AI learns about the world map, it creates a qualitative terrain map. This map is generated by noting contiguous sections of common terrain, called *Blobs*. An obvious, and useful, blob classification is the distinction between land and ocean tiles (i.e. continents). A second type of division used by the QRG FAP is based on the terrain type of a tile. The specific blob classification used by the QRG FAP will be discussed later in this section. Once the known portion of the map has been dissected into blobs, the system finds the outer edges of the blobs. The blobs and edges comprise the qualitative terrain map. Blobs of common terrain can be useful in many terrain reasoning tasks. They help define strategic terrain positions (e.g. choke points, mobility corridors or terrain walls). They can also be used to improve path-finding and reason about unit movements or positions. The Explorer Agent in the QRG FAP uses the qualitative map to drive strategic exploration.

One goal of the Explorer Agent is to determine the extent of blobs in a systematic and thorough manner. Initially, the Explorer Agent focuses on the outer edges of the blobs. The most strategically interesting tiles in the game map are at the junctions of two distinct blobs. For example, the Explorer Agent will want to follow a coast or terrain wall before it wants to discover the middle tiles of an ocean, plain or mountain range. This strategic exploration drive is balanced with a desire for uniform knowledge of the terrain surrounding the civilization. So the Explorer Agent will not necessarily want to follow a coast to its end. At some point, it will become more important to discover the middle tiles of the blobs which are closer to the civilization.

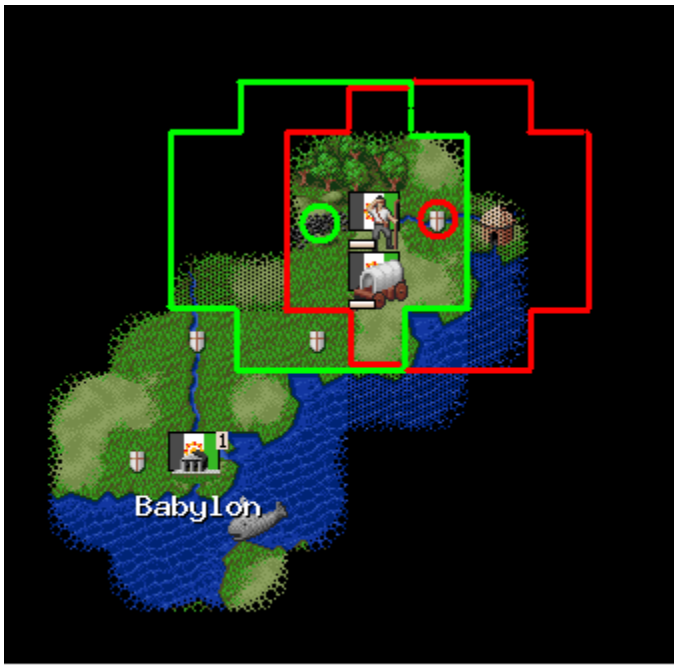
The qualitative map drives construction of the explore-map. The algorithm cycles through all of the blob edges looking for unknown tiles adjacent to the edges. When an unknown tile is found, the known tiles adjacent to it have their blob-types queried to guess the strategic importance of the tile. The strategic importance categories come out of the discussion above. They are defined as one of the following: coast, terrain wall, river, middle ocean, middle plain or middle wall. The screenshot below shows examples of each category for an early game state.



Each category has a parameter value that adds to the exploration desire for an unknown tile in the explore-map. In the current code, these values range from 1 for a middle wall tile to 5 for a coast tile. These values were chosen to balance the city site exploration desire described in the following section. They provided good exploration results and were never analyzed for potential improvements to the Exploration Agent.

5.2 Potential City Sites

In order to evaluate potential city sites, the tiles that would comprise the city need to be discovered. Terrain tiles that contain special resources are usually far superior to ordinary tiles. These tiles will be called “good” tiles in this paper. The Explorer Agent is driven to discover these tiles with a mechanism that converts “good” work tiles to city sites that could harvest the resources at these tile positions. In the screenshot below, the colored boxes surrounding two of the goods indicate the tiles affected by this stage of analysis. The good tiles were circled to make them stand out more. Any unknown tile in this grouping of potential city sites is given a desire weight for exploration.

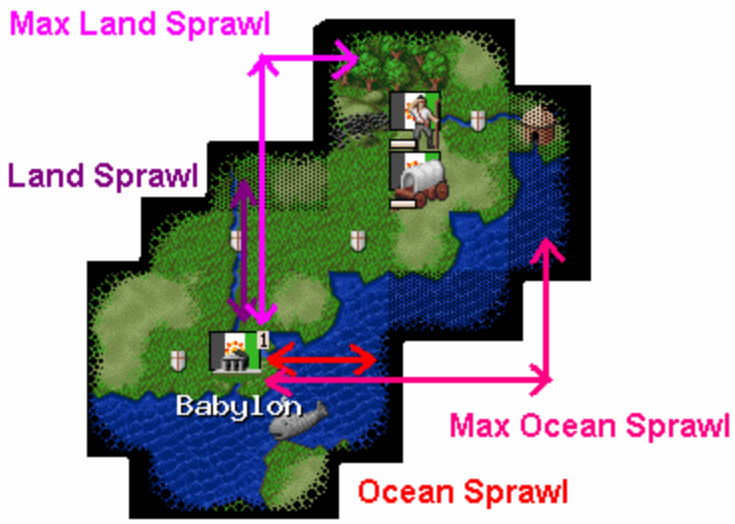


The exploration desire for these positions in the explore-map is increased by the total raw resource value of the "good" tile, as defined by the game rules. The total raw resource value for a "good" tile can range from 3 to 7, depending on the terrain type. These values provided good exploration results and were never analyzed for potential improvements to the Exploration Agent. When an unknown tile has several "good" tiles exerting discovery desire on its position the tile has excellent potential as a city site and therefore has a very high discovery value.

5.3 The Aggregate Influence Map

After the unknown tiles in the explore-map accumulate discovery value from the qualitative terrain and potential city site analyses, the exploration agent needs to determine the next tile to explore. The output of this decision is the currently known tile position that the exploring unit (explorer) will approach during its next exploration movement. Making this decision involves a tradeoff between the amount of exploration gain at any map position, the thoroughness of terrain knowledge surrounding the civilization and the distance the explorer must travel to discover a tile. Exploration gain is calculated in the explore-map for every known tile that is adjacent to an unknown tile. An example of this is shown in the screenshot below. The exploration gain value at a tile X is the sum of the discovery values for all the unknown tiles (marked O) that would be discovered if the explorer were to reach tile X. Similar discovery squares could be drawn for each of the edge tiles.

In order to make sure explorers balance the exploration radius around the civilization, a sprawl slot was added to the unknown tile class used in the explore-map. Sprawl is a measure given to the aforementioned known tiles adjacent to unknown tiles (called edge tiles for this discussion). The sprawl calculation is the distance from the edge to the nearest civilization city tile (or the game starting position if no cities exist). The screenshot below shows typical land sprawl in purple and typical ocean sprawl in red. The maximum respective sprawls (for this game state) are indicated in light pink and hot pink. After calculating sprawl for the explore map, the maximum land and sea sprawls are cached separately for the exploration decision making process. These maximums are incremented so the decision maker never gets a zero result in its subsequent calculation.



The exploration gain and sprawl values are used by the Exploration Agent to make exploration decisions. This decision calculation is explained in the next section.

5.4 Choosing a Tile to Explore

The explorer's next tile destination decision is based on a comparison of the edge tile's exploration gain weighted by the sprawl and the distance from the explorer via the following equation.

```
(floor (* (explorer-gain utile)
          (- max-sprawl (sprawl utile)) dsp-wt)
       (* (map_distance_vector x-e y-e x y) de-wt))
```

- `utile` is the edge tile
- `max-sprawl` is the maximum sprawl associated with the explorer's move type
- `dsp-wt` is a sprawl distance weight, currently 1
- The `map_distance_vector` function yields the distance from `utile` to the explorer.
- `de-wt` is a explorer distance weight, currently 1

The current sprawl and explorer distance weights obviously don't give any input to the equation. They were added to the equation to give the Exploration Agent better control over this decision. I experimented with different weights and found that they were not helpful to the Exploration Agent. They were left in the code in case other exploration scenarios could benefit from their use.

5.5 Exploration path-finding

After the exploration agent chooses a destination, an enhanced path finder is called to generate a path that keeps movement cost low but allows the explorer to wander a little along the way if it can discover some unknown tiles. This path finder draws heavily from the well written FreeCiv path planning code (in version 1.14.1). In the game code, the path finder uses Dijkstra's algorithm with movement cost and an extra cost (typically used to deter the use of a tile in a path). The QRG FAP exploration path finder uses an A* search with a heuristic that divides the path's movement cost by the path's explorer-gain. The explorer gain is the sum of actual and potential future (based on straight line distance) exploration gain. Initially, the potential future gain for each tile in the straight line distance is set optimistically (to the maximum explorer gain found anywhere in the explore-map). This potential future gain provides heuristic incentive for wandering. In order to reduce the search space and explorer wandering, the optimism for potential future gain is reduced as the path becomes longer than reasonable (an indication of too much wandering). This reduction is controlled via global parameters.

