©Copyright 2007, DigiPen Institute of Technology and DigiPen (USA) Corporation. All rights reserved.

RTS AI Wall Building Research and Improvements

BY
Abhishek Chawan
M.S., Computer Science, Digipen Institute of Technology

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the graduate studies program
of DigiPen Institute of Technology
Redmond, Washington
United States of America

Spring
2008

Thesis Advisor: Dr. Dmitri Volper
DIGIPEN INSTITUTE OF TECHNOLOGY

DIGIPEN INSTITUTE OF TECHNOLOGY

GRADUATE STUDY PROGRAM

DEFENSE OF THESIS

THE UNDERSIGNED VERIFY THAT THE FINAL ORAL DEFENSE OF THE

MASTER OF SCIENCE THESIS OF ABHISHEK CHAWAN

HAS BEEN SUCCESSFULLY COMPLETED ON _____

TITLE OF THESIS: RTS AI WALL BUILDING RESEARCH AND IMPROVEMENTS

MAJOR FILED OF STUDY: COMPUTER SCIENCE.

COMMITTEE:

Dmitri E.Volper, Chair	Xin Li
Ken Meerdink	Michael Moore

APPROVED :

Xin Li 3/ 21/2007 Graduate Program Director	Matt Klassen 3 / 21/2007 Associate Dean
Samir AbouSamra 3/ 21/2007 Chair of Computer Science	Claude Comair 3/ 21/2007 President of DigiPen Department

THE MATERIAL PRESENTED WITHIN THIS DOCUMENT DOES NOT NECESSARILY REFLECT THE OPINION OF THE COMMITTEE, THE GRADUATE STUDY PROGRAM, OR DIGIPEN INSTITUTE OF TECHNOLOGY.

INSTITUTE OF DIGIPEN INSTITUTE OF TECHNOLOGY

PROGRAM OF MASTER'S DEGREE

THESIS APPROVAL

DATE: 3RD MARCH 2008

BASED ON THE CANDIDATE'S SUCCESSFUL ORAL DEFENSE, IT IS RECOMMENDED THAT THE THESIS PREPARED BY

Abhishek P. Chawan

ENTITLED

RTS AI Wall Building Research and Improvements

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE FROM THE PROGRAM OF MASTER'S DEGREE AT DIGIPEN INSTITUTE OF TECHNOLOGY.

Dmitri E. Volper
Thesis Advisory Committee Chair

Xin Li
Director of Graduate Study Program

Matt Klassen
Dean of Faculty

The material presented within this document does not necessarily reflect the opinion of the Committee, the Graduate Study Program, or DigiPen Institute of Technology.

Table of Contents

Abstract

As we know even in history all the great empires have their capital cities and they needed to protect them. One cannot ignore walls they build to protect their empires. The Great Wall of China is the perfect example to show how important walls were and role they played in protecting from the enemy. Now comparing that with current Real Time Strategy games we can see the similarity in the role of wall. Even Though Real Time Strategy games are simulations one needs to understand that building walls is not easy task.

Wall building is problem faced by game developer since the start of Real Time Strategy games. RTS wall should provide good challenge for an average human player on a random, unexplored map without unfair advantage which makes the task difficult for AI. Techniques currently used in industry like terrain analysis, predefine wall structures, convex hull etc. are not flexible enough to be used in variety of different types of games and maps. Most of them produce good wall by doing some trade off in ideal wall conditions.

In this thesis we propose a new approach for wall building which will allow us to build a wall which is strategically more useful and flexible than existing one; without taking any unfair advantage. With this approach we implemented an algorithm that allows better use of natural barriers and resources. Achievements of this algorithm are explained with the help of result comparison with existing method.

Chapter I: Introduction

In this chapter we will take an overview of Game AI. Also compare Game AI with Academic AI. Next we will cover what Real Time Strategy games are and some history about them.

Artificial Intelligence and Computer Game

Artificial Intelligence (AI) has been integral part of computer games since the birth of computer games and AI basically fits into game play part of game. Graphics engine, physics engine and UI are core parts of any game apart from these there is a part of game which handles the computer player. This part of game is known as Game AI. Game AI has been a standard feature of games -especially with developers' emphasis on single-player games, which today still represent the majority of released titles. Game AI also making its way into multiplayer games where a person plays with other human player with the help of Computer Player. Even though the main control stays with human AI player can mimic or make some decision to help human player.