An example best illustrates the algorithm. The screenshot below shows four paths the explorer could take to get to the red tile. The green path minimizes movement cost and only discovers the red tile. The other paths aim to discover tiles along the way. The hot pink path is an example of excessive wandering. I have set the parameters so the Exploration Agent tends to take paths more like the blue and dark purple paths.

This cost heuristic was devised via a lot of testing. It was tricky finding a balance between lowest movement cost paths and aimless wandering for explorer-gain. The goal of this path finder was to pick up a little more information as the explorer moved toward the tile with the best exploration gain. In order to achieve this goal, the wandering reduction parameters needed to be integrated into the cost formula. They are automatically adjusted based on the straight line distance between the explorer and its

destination. Without this adjustment the explorer will take one of two extreme paths. With one set of settings, the explorer takes the lowest movement cost path to the tile chosen by the Exploration Agent and it doesn't explore along the way. The other set of settings let the explorer take a long and indirect path that includes every unknown tile in the region.

Another technique used by the path finder to limit the search space was borrowed from FreeCiv's implementation of Dijkstra's algorithm. When evaluating paths going through a particular tile, the path with the lowest cost as it reached that tile was cached in a path-finding-map. Only paths with a lower cost get extended. This cache made path-finding tractable for paths longer than 14 tiles. Without the cache, the number of potential paths considered by the path-finding algorithm became larger than system memory could hold.

5.6 The Exploration Agent Procedure

The current Exploration Agent manages exploration for only one explorer unit. It incrementally updates the qualitative map and creates a new explore-map at the beginning of each turn. It chooses the best tile to explore, finds a path and moves one tile along that path. During the turn the agent will incrementally update the qualitative map and the explore-map and check for a new destination tile every time the explorer moves. New destination tiles will obviously get new paths. If the destination remains the same, the explorer will keep its original path.

When an explorer moves around the map, there are some additional rules that need to guide its movement. For example, an explorer doesn't have a very good attack/defense capability. So explorers should avoid uncovering huts and contact with (non-explorer) enemy units. The Explorer Agent avoids huts as it explores unknown terrain. This behavior is created via a hack that makes hut tiles unknown when the Explorer Agent is working with a defenseless explorer. Non-explorer unit types can be given the exploration task via the `ai-activity` slot. These "explorers" will uncover huts as they are found. The exploration code will identify and control these "explorer" units, but the QRG FAP does not currently assign units this task. Idle attack units are sent by the QRG FAP to uncover known huts.

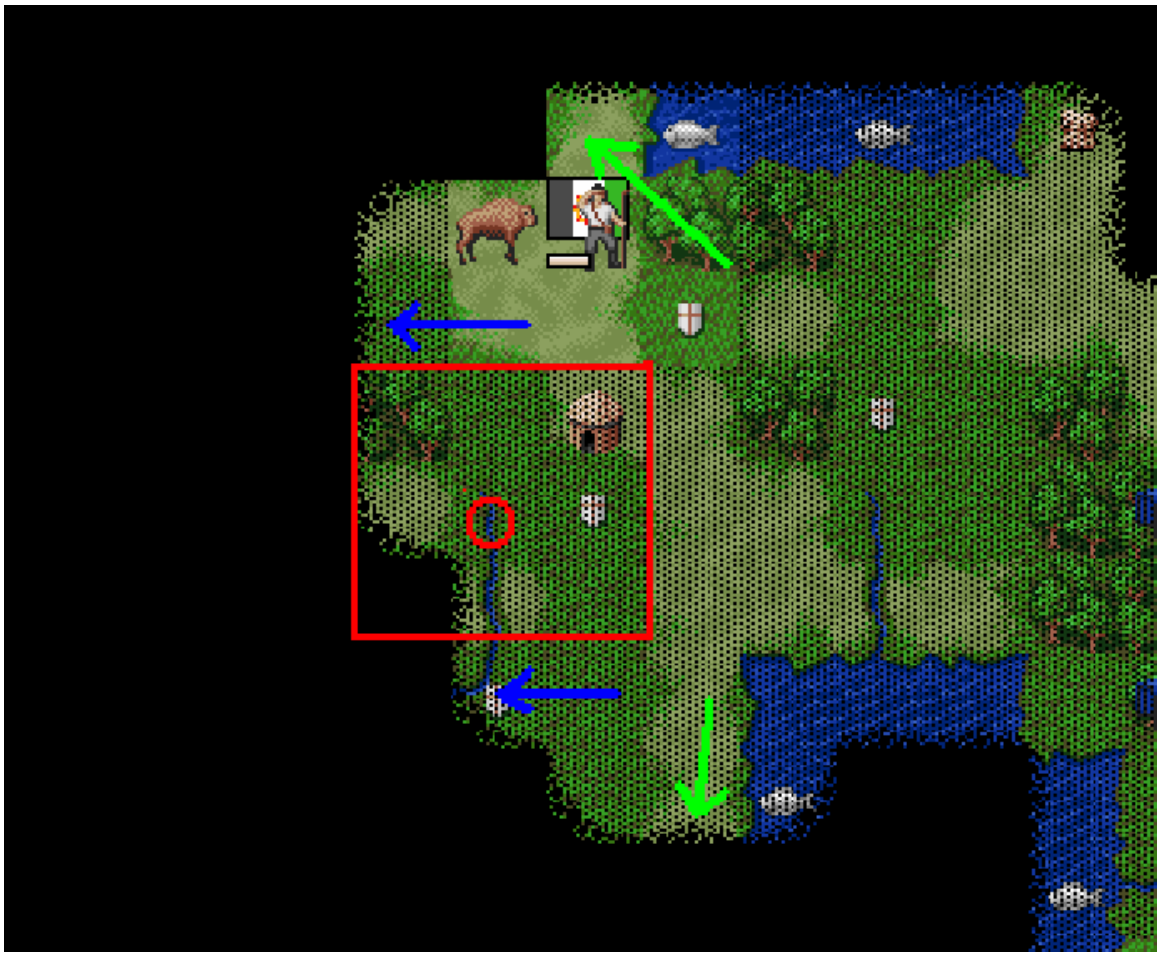
5.7 Other Approaches

5.7.1 Life without a qualitative map

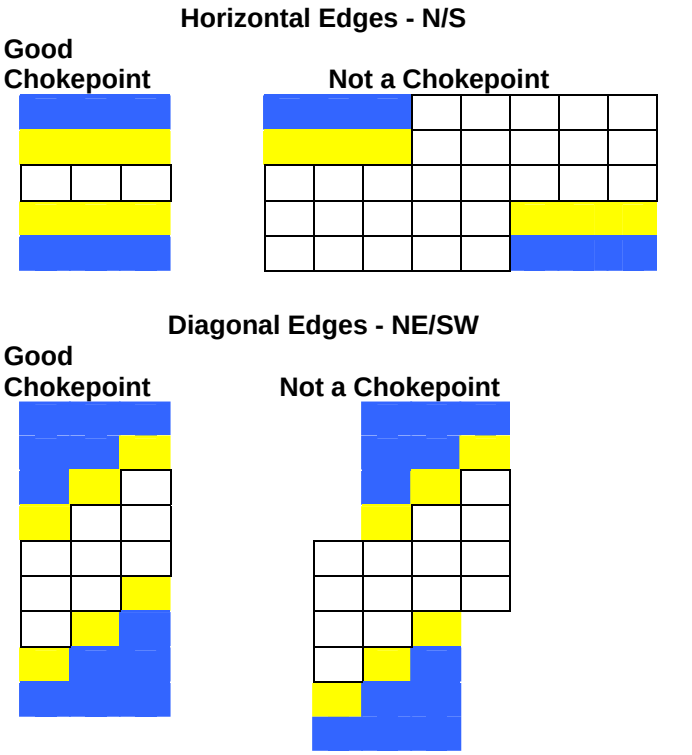
The qualitative map simplifies the task of determining unknown tile strategic importance. However it is not absolutely necessary for driving the explorer. Instead of using the blob edges to find the relevant unknown tiles, the program could cycle through the entire map. When the unknown tiles were found to be adjacent to known tiles, a classifier could give the appropriate exploration desire weighting to the unknown tile. The qualitative map simplifies both aspects of this algorithm. I chose to use a qualitative map, because I felt the map would be useful to many other game reasoning agents. It is similar to the mip mapping used in Age of Empires [Pottinger, 2000] with the important difference that the

QRG FAP system knows each blob of terrain has a more interesting commonality. Instead of combining the terrain based on location and continuity, blobs are combined by location, continuity and some function of the terrain type. The qualitative map also supports higher level path planning [Pottinger, 2000] and [Forbus, Mahoney & Dill, 2001].

There are more reasons for using a qualitative map. The qualitative map should provide useful information for reasoning about the positions of other civilizations. Edges that are adjacent to known enemy retreat directions will help direct the AI's offensive and defensive military operations. Plans for scouting missions will be easier to develop with this information. The current method of handling this situation, described in many sources [Tozour, 2001] [Woodcock, 2002], would involve influence mapping known past and present enemy positions. The screenshot below shows an influence area in red, based on an enemy sighting at the tile marked with a red circle. Using this influence area would give some idea of the enemy's boundaries, but would make them seem small. Some computational analysis would be needed to find the edges of the influence area. Exploring the edges of this influence, via either of the blue arrows, could send the explorer into the heart of the enemy civilization. With the qualitative map, all of the unknown blob edges that could border the enemy civilization are easily found. Combining this information with the blob's mobility information allows the agent to deduce that the enemy has probably continuously settled this unknown land from the two coastal extremes marked with green arrows. This more realistic information sets the stage for better planning when scouting the enemy terrain.



Strategic terrain information will be easy to define with the qualitative map. Although it is not used by any of the currently implemented agents, I developed code that finds choke points based on the qualitative map. Choke point determination is quite fast because choke points can be defined by a situation where two edges of opposite direction, for example north – south, overlap and are separated by a small distance. The following diagram shows examples and counterexamples for chokepoints. In the diagram, the blue tiles are impassable tiles. The yellow tiles are the edges of the passable terrain. Only edge pairs are shown to better illustrate the algorithmic comparisons. The good chokepoint examples have both good distance and overlap. Both counterexamples show the case where the edges don't overlap.



The algorithm is fast because it only needs to look at edges with opposite directions that are part of blobs with good mobility characteristics. Opposite edge comparison also enables the AI to find diagonally oriented choke points. Terrain walls and mobility corridors should be equally easy to find because the search space is limited by the blob types. Age of Empires uses a choke point determination algorithm that looks at the shapes of the blobs [Pottinger, 2000]. They have found this algorithm to be quite good with horizontally and vertically oriented choke points, but less effective with diagonal choke points. The mip maps used in Age of Empires for choke point finding should provide enough information to find walls and corridors, if this information was desired, unless they are diagonally aligned.

5.7.2 Other ways to calculate exploration desire based on city sites

The city site exploration desire could have been calculated with a more complex weighting function. The unknown tiles could have been analyzed as city sites. In this scenario, all of the known city-site resource tiles would exert influence on the unknown tile rather than just the “good” tiles. This would require a lot more calculation and would still have a lot of uncertainty. I felt that the “good” tiles were a quick way to estimate this larger calculation. The city site finding agent will do this more thorough analysis when it compares potential city sites for settling. During the exploration phase of the game, this level of detail seems excessive.

5.7.3 The FreeCiv Approach

In FreeCiv version 1.13, the exploration code focuses on unknown tiles and movement cost to explore these tiles. The auto-explorer chooses to explore the unknown tile with the smallest movement cost from the explorer. In order to limit the search for unknown tiles, the algorithm first looks in a small box around the unit. If no destination is found, the explorer will look at the entire map. If no tiles remain unknown, the explorer stops. In my experience, the explorer will sometimes stop if the unknown tiles are too far away.

In FreeCiv version 1.14.1, the exploration code has been improved. This algorithm still focuses its attention on a small box around the unit. In addition to comparing the movement cost for exploring a tile, it includes a desirability rating for the unknown tiles. The desirability is calculated with the philosophy that coastal tiles are highly desirable. The desire calculation uses the adjacent tiles of an unknown tile to guess if the tile is along the coast. This approach is similar to my algorithm's qualitative terrain map checks. The main difference is that my algorithm checks for more terrain attributes. They have also added a desire for exploring huts. The FreeCiv developers have thought about adding desire for unknown city map tiles. They have not added this feature because they feel the improvement would not be worth the computational cost. If no tiles remain unknown, the explorer is sent back to its home city.

Both of the FreeCiv auto-explorers can control multiple explorers. The strong focus on movement cost and localized unknown tile search keep the units exploring in a cooperative manner. This cooperative behavior should be emulated as the QRG FAP Explorer Agent is extended to manage multiple explorers. Both auto-explorers choose to uncover huts. In many cases the explorer dies. This problem is less crucial to the FreeCiv AI, since other units can take up the exploration task. However, the explorer does have special attributes that make them nice to have around.

5.8 Results

It is difficult to compare my Explorer Agent code with the FreeCiv auto-explorer. The FreeCiv AI currently cheats, by using map omniscience. Their explorer is only needed to uncover city map tiles so they can be worked.[1] The explorer is also used to give human players the impression that the AI is really exploring based on need.

The FreeCiv AI's map omniscience and ability to control multiple explorers makes fair comparisons to the QRG FAP difficult. The FreeCiv auto-explorer's effectiveness has no impact on city site planning or any measurable part of the game. In order to make a fair comparison, I ran four games as a human player and set the explorer to use the FreeCiv auto-explore mode. Then I used the QRG FAP city site and management code to handle the rest of the game playing. The maps were randomly generated without huts or barbarians. Furthermore the AI players were always placed alone on fairly large islands.

In comparing the two AI's, I wanted to see which Explorer Agent better supported the game-playing agent in the initial expansion phase. This phase of the game sets the stage

[1] Even though the FreeCiv AI has access to all of the map information, it can only place a worker on city tiles that have been explored.

for the remainder of the game. The faster the game player moves through this phase of the game, the better positioned they are to excel in further stages of the game. Three time-based statistics measure the Explorer Agent's effectiveness. The number of turns spent exploring the entire island measures exploration speed. The number of turns taken to settle the entire island measures expansion speed. The number of turns taken to reach the technological state of "The Republic" measures the effective use of city resources. The Republic form of government provides significant advantages in the game. Rapidly reaching this government type can make a huge difference in the game outcome.

In addition to the time measures, I used an economic measure of civilization growth. The economic measure is based on the idea that a city adds value to the civilization every year. The longer a city has been around, the more value it has given to the civilization. I have estimated this effect using an economic formula, future value[2]. My calculation uses the city's total resource value (described later in this report) as the asset value. The future value is calculated for the year T the second AI finished settling the entire island. Each city was founded at some year t (where t <= T). At time T, the city has been contributing to the civilization (future value) for T-t years. I calculated the future value for each city based on its asset value and the time it contributed to the civilization. The intuition is that the resources are available to the city starting in the year it was founded. The longer the city has access to these resources, the more value they convey to the civilization. The measure of the civilization's future value at time T is the sum of the future values for all of its cities.

The following table summarizes the results comparing the QRG FAP with the FreeCiv auto-explorer for four randomly generated games. Since there was no clear way to decide the interest rate to use in the future value calculation, I made the comparisons using three distinct values 1%, 5% and 10%.The individual game statistics can be found in Appendix A.

	Sum All Experiments		
	QRG FAP	FC Explorer	Percent Improvement
Republic	171	179	4.47%
Goods Left	17	19	10.53%
Explored	182	169	-7.69%
Settled	283	298	5.03%
FV 1%	153978.55	141862.94	8.54%
FV 5%	655728.83	612557.32	7.05%
FV 10%	9519642.97	8557894.68	11.24%

The QRG FAP Explorer Agent outperforms the FreeCiv auto-explorer in every measure except for the time to explore the island. This result is not surprising because the auto-

[2] Future Value takes an asset's value at some time t and uses an interest rate to project the asset's worth at some future time T. The projection is based on the idea that the asset's value compounds over time. An asset that has been owned for a longer period of time will be worth more in the future than an asset that was owned for a shorter period of time. This calculation was designed for revenue generating assets (e.g. money) and ignores depreciation of the asset.

explorer doesn't attempt to use any strategy when exploring. It locally explores. So it tends to completely explore one area before moving to another. The Explorer Agent will cut across the civilization (expending time) to ensure a fairly uniform knowledge of the world surrounding its civilization.

The auto-explorer's effectiveness is random, based on the location of good city sites in the randomly generated map. Essentially, if a good city site is to the northwest of the initial starting point, the auto-explorer will probably work okay. Otherwise, it will explore in this direction and not find the best sites until later. This lack of strategy reduces the effectiveness during the initial phase of the game.

I was unable to compare the Explorer Agent to the auto-explorer in FreeCiv version 1.14.1. My guess is that the results would be closer, but this would also depend on the map. Aside from movement cost, this auto-explorer doesn't seem to have a mechanism that keeps it from following the coast all the way around the island. This exploration strategy would not support good initial city settlement either. Furthermore, I would hope that the potential city site exploration influence would be an advantage for the Explorer Agent.

It is interesting to note that effective exploration does improve all aspects of a civilization. Early exploration is especially important because the effects of early game decisions compound the most during the game. The first few settlements must quickly build settlers to support civilization growth. These cities also reserve "good" tiles and define the starting point of the civilization expansion. Poor city placement decisions can cause poor future city sites.

5.9 Future Exploration Work

Currently, none of the agents recognize enemy units. This is an obvious shortcoming of the current QRG FAP. My approach to this problem would be multi-faceted. First of all, defenseless explorers should run away from enemies and toward the main part of the civilization. Explorer unit types should take advantage of their terrain ignoring special capability and retreat into tiles with higher movement cost. Military units should be dispatched to handle the intruder. A TMS assertion should be used to allow the AI player to keep track of the original location of the enemy unit. This location can be used along with the qualitative map to reason about locations of enemy cities and units. The AI player should build cities and fortresses in strategic locations to prepare for future attacks. The AI player might want to settle land faster in this area of the map to get it before the other player settles there. Depending on the AI strategy, diplomatic relations could be attempted.