At the start of gaming era game AI was not AI rather just a simulation of AI and still most of game doesn't try to implement AI. This was the case because AI implementation takes lot of processing power. Now a day's computer processing power is exponentially expanding thus we can see more and more complex AI techniques implemented in games.

There is difference between computer game AI and Academic AI. Academic AI always tries to implement the AI which measure up to human and accuracy is main agenda for its development. On the other hand Game AI can be defined as set of Technique or Algorithms which are used for creating illusion of intelligence in non playing character (NCP).

Game developer's increasing knowledge of academic AI is filling the gap between Academic AI and Game AI. Still both are viewed as distinct subfields of AI since the ability of game AI to cheat to achieve best result is still a major distinction. Example of cheating could be; keeping track of all the statistics even though in actual world it is not possible. Sometimes cheating becomes a major part of Game AI. Cheating is useful in changing the difficulty level of game. For example in RTS game computer player get access to the data of human player as difficulty of game increases. Now computer player can keep track on all human moves which is very hard to do in normal circumstances. Based on that computer player utilizes its resources more efficiently. This way Game AI cheats to become better than human player.

The game industry is starting to recognize that sophisticated AI that could enhance the entertainment value of their products and consequently increase revenues. This way cheating can be minimized. Already, many computer games are marketed based on the quality of their AI like Halo 3.

Real Time Strategy Games

What are Real Time Strategy (RTS) games? It is Game Genre where everything is real time and involves lot of strategy and planning. Real time means you should be quick to think and respond, everything in these games is real-time. There are no turns for players; every movement each player is doing something which might affect the entire game or certain part of game. Some important concepts related to real-time strategy includes tactical combat and fast pace action. Typically RTS games are war base games where you need to create your base and improve it to make your own army and society. Main Aim of these games is to conquer the opponent but this may very game to game.

In most RTS games, the key to winning lies in efficiently collecting and managing resources, and appropriately distributing these resources over the various game elements. Typical game elements of RTS games include construction of buildings and walls, research of new technologies and combat. This structure is such that player has to maintain economy aspect and a combat aspect of the game balanced to win. Player has to divide its attention on these two aspects since one requires other to get good output. Each player in an RTS may interact with the game independently of other players, so that no player has to wait for someone else to finish a turn. This lends the genre well towards multi player gaming, especially in online play, compared to turn-based games.

If look back to the history of RTS games Dune II is considers being a first RTS game. The genre name was invented by Westwood's Brett Sperry. Before any game under RTS genre is released there were some games similar to RTS games but named as war games. Due to complexity of war games they were not up to the mark with compare to other genre. So Brett Sperry decided to create new genre so that people get attracted to

these games. However, many serious strategy gamers disagree with the use of the word strategy in RTS, arguing that RTS games are nothing more than a cheap imitation of turn-based games because of the tendency of RTS games to devolve into 'click fests' in which the player who is faster with the mouse generally wins, because they can give orders at a faster rate (Geryk 1998).

After Dune II lot of RTS games were released and become successful. In 1994 Blizzard games released a game called Warcarft which was based on fantasy world. Warcraft was a successful game but Warcraft II released in 1995 had the biggest success rate any RTS games seen till that point. In years to follow game industry seen lot of RTS games which were highly successful e.g. Age of Empire, Empire Earth. Improvement in processing power and memory allowed RTS games to stretch their limits which we can see in recent games like Command and Quanqcer II and Supreme Commander. Starcraft will be the longest running RTS game till now and Starcraft II is anticipated to be the biggest hit in RTS genre.

Figure 1: DUNE II was the first RTS game ever.

RTS Game AI

AI always has been the major part of any RTS game. Other game genre heavily relies on graphics and physics on the other hand RTS garner gives more importance to Game AI. Core part of any RTS game is to keep NPC in competition with human player. Planning for success in RTS games is heavily depending on two things Tactics and Strategies. Both are part of Game AI and requires good amount of research to implement it. By definition tactics cover small-scale interactions such as handling units and capturing cities where as far as strategy goes it covers everything but at a higher level. Strategy AI will not care how to gather resource of fight with enemy; rather it will take decisions like we need to gather resource or attack city.

RTS Game AI normally works with layers since there are lot of tasks which will be handle by Game AI at the same time. Each layer works as a module so that module can be used for human player e.g. path finding can be used for computer player as well as human player since human player actually do higher level decisions. RTS Game AI has to make sure that it gives time to each part of AI; wall building can be one process that Game AI should handle but at the same time it finds that there is an opportunity to attack opponent then he should make both processes work at same time.