Exploration in enemy terrain will make heavy use of these strategies and might benefit from partial order plans. For example, plans could be instantiated where an explorer runs through a section of enemy terrain to a transport waiting somewhere along the coast. The boat then would ferry the explorer back to safety. The explore-map influence parameters would be altered to handle enemy terrain exploration. When exploring enemy terrain,

possible city sites might not be a major driver for exploration. Alternatively, the good tiles could provide clues that an enemy city could be nearby. In either case, the exploration desire associated with city sites might be reduced or eliminated to reflect a change in exploration interest. Exploration path planning in this situation might favor high movement cost paths, especially for explorer and spy unit types that have the special ability that they ignore terrain movement cost. The current exploration path finding doesn't handle this special ability and would also need extension to explore in this manner.

There are times when the Exploration Agent leaves a one or two tiles unknown and moves to a different part of the map to discover more strategically interesting tiles. This behavior is sensible, based on the current exploration calculations. However the total island exploration time suffers because the explorer must return to discover these tiles after the remainder of the island is discovered. It would be nice to extend the Exploration Agent so it has a greater sense of thoroughness. On the surface this problem seems trivial. However it is difficult for the agent to look at these unknown tiles and know with certainty that they form the edge of the island. It must reason with the known edge information and determine the probability that these unknown tiles will not lead to more unknown tiles. For example they may uncover land bridge to another island. Although it would be good to know about the land bridge, it might not be as important as the information about the rest of the island.

Ocean based exploration has many similarities with land exploration. The current code has a few hooks meant for ocean exploration. However some important differences will require study when extending the land Explorer Agent code to handle boat exploration. Obviously, boats will only be able to explore the outer edges of a land mass. This constraint might warrant changes to the use of the sprawl parameter when choosing a tile to explore. Additionally, early ocean vessels need to remain close to shore to prevent loss at sea. Rules or modifications to the path finding heuristic will need to ensure this and other enemy based constraints hold. Finally, some agent will need to assign boats to the task of ocean exploration, because there are no boats in the game with this specific designation.

Part of the boat's role in exploration involves its use ferrying explorers that can explore the inner parts of the newly found continent. It would be a good idea to extend the exploration code to make sure the land-explorer will only be assigned exploration tiles that are on its continent. Otherwise the explorer may freeze trying to get to an explore tile on another continent. Once the new continent is better known, the AI player will also need settler and military unit ferrying. Based on my reading about FreeCiv's AI, the seemingly simple task of ferrying can be quite complicated to implement. Hopefully the TMS gives better leverage when approaching this problem.

A final extension needed for this agent is the ability to coordinate the activities of multiple explorers. When several explorers are working in unison to explore the outskirts of the civilization it will be important to make sure that the unit closest to the most desirable explore-tile gets the assignment. If the agent doesn't handle exploration

assignments in this manner, the explorers could constantly run back and forth across the center of the civilization which counteracts any gains from having multiple explorers.

6.0 Settling Cities

As the world terrain becomes known, the AI player must effectively settle the land. An effective settling arrangement involves tradeoffs between resources available, the amount of city tile overlap, defensive bonus for the city and effective use of qualitatively defined terrain blobs. The QRG FAP evaluates this tradeoff and settles the explored land in a systematic manner. The QRG FAP aims for big cities that share few tiles, a "large-pox"[3] strategy.

6.1 Evaluating City Sites

When the AI player has an available settler, it searches the known terrain for the best city site to settle. The City Site Evaluation Agent compares potential city sites using a few criteria. City site statistics are cached in a structure with slots for the raw resources, modified resources, count of "good" tiles, count of shared tiles, count of valid tiles, count of unknown tiles, the defensive bonus and a Boolean identifying coastal city. Some of the slots are currently ignored when evaluating city sites. In addition to using these statistics, city sites are evaluated based on their distance from other cities and a weighted, best-three worker resource value.

The major contributor to city site effectiveness is the net resources available to the city. The AI evaluates the resources on a raw and modified level. The raw resources are the basic tile resources in the potential city site's city map. The modified resources extend these raw resource values, by imagining that they have been modified in every potential manner. For example, a modified resource calculation would assume that a tile is irrigated and mined. Aside from Hill terrain, the modification calculation does not check the rules for consistent combinations of tile improvements. It applies all possible improvements. The intuition here is that we don't know which improvements might be desirable so we apply them all and compare city sites at this common level. Hills are treated differently because they yield 3 shield points for a mine and 1 food point for irrigation. Allowing both improvements gave too much value to hill tiles so only the mine modification is used. By summing the raw and modified resources, the city site value judges the long term city value.

The site evaluator has a few constraints that guide the accumulation of the potential city site's net resources described above. In order to make a fair AI player, tile statistics only

[3] The FreeCiv community has named two common settling strategies "small-pox" and "large-pox". The distinction comes from the inter-city distance and the resulting amount of workable tile sharing between cities. "Small-pox" means small inter-city distance, which leads to a large amount of tile sharing between the civilization's cities. This strategy typically results in a larger than average number of small cities. Each city cannot grow too big, because they don't have enough resource tiles available to support a large population. "Large-pox" means large inter-city distance. These cities share few tiles and can grow quite large. With this strategy, fewer cities are settled to cover the civilization's land.

get added for tiles currently known to the player. Unknown tiles are counted for future evaluation. The “good” resource tiles drive the city site evaluation process. A potential city site will only be evaluated if it has at least 2 “good” tiles available to it. The “good” tiles are blocked from potential city sites by the actual city sites. If a city is working a “good” tile, the tile is not considered valid to any other potential city sites. If the “good” tile is part of a city’s map area but not worked, it is counted as an assigned tile. When assigned, the “good” tile will count toward the 2 “good” constraint but the tile resources are not added to the site statistics. This manner of handling city sites reflects the intuition that a “good” tile that is part of an established city will most probably be worked by that city. However this “good” tile could be shared if the proposed site is still viable without the assigned “good” tile’s resources.

In order to measure the short term effectiveness of a city site, the citizen placement algorithm (detailed later in this report) is used to find the best resource harvest value using three workers. Since food and shield resources are more important to short term civilization growth, they receive a weight of two when adding the best-three value to the long term city value. This extra weighting also helps balance the impact of short term and long term city site effectiveness.

A city will be more effective if it is easy to defend. Cities that are settled on tiles with defensive bonuses convey those bonuses to their defenders. In cases where a potential city site yields a defensive bonus, this bonus adds value to the site’s evaluation function. Typically, there exists a tradeoff between defensive bonus and the resources harvested at a particular site. In my experience with the game, the resources are more important than a city’s defensive bonus when creating a winning civilization. For this reason, the city’s defensive bonus has a fairly low value and should only act as a tie-breaker.

Coastal city sites are tracked for two reasons. When a city is along the coast, it enables the harbor building improvement. Harbor’s make ocean tiles yield an extra food point and add to a site’s modified food value. Additionally, at this stage in the game, the AI player does not have access to boats. Without boats, the explorer can only discover tiles within one tile of land. This makes a lot of the coastal city site’s tiles unknown. Without these unknown tiles contributing some value to the site’s resource statistics, even good coastal city sites get passed over. The City Site Evaluation Agent will pro-rate up to 3 unknown tiles in proposed coastal city sites. The unknown tiles get the average known tile value, in this proposed site, for their value. If more than 3 tiles are unknown, only 3 estimated tiles are added to this site’s statistics. This behavior is a balance between the intuitions that most of these unknown tiles will be basic ocean tiles, but there are times when another continent will lend a “good” tile to this city site.

In order to limit settler travel distances, the city site value is reduced by the straight line distance between the settler and a proposed city site. This value reduction will tend to make closer city sites more desirable. Really good sites can easily overcome any possible distance reduction. Discounting the site value with this distance measure makes the civilization grow more uniformly and reduces the potential for settlers crossing the civilization to reach their settling site.

Instead of evaluating city sites based on the shared tile count in the site statistics, the AI site evaluator has parameters that define the minimum and maximum distance between city sites. The site evaluator will only consider potential sites that are within this distance range of the current or in-process cities. In-process cities are sites that a settler is currently moving toward to settle. They are maintained on a global list in the current implementation of the AI and assign themselves to their future city map tiles. Modification of these distance parameters will impact the civilization's strategy. Smaller distances lead to a "small-pox" game strategy and larger distances lead to a "large-pox" strategy.

6.2 City Site Evaluation Agent Procedures

An idle settler signals a call to the City Site Evaluation Agent, which chooses a site for it to settle. The site evaluator finds the proposed city site with the best value that is within the specified city distance range. When a valid site is found, the agent assigns the settler to this site, updates the in-process city list and sends the settler on its way.

When a settler reaches its intended city site, it requests a site evaluation check. This check lets the site evaluator slightly modify the city site location in case new information makes a nearby city site more desirable. Since the other in-process city sites were evaluated based on this site location, they could possibly affect the re-evaluation process. For this reason, the in-process sites are ignored during the re-evaluation process. This procedure allows for cascaded in-process city site repositioning when new information dictates changes to the plan.

When an in-process settler changes its city site, it can often make other in-process sites invalid. In order to reduce settler movement, all in-process settlers check to see if the site they are approaching is still valid with respect to city site distance. When an updated city plan invalidates the settler's site, the settler finds a new site to settle.

6.3 Other Approaches

My original idea was that the AI should break the known land into settlements using some sort of constraint satisfaction algorithm. It seemed like a clean idea to develop a settling plan for the entire known world. As I devised heuristics for the search, it was clear that this approach would be too computationally expensive. Each potential city site has a far reaching impact on the entire civilization settling plan. When new information becomes available, the plan needs to be scrapped and re-evaluated. Another idea was to start with some initial settlement points and use simulated annealing to refine the plan. Since each city site has such a strong interaction with the neighboring city sites, the simulated annealing approach would most likely turn into the same computationally expensive constraint satisfaction problem with a simulated annealing implementation. In the end I opted for the incremental settling method outlined above because it was a more tractable solution that still provided strategy to the city settling problem.

6.3.1 The FreeCiv Approach

The FreeCiv AI code uses a "small-pox" strategy. It aims for many small, coastal cities that have a lot of city map overlapping. Interestingly, their code estimates city site effectiveness based on a virtual city size of 20 citizens. In a "small-pox" settled civilization, the cities share too many tiles to allow any city to work that many tiles. The algorithm looks at all of the potential city's tiles. It temporarily caches the raw resources for the tile and the modified resources. Each resource is multiplied by a desire weighting (Food 19, Shield 17 and Trade 12). The modifications to tile resources include all of the possible terrain and many building improvements. The improvement values are amortized to approximate the cost associated with them. Then a virtual city of 20 workers gets incrementally constructed. For each worker added, the worker is placed on the "best" tile and the resources (with an extra food weighting) found at that tile are amortized, based on the number of workers, and added to the site value. Defensive bonuses for the site and some wild guesses/weights are added into the site value. These site values are cached for each map tile for the remainder of the game. As cities are planned or settled, the site values for cities within the settled city's map are decremented by 1. The settler chooses a site with the best value that is within 11 tiles of its current position and is not within the city map of another city. If necessary, a settler will request a ferry to get to this site.

This algorithm has many similarities with my algorithm. Looking at raw and modified tile resources can be found in both systems. Their emphasis on food and shield tiles will ensure good city growth and building potential. I chose a similar emphasis for the best-three calculation (the short term value), but did not weight the total site resources (the long term value). I felt that after the initial growth phase each resource is equally valuable, as long as it is used to its potential. Amortizing the resources harvested based on the city growth from 1 to 20 citizens seems to estimate the city effectiveness throughout the game. My approach of adding the best three worker output multiplied by 2 and the total raw and modified resources estimates the short term and long term health of the city. The FreeCiv AI calculates the city site value one time and only changes it slightly based on planned and settled cities in the area. It does not discourage sharing of tiles between cities. I chose to update the site map throughout the game and restrict potential city sites from harvesting other city's tiles with goods.

6.4 Results

The large number of tradeoffs involved with city site evaluation makes this decision making agent difficult to evaluate. The two city settling approaches have many important similarities and differences. It is difficult to compare cities that were settled with different goals in mind. The overlap inherent to the FreeCiv AI's "small-pox" strategy reduces the long-term resource value in the resulting city. The QRG FAP's "large-pox" strategy has the potential for slower initial growth. The web-based literature and opinion is that the "small-pox" strategy is a slightly better strategy. In my opinion, either the "large-pox" strategy or a combination of strategies should be the better approach to the game of Civilization. The successful civilizations in our world seem to match this belief. It would

be worth investigating how the game rules might be modified in order to more accurately reflect reality. This issue is a matter of considerable debate in the Freeciv community, and is beyond the scope of this report.

In addition to having different settling styles, the QRG FAP and FreeCiv AI have different views on cheating. The QRG FAP does not cheat; it operates using the same information that a human player would have. The FreeCiv AI has special abilities. Two of these cheats make it impossible to compare these AI's in the expansion phase of the game. The FreeCiv AI has map omniscience. This knowledge lets the AI plan and settle cities before exploring the terrain that would make up the city site. The second cheat is the ability to set the trade conversion rates to 100% regardless of the current government type. In the early phase of the game the FreeCiv AI gets 100% science and reaches "The Republic" technology in record time. In contrast, a human player can only set science to 60% at this stage of the game. Once the FreeCiv AI changes to a Republic government, the resource yields rise dramatically. Obviously, this resource increase drives faster civilization growth. In my observations, the FreeCiv AI is settling other islands before a human player reaches the Republic form of government.

Between the different city settling styles and the cheating, it was impossible to find a way to compare city settling effectiveness for the two AI's. Instead, I compared the QRG FAP to a "reasonably good" human player. For consistency, the human player settled with a "large-pox" philosophy and only explored with one explorer unit. The following table compares the QRG FAP with a human player for one randomly generated game. In addition to using the previously defined measures, the table includes the total site value of all settled cities and the average site value of the cities.

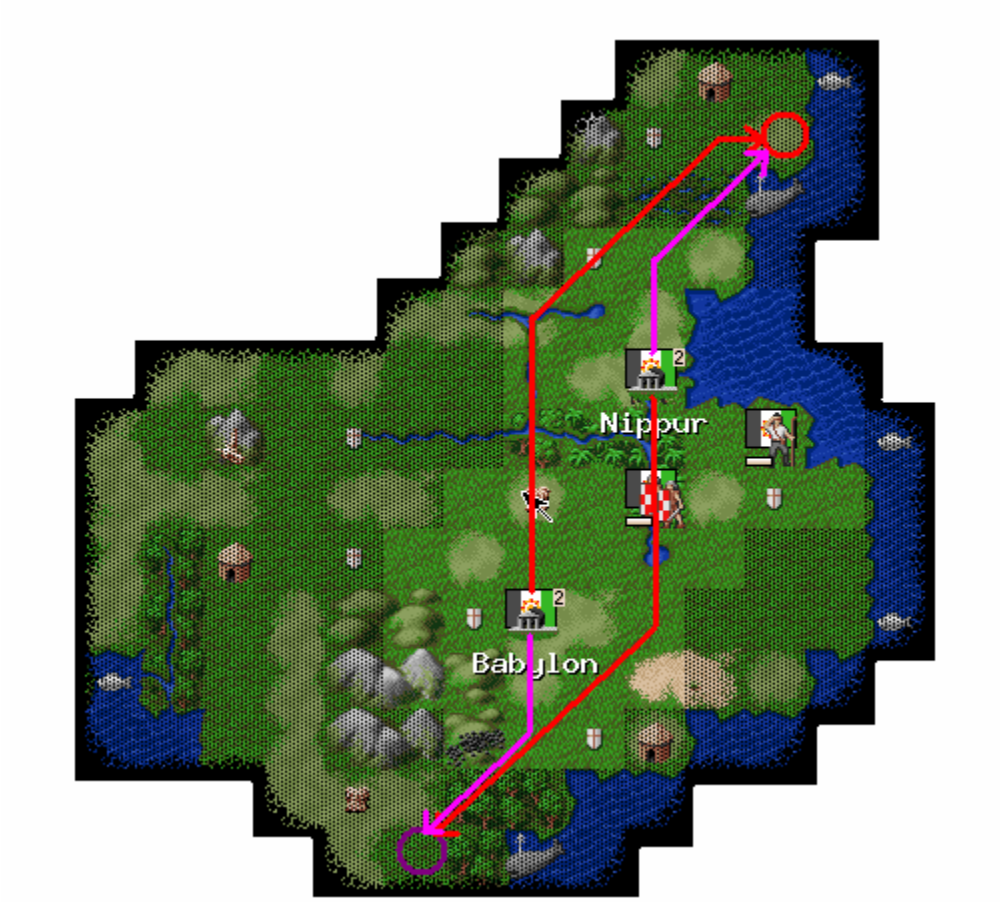
	QRG FAP	Human Player	Percent Worse
Total Site Value	863	811	-6.00%
Average Site Value	78.45	81.1	3.38%
Republic	42	40	4.76%
Goods Left	2	1	50.00%
Explored	44	34	22.73%
Settled	72	60	16.67%
FV 1%	34,640.57	39,351.81	13.60%
FV 5%	134,917.37	144,283.69	6.94%
FV 10%	1,399,333.34	1,345,296.67	-3.86%

Interestingly, the QRG FAP ended up with 11 cities as compared to the human player's 10 cities even though it left 2 "good" tiles. So the total site value for the QRG FAP was better than the human player's. Of course the human player stopped playing once the island was settled. So the QRG FAP essentially got 12 turns of extra game play. The average city site value was pretty comparable.

Most of the other measures were in the human player's favor, but not by too much of a margin. The human only reached Republic two turns earlier than the QRG FAP. Exploration time was in the human's favor because he put a little more emphasis on

exploration thoroughness and didn't leave single tiles unexplored before moving to other areas of the map.

The island settling time was considerably faster for the human and presumably this would translate into bigger gains if he kept playing the game. The major thing the human player did to improve island settle time was improved planning of the in-process city sites and the in-process settlers. An example best illustrates this improved planning process (see the screenshot below). A settler S might be available at Babylon. The best site to settle is marked by the red circle, a distance of 10 tiles away. The next best site marked by the purple circle is only a distance of 4 tiles away. Nippur is building a settler and it will be done in 4 turns. Nippur is only 4 tiles from the red site. The human player would send settler S to the purple site and let the in-process settler settle the red site. Both sites would be settled in 8 turns. The QRG FAP would send settler S to the red site, which would take 10 turns. The new settler would be sent to the purple site. The purple site would be settled in 12 turns. The QRG FAP would spend more time settling these two sites. This inefficient planning can really affect the island settling time.



The future economic value of the different player's cities was mixed. The lower interest rates favored the human player. The larger interest rate gave more value to the QRG FAP's cities. In this game, the first city chosen by the human player had less long-term value than the first city chosen by QRG FAP. The larger interest rate made this decision

more important to the total future value of the cities. The fact that the other measurements of the game were much more favorable for the human player suggests that an interest rate of 10% might be too high for comparing FreeCiv games.

6.5 Future City Settling Work

The planning issue raised in the results section above should be improved. In the site finding agent, I discounted the city site value by the distance from the settler to the proposed site to solve gross problems of this kind. Maybe a different weighting function for settler travel would help in more subtle cases. Alternatively, it might be okay to loop finding city sites until a site is found that an in-process settler won't be able to settle first. However, I would be careful to examine the potential need for generating settling plans based on the best sites along with the current and in-process settlers.

Currently the city site evaluation agent looks to settle cities within a maximum and minimum distance range of other current cities. A more flexible game AI might include the possibility of settling the best city in an area. For example, the agent might want to settle a city near a choke point that will protect its civilization from a known enemy. Settling this point might also effectively lay claim to a potentially large area of unsettled land between the new city and the current civilization. In fact, this could be a game strategy. Trying to lay claim to a large portion of land, protecting the borders and filling in the area as quickly as possible. Alternatively, the agent might find the best site, regardless of distance from other cities. This agent would act like a greedy algorithm. Each behavior could drive other strategic game decisions and would lead to personality in the game AI.

There are times when a city might be settled to harvest the resources from one "good" tile. The Goods Left row in the results table identifies possible examples of this situation. The site evaluator should be extended to find and seize these opportunities. In some cases, these city sites have no shared tiles with other cities. For example, the proposed city site might be along one of the poles and have very few workable tiles. In other cases, the "good" tile is inaccessible to the few cities nearby. This situation is caused by unknown information at city settling time, the inability to place a city that harvests all of the resources in an area or the fact that city sites using this tile were not very good overall. In any case, a city could be settled and managed to stay small, but harvest the "good" resource along with the other normal tiles in the area.

An important game strategy in FreeCiv involves city decision making based on the city type. I would define city types based on the resources available to the city. For example, a city that has an extraordinary amount of production available should be managed to take advantage of that production. In this case, barracks might be built in this city if a war is planned. Alternatively, the city could be set up to build the civilization's wonders. The problem with this approach is creating a definition for the city types that will remain valid over many game maps. It is a classification problem where the classifications are not defined absolutely. Instead they are defined relative to one another. Additional research will be needed to solve this classification problem. Although this paragraph

defines an alternative approach to the city management agent, it might prove useful to the site evaluation agent. For example, a civilization might have a large number of production typed cities. A city site evaluator might give some extra value to the other city types to maintain a balance of city types in the civilization. Of course the better strategy might be to take the best city we can get and balance the civilization via expansion and colonization.

Qualitative terrain information was very helpful for guiding the QRG FAP explorer. It should provide the same assistance to a military strategy agent. One important aspect of military strategy is the positioning of cities and fortresses. The city site evaluator needs to be extended to account for qualitative terrain information. Effective use of terrain masses, for example choke points or terrain walls, can significantly impact the game. There are many cases where qualitative terrain information would suggest city sites that excel at military strategy, but have poor access to resources. In these cases, fortresses should be used to allow for the best military strategy without compromising the civilization's ability to harvest resources with effective city placement. As section 5.7.1 describes, I have implemented code that checks for choke points. Extensions to the city settling agent would use this code along with code that identifies other terrain attributes to integrate qualitative terrain attributes into the site evaluation process. The extension should provide the opportunity for creating fortresses instead of cities if this decision makes the most sense for the civilization. A third option would involve settling a city and managing it as if it were a fortress. For example, keep the city population low, build all possible defensive buildings in the city and keep the citizens away from tiles that the "real" cities are harvesting.

7.0 City Management

7.1.0 Citizen Placement

Citizens in a particular city may be allocated to tasks that support the civilization's needs. One important job for citizens is the worker. The worker harvests the tile resources every turn they remain on a particular tile. Later in the game, citizens may be given one of the specialist tasks: entertainer, taxman or scientist. These city specialists respectively generate luxury, gold and scientific research. In the early phase of the game the city manager focuses on placing workers in configurations that keep the city and its supported units alive while maximizing surplus resources. As city and unit survival becomes secure, the city manager can make specialists to meet the civilization's needs.

In order to keep the city in good shape, the workers must generate enough food for the citizens and any supported units. Furthermore, the workers need to generate enough production for any supported units, otherwise they die. The city manager must configure a city's workers to meet these minimal constraints while striving to meet desired resource levels.

The fact that only whole workers may be used to work tiles suggests an integer programming approach. Since each workable tile has different amounts of the three

resources, the problem has inherent non-linearity. Instead of attacking this complex problem head-on, I decided to estimate the optimal solution with a staged approach. My approach involves two stages. The first stage is based on the FreeCiv city management agent described below. In the first stage, the city map is cleared and one worker is systematically placed on every open position. This algorithm branches these initial configurations until there are either no more workers or some number of configurations meet all of the minimal constraints. If the algorithm runs out of workers, the city can not meet the minimal constraints. In this case, an evaluator chooses the "best" configuration by comparing the resource vector distance from each alternative to the minimal constraints.

The resource vectors are defined along six resource dimensions that can be generated. The six dimensions include Food, Shield, Trade, Luxury, Gold and Science. The vector distance is the Euclidian distance between the two vectors. In addition to using the resource distance to find a "best" configuration that approaches the minimal constraints, the vector distance is also used by this algorithm when comparing the configurations to the desired resource levels. Vector distance comparisons are quite good when looking for configurations that maximize a resource or balance some number of resources. However these comparisons don't always produce balanced results when the goal is a small amount of one resource while maximizing another resource. A configuration's ability to harvest the maximized resource carries a stronger weight in this distance calculation than the smaller resource goal. In some situations, the configuration practically ignores the smaller resource goal. One approach to this problem involves inflating these desires a little.

A second approach allows the city manager to require a small amount of desired surplus for a resource. To implement this approach, I added a minimal surplus parameter to the function call. Minimal surplus is for desires that must all be met before attempting the dreamed desires. Without this distinction, it was too easy for the configurations to focus on the dreams while ignoring the required portion of the desires. The minimal surplus values are added to the minimal constraints at the beginning of the configuration finding process. For this reason, it is important that the minimal surplus be reserved for requirements that are reasonable expectations. Otherwise the newly defined minimal constraints may be unachievable, which can lead to configurations that don't meet the real city support requirements.

Configurations are pruned from the search using a technique found in FreeCiv's city management agent. When two configurations use the same number of workers and one of the configurations yields resource values greater than or equal to the other for all three resources, the first configuration is definitely better. This pruning step bounds the search enough to make it tractable.

When configurations exist that meet the minimal constraints they proceed to stage two. In the second stage, the city configurations are expanded in an attempt to meet the manager's resource desires. The desires are set with the philosophy that each "extra" worker should generate resource points for each of the resources in a desire-weighted

proportional manner. This desire vector essentially defines the relative desire for each of the resources. For example, a city manager might request that each "extra" worker generate 1 food point and 6 production points. The best configuration probably will not find tiles to work that individually yield resources in these proportions. The goal is to have the aggregate resource proportions line up with the city manager's desires. In order to have a city desire resource vector for the distance calculation, the desires are multiplied by the number of remaining workers and added to the minimal constraints.

The second stage of this algorithm expands the city configurations by exhaustively placing all of the "extra" citizens in all of the combinations of remaining potential positions. Each expansion is evaluated using the resource vector distance from the expanded configuration to the reified desires. The configuration finding code automatically instantiates the worker configuration that best meets the city needs and desires.

Although the above discussion focuses on worker placement, the configuration finding code has the ability to weigh decisions regarding city specialists. The desire for city specialists is conveyed by setting values in the desire resource vector for gold, luxury and science. After minimal constraints are met, the configuration finding code reifies trade into these resources and uses a combination of reified trade and specialists to meet city desires.

Instead of handling the complexity of city happiness in the configuration finder, the code starts by inspecting the current city happiness. If the city has a happiness problem, the configuration finder creates enough entertainers to resolve the problem. The remaining citizens may be assigned tasks to meet the city's needs and desires.

The QRG FAP uses the configuration finder at the beginning of each turn. I have set the city desires manually. The minimal constraints are automatically calculated based on the city population and supported units. I drive the city to grow some and produce some with minimal food and shield surplus of 1. The city desires are set to maximize the balance of all the harvested resources (food, shield and trade). In my experiments, the configuration finder code has found worker configurations that I would have chosen based on the stated goals.

7.1.1 The FreeCiv Approach

FreeCiv's city management agent (CMA) was designed to be a helping agent for human players. The interface requires an input of the minimum resources the configuration should generate and the surplus resources desired. If the CMA cannot meet the minimum and surplus resources, it notifies the human player of the problem and stops managing the city's workers. This behavior is not ideal in the case where the "human" player is an AI player. The alternative would either involve recursively calling of the CMA while incrementally changing the surplus values or settling for configurations that meet the achievable surplus but don't always use the remaining workers effectively.

For this reason, the first stage of my configuration finder is like the CMA and the second stage maximizes the desired resource values in a proportional manner. With this second stage, the QRG FAP player can set lofty goals and the best configuration match will get instantiated. The one caveat with this approach is that the goals must be lofty. The distance measure will penalize configurations that exceed the goals in the same manner that it penalizes configurations that fall short of the goals.

7.1.2 Future Work with Citizen Placement

The current configuration finder code builds a worker and specialist configuration from scratch each time it is called. As I will discuss shortly, I have generated FIRE rules that identify situations where the AI player has an extra entertainer (Elvis) or needs an extra Elvis. It would be useful to extend the configuration finder so it can make these types of incremental changes to the worker configuration without re-configuring the entire city. Incremental updates may not be optimal due to the non-linearity associated with worker placement. However an approximation might prove good enough.

Another piece of code that is needed to interface with the configuration finder is a desire setting agent. This agent would ideally look at the city type, civilization goals and city goals and set resource desires for the city. The tricky part of this agent will involve the tradeoff between setting goals high enough to drive the good configurations without swamping the desire in only one of the desired directions.

There will be many cases where cities share workable tiles with one another. The current configuration manager has no way of knowing about sharing. The tiles are worked in a first come first served basis. If a tile is really good, the first city to work it will probably keep it. There may be times where a worked tile in one city should be set free for another city. This decision will involve many tradeoffs and might be more appropriate for a strategic level agent.

Currently the configuration manager runs for every city at the beginning of each turn. As the number of cities and workers increases, this activity consumes a lot of CPU cycles. In most cases, re-configuration of the workers is unnecessary. It would be good to extend the QRG FAP so it knows when to re-configure the workers. For example the configuration definitely needs to change when the city size changes.

The current configuration manager does not take into account the building and government effects for the specialists. My intuition is that these effects will rarely come into play when configuring a city. For example, one city will specialize in science and maximize scientists. Another city will do the same with taxes. In this case, the configuration management code doesn't need to know the exact value for the resource. It just needs to see the effect of the specialist placements and have the ability to compare this effect to the city desires. If this becomes a shortcoming of the configuration manager, it could easily be extended to include these effects.

7.2.0 City Plans, Needs, Goals

As I began this project, I realized that many of the city management decisions would involve consideration of constraints that enable/disable possible actions along with a mechanism for comparing and deciding between the sometimes prolific lists of these applicable actions. My approach to this situation was to split the problem into two distinct reasoning activities. In the rule-based part of this project I focused on reasoning that would generate all of the relevant plans, based on the city's current game state. Future extensions to the QRG FAP should involve work on a reasoner that will weigh the various plans and make decisions.

In order to split these two reasoning areas up, I had to create an assertion protocol for communicating possible plans to the decision maker. Plan assertions take the form:
(Plan <FC-action>)
FC-action is either a single statement or list of statements that can be easily translated into requests to send to the FreeCiv server.

Plans take a variable number of turns to complete which is captured by the assertion:
(PlanTime <plan> <number of turns>)
The plan is a statement of the above form. I have used an integer for the number of turns. I should point out that there were several cases where I just used numbers in relations, rather than predicates that define the numbers. For example I could have used (Turns 2) to define the number of turns in the above statement. I decided that the PlanTime relationship gives enough information about the integer. This decision has the benefit that it reduces the search time when matching relationships in FIRE.

Plans are usually generated to meet some stated need.
(PlanNeed <plan> <need>)
I created two need predicates for city happiness.
(CityNeed <city> <resource> <amt needed> <weight>)
(CityBuildNeed <city> <resource> <amt needed> <weight>)
I felt that the distinction between these two predicates was important enough to create them both rather than combining them into one assertion.

Plans have costs and benefits.
(PlanCost <plan> <resource amt> <resource>)
(PlanBenefit <plan> <resource amt> <resource>)
In cases where plans generate more than one cost/benefit, I generate multiple statements rather than putting lists into these predicates. The costs and benefits of these plans are given in terms of the per turn value. In many cases, the cost associated with performing an action in a city is weighed via the time to complete the plan.

Although deciding between these plans will be complicated, these statements should give the planning decision maker everything needed to make the appropriate decisions.

I also created a protocol for communicating city goals. I have envisioned a strategic civilization manager that would decide what types of resources/actions it needs from each

of the cities to move the civilization in the appropriate way. These decisions will get communicated via a goal relationship.
(CityGoal <city> <resource> <weight>)
Each game resource/activity will be metered in this manner. For this project, I created the goals for Growth (which combines food and population), Shield, Gold and Science. I am sure that more of these goals will be created in the future, but these provided enough information to limit the plans generated to "reasonable" plans. They also allowed me to generate actions in some of the rules, because the decision between plans was clear after considering the goals.

7.2.1 Future Work with City Plans, Needs and Goals

This section of the QRG FAP player defines and implements a plan generation framework and knowledge representation. The obvious extension involves creation of the decision making agent that compares the plans and makes a decision. When designing this agent, it might be helpful to convert the rules-based approach for plan generation to a query-based approach. Instead of asserting all of the possible plans in the TMS, the decision maker can query the TMS for possibilities, make its decision and make assertions as needed to implement the plan. A plan execution monitor could also be incorporated into this system. This sub-system would compare the game state with the plan and initiate re-planning activities when the plan became jeopardized in some way.

7.3.0 Happiness

City managers need to make sure the people are happy, or at least happy enough. As citizens become unhappy, bad things start to happen. Examples include: reduced productivity, increased crime or revolution. The basic measure for happiness is the comparison between unhappy and happy citizens. If the number of happy citizens is greater than or equal to the number of unhappy citizens everything is great. Lots of game factors control citizen happiness. The QRG FAP has a couple of methods for maintaining a positive city happiness state.

Happiness can be managed in a few ways. Workers can be changed to the specialist type, Elvis, to entertain the citizens. Certain building improvements will also impart happiness to the citizens. Military units can impose martial law. The city can build a settler/engineer, which reduces the population and city overcrowding thus improving happiness. And finally, the civilization manager can affect happiness by creating more luxuries on a civilization wide basis or changing the government type. I focused my efforts on the first two methods of happiness control. However, the happiness needs and plans asserted in the TMS are certainly available for other reasoning activities.

As described above, the city configuration finder will handle disorder in a city by adding an extra Elvis before assigning the remaining citizens to more productive tasks. Although this solves the city's current problem, it is only a short term solution. In addition to this stopgap measure, I have created a series of rules that generate plans for resolving city happiness issues.

There are a couple of short term happiness rules that mimic the city configuration finder's solution to the problem. When there is an immediate happiness shortfall, an Elvis is created. Plans are suggested to convert any city specialist or worker to this Elvis. The appropriate plan should be chosen based on the city goals and the plan costs. Depending on the implementation, either the configuration finder or these rules can handle short term happiness problems.

The reasoner also looks at long term solutions to city happiness. The reasoner has three rules that generate plans to build luxury generating improvements. Two proactive rules try to anticipate a happiness need and propose city improvements before happiness is a problem. A "dire straits" rule increases the weight of the building request, because we are either reducing our food stock or starving citizens to keep the city happy.

The rules that generate city improvement plans handle a wide variety of cases. I was able to abstract these rules so they search for improvements that meet the resource need (in this case, Luxury). The rules are huge, because they only propose improvements that are possible at this point in the game. The complexity of the rule pattern matching allowed me to make these rules as general as they are.

The Temple, Cathedral and Coliseum actually make a certain number of citizens one step more happy. I abstracted this effect and converted it to the generation of an equivalent number of luxury points. While some building improvements yield an absolute number of luxury points, others yield a percentage improvement to the city. In order to calculate the plan benefits in these two different ways, I had to create two separate rules. When the next luxury producing building is unattainable due to lack of scientific research, a rule asserts a request to research the technology.

The city manager regards Elvis' as parasites (i.e. a necessary evil that steals from the civilization's ability to produce food, shield, trade, science or gold). Whenever a city has Elvis' the reasoner proposes plans to replace them with buildings that will satisfy the luxury need. Another reasoning rule that attacks this problem is the rule that makes sure the city isn't too happy. If we can convert any Elvis to a productive citizen, we create a city action request to place the Elvis.

7.3.1 Future Work City Happiness

As stated in the future work section for city plans, needs and goals, these rules generate plans and requests in the TMS. This might be better implemented with a query interface that feeds a decision maker. The decision maker also needs to pick a plan and implement the plan. This work is an obvious extension to these happiness rules.

In order to reduce the size of the building request rules, they could be broken down into two rules. One rule would maintain the list of buildings that the civilization can presently build. The second rule would find the buildings in this list that would help the city with its happiness problem. The buildable improvement list would be useful to any game agent

that needs to make city build plans and decisions. A similar list would be useful for buildable unit types.

Happiness can be affected at the civilization level by setting the global trade conversion weights. This control has far reaching impact on the civilization and changes must be considered carefully. The FreeCiv AI uses this technique to induce rapture in a city. The QRG FAP player should be extended to make decisions regarding the global rates. The number of Elvis' in each of the cities will provide input to the luxury part of this decision. Other factors will be based on the civilization's need for scientific discovery and gold. This reasoning effort will require information from many areas of the game AI. Wonder building should also be considered as a long term solution to these problems.

7.4.0 Tile Improvements

The terrain of the game can sometimes be altered to help the civilization attain its goals. The settler and engineer units make the terrain changes happen. I'll use the generic term settler in this section of the paper to denote either of these units. A series of rules suggests plans for making these improvements.

In some cases, terrain tiles can be improved by adding irrigation, farms, mines, roads and railroads. Sometimes, the terrain type may be altered. And finally, pollution or fallout (generated by the city and deposited on city map squares) may be cleaned up. Settlers are called upon for all these tasks. Each of these tile improvements will improve a city worker's resource yield from that tile in specific ways. The cost associated with settler work was difficult to develop. Instead of a cost, I have notified the plan decision maker that the need is Settler work. I have identified the time required to do the activity. The decision maker will use this time as the plan cost measure. Unfortunately some of the quantities associated with these activities were hard coded into the game code rather than sent to the client. In these cases, I had no alternative but to hard code the values into my rules.

The simplest tile improvement decisions involve fallout and pollution. In these cases, all of the tile's resource yields are dropped by 50%. So the rules generate plans to clean up the tiles along with the resource benefits of this action. In a similar fashion, the rules generate plans for irrigating, mining, making roads and making railroads. These rules account for city goals where applicable. The reasoner only generates these plans if the improvement will yield a resource gain. So roads will not be generated to connect two distant cities. In the case of railroads, the rules will generate the plan to both build the required road and then a railroad on the tile when necessary.

7.4.1 Future Work Terrain Improvements

As with the aforementioned city plans, this work needs extension by adding a decision maker. In this case, the decision maker needs to choose tiles to improve and assign settlers to the task if they are available. In some cases, the decision maker must request that a settler be built. These requests would be passed to a reasoning agent that manages

settler improvements over the entire civilization. Alternatively this could be done at the city level, where a settler would be built to make improvements as needed. A key point is that improvement making doesn't usually take long. What should the settler do after making all of the desired or possible improvements in this one city? The settler could move to another city and either make tile improvements there or join the city, it could go to a proposed city site and make a city or it could go back to its home city and wait for a new need to arise, or it could disband. These alternatives suggest that there may be several possible plans that can be used depending on the specifics of the situation. At some level, a reasoner will need to generate/instantiate a plan for settler work. I didn't do any reasoning about these plans, but aimed to provide enough information so a reasoner could generate these types of plans.

Similar rules could be generated for terrain transformations. Since transformations take a lot of time and usually are irreversible, these decisions need to be weighed more heavily than tile improvement decisions. Terrain transformation can be used to improve the resource balance available to a city. For example a forest can be created to yield more shield resources to a city. The rules for generating possible transformations should be driven by specific city need to focus plan generation efforts.

In addition to the resource improvements that come from road building, the roads and railroads reduce movement cost. A military strategy agent or infrastructure agent could generate plans and make decisions regarding potential roads from one tile position to another.

Sometimes a worker is on a tile that cannot be improved. However, there is another tile available to the worker that could be improved and with this improvement; it would yield better resources to the city. For example a worker might be on a grassland tile that cannot be irrigated. There may be another grassland tile available to the worker that could be irrigated. It would be helpful to extend the QRG FAP to notice these situations, make the improvement and change the worker location to the improved tile.

7.5.0 City Worklist

Cities use their per turn production yields to support units and build things: building improvements, units or gold. The production items are controlled via the worklist. A city worklist is the city's sequential plan for building these types of things. Building improvements and wonders provide various benefits at the city and civilization level. Each building has one or two specific effects. Within the game's list of buildings, there exists some building that can affect almost any aspect of the city and/or civilization. A city manager must reason with game information along with the goals for the city and civilization when deciding on buildings for the worklist. If the civilization needs money, or doesn't have the need for a building or unit, it can convert its production into gold via the Coinage build item.

The QRG FAP player currently builds the best "buildable" defensive unit when a city is initially settled. It then places Coinage at the end of the worklist. At the beginning of the

turn, the city manager looks for cities that are building Coinage. It then checks to see if a settler is needed for a potential city site and if this settler can be built without disbanding the city. When both conditions are met, a game request is made to build a settler.

I described the city manager's ability to make building improvement requests for happiness above. In addition to these rules, I have created rules to suggest building improvements for population growth and science. A city's population needs infrastructure to grow beyond certain thresholds defined by the game's rule set. FreeCiv has a threshold for both aqueduct size and sewer size. There are rules that trigger on these thresholds along with the city's growth goals to generate plans for these building improvements. If the required technology is unknown, requests for research are generated. Unfortunately, the game is somewhat hard coded for these improvements, so my rules are also hard coded. It would be easy to add assertions for the hard coded thresholds and change the rules so they use these more flexible assertions.

The three buildings that improve science can be represented in a format that allows building improvement plans to be generated via one rule. Like the happiness building rules mentioned earlier, this rule only generates plans for improvements that are currently possible. This rule also limits these plans to cities that have a science city goal greater than 6 to avoid nuisance plan generation. As above, technology requests are generated when pertinent.

7.5.1 Future Work with City Worklists

There is an obvious need to extend the city manager to consider other building options. As I stated earlier, the city type information could be used to weigh building improvements and wonders. Strategic wonder building would be a big win for the QRG FAP. A military agent might request building units for combat. Other agents would weigh decisions regarding non-combat units like transports and settlers. Somehow these needs should be conveyed to the city manager and prioritized.

More complicated reasoning is required when evaluating food and production building improvements. These buildings have effects controlled via the tiles worked by the city. Calculating these effects will probably involve functions that operate on the city map in the lisp representation of the game. The AI player should be extended to evaluate and build these types of buildings.

8.0 Summary

The QRG FAP fairly plays the initial expansion phase of FreeCiv in a limited, but successful manner. It plays the game strategically due to the use of several different artificial intelligence programming techniques. Qualitative terrain representations and influence maps are used with an A* path finding algorithm to strategically explore the world. Weighted utility functions guide city settling. Linear approximations of non-linear integer programming help configure city workers. A Logic-based Truth Maintenance

System along with a set of reasoning rules suggests plans for resolving or avoiding many city management issues.

Additionally, the system provides a Lisp API to the game that maintains the entire game state in a Lisp image and a TMS. This system provides many opportunities for extension into other areas of the game. Many of these opportunities have been explained in this report, but many remain open areas for research. I have a great appreciation for the knowledge I have gained during this project and I hope this system will be used by others.

9.0 Acknowledgements

I would like to thank several people for their help and support during my graduate studies. First of all, thanks to Ken Forbus for believing in me and mentoring me throughout this project. Special thanks go to my grandparents James and Elloween Houk for their hard work and generosity that made this pursuit possible. My parents Jim and Ninette Houk have helped me in countless ways get to this point in my career. Leslie Dorfman has given me much of the support I needed to push through and excel in my studies. Samuel Houk is too young to realize the effect this pursuit had on his life, but his constant request to see my game play provided spark when it was needed. Finally, thanks to everyone in Northwestern University's Computer Science Department. They have provided a challenging, stimulating and wonderful learning environment.

10.0 Literature Cited

1. Pottinger, Dave C. 2000. Terrain Analysis in Realtime Strategy Games. In Proceedings of Computer Game Developers Conference 2000. Retrieved from Studio Programmers Journal, Ensemble Studios http://www.ensemblestudios.com/news/devnews/terrain1.shtml
2. Forbus, K., Mahoney, J., and Dill, K. 2001. How qualitative spatial reasoning can improve strategy game AIs. AAAI Spring Symposium on AI and Interactive Entertainment, March, 2001.
3. Tozour, P. 2001. Influence Mapping. In M. Deloura (Ed.), Game Programming Gems 2. Hingham, MA: Charles River Media, Inc., pp. 287-297. Excerpt retrieved from http://www.itee/uq/edu/au/~penny/strategy.htm
4. Woodcock, S. 2002. Recognizing Strategic Dispositions: Engaging the Enemy. In S. Rabin (Ed.), AI Game Programming Wisdom. Hingham, MA: Charles River Media, Inc., pp. 221-232. Excerpt retrieved from http://www.itee/uq/edu/au/~penny/strategy.htm

Appendix A

	Experiment #2		Experiment #3	
	QRG FAP	**FC Explorer**	**QRG FAP**	**FC Explorer**
Republic	37	42	41	43
Goods Left	4	4	4	6
Explored	50	55	46	38
Settled	80	87	72	74
FV 1%	54,896.25	48,139.40	35,391.08	36,972.37
FV 5%	297,868.34	258,624.66	141,895.86	151,567.48
FV 10%	6,133,385.13	5,043,287.92	1,549,475.69	1,642,976.80
	Experiment #4		**Experiment #5**	
	QRG FAP	**FC Explorer**	**QRG FAP**	**FC Explorer**
Republic	49	48	44	46
Goods Left	2	1	7	8
Explored	45	37	41	39
Settled	64	64	67	73
FV 1%	25,533.32	25,194.25	38,157.91	31,556.91
FV 5%	79,622.87	78,976.76	136,341.76	123,388.40
FV 10%	555,577.25	549,607.91	1,281,204.90	1,322,022.05