RTS Game AI should be capable of working with human player as an opponent as well as an alliance. When human decides to play multiplayer game and it takes computer player as a partner computer player should follow human lead.

Due to this complex nature of Game AI RTS game tried various techniques to overcome these hurdles. There is lot of different techniques used from simple state machine to complex multilayered implementation.

Ramsey [4] proposes a Multi-Tiered AI Framework, where different levels of managers control the AI, this allowing ‘grand strategic decisions’ to be made by AI at a higher level, which then has the corresponding manager execute the task. EMPIRE EARTH by Stainless Steel Studios, arguably the game with the most successful RTS AI up to now, decomposed the AI into the following managers:

- **Build manager**: responsible for placement of structures and towns. Most buildings have requirements on where they can and cannot be placed.
- **Unit manager**: keeps track of what units are in training at various buildings, monitors the computer player’s population limit and prioritizes unit requests.
- **Resource manager**: responsible for tasking citizens to gather resources in response to requests from both the unit and build managers. This component is also responsible for the expansion to new resource sites.
- **Research manager**: the research manager examines technologies and selects them based on their usefulness and cost.
- **Combat manager**: responsible for directing military units on the battlefield. It requests units to be trained via the unit manager and deploys them in whatever offensive or defensive position is beneficial.
- **Civilization manager**: coordination between build, unit, resource and research managers. It handles player expansion, spending limits, building and units upgrade.

All these managers work with each other as well as work individually to achieve their personal and global goal. This approach gives flexibility to Game AI to perform various tasks at same time.

Chapter II: Problem Description

This Chapter takes brief overview of the problems faced by RTS AI. Chapter also explains the wall building basics and problems faced by the wall building.

RTS game AI problems

In previous chapter we have seen that AI for RTS games is the most complex one in all kinds of games. RTS game requires layers of AI to handle it properly; only path finding and state machine is not going to solve the problem. After playing with computer Player two three times human can recognize the pattern of AI player and can beat him without any trouble. In games like Age of empire people play with three computer players opponent with hard level and still able to beat them. This monotones nature of RTS game makes it very boring in single player mode. Other than this there are issues with AI which causes major problem in building go NPC. Some of problems are explained below.

Terrain Analysis: For a human terrain analysis is very simple on broader level. Human can analysis the terrain and decide which area is good for town which is good to hide which is good to attack. These simple things are very difficult for computer player. Computer player has to go through rigorous iterations of terrain analysis to find solution to these simple problems.

Strategy making: Making a strategy is very simple for a human player he can see what his current status on army units, resources is and take decision that he should improve his town or attack enemy town. This is top most level of decision making.

Computer player requires heavy learning algorithm to get even with human player and still he might struggle.

Co-Operation: It is very hard to teach computer player how to work with human player as a partner. Computer player can not adjust to each human player with whom it plays but it need to come up with some kind of plan which can be suitable for all human players at least on some terms. Learning might solve this problem to some extend but it is very hard to maintain and implement learning algorithm in RTS.

Attacking-defending: Most of the RTS game cheats when it comes to attacking and defending. They cheat and try to maintain the same or more number of army units as human player has. This cheating makes game interesting since human player cannot make a small army and attack computer player at the start of game. Computer player faces a problem when it comes to when and where to attack. Human player can analysis the situation and change its attacking strategy based on opponents strength and location. NPC will normally go through patterns which are feed by the programmer.

All these issues make RTS AI very hard to implement. All these issues can be taken as research topics on their own. In this thesis we are researching a topic of wall building for NPC.

Wall Building

After seeing all the topics which needed to be solved for better implementation of RTS AI we decided to narrow our research to the problem which ignored by most of game developers. The problem this thesis trying to address is wall building by RTS AI. For understanding the problem we need to go over basics.

What is wall? Walls are static defenses that surround bases to slow down attackers. [2] Wall can be made up of any material which depends on game. Basic function of any wall in RTS game is to protect town from quick raids of opponent army. Normally walls are tuff to break as compare to any other structure, to breach the wall opponent will need a larger force. When wall is under attack it gives a warning to the player that opponent is planning an attack and he can take decisions depending on that.

Wall building has been an issue for RTS AI since the beginning and this problem is mostly ignored or fixed by some hacks rather than actual implementation. Wall building is very necessary when it comes to playing multiplayer but even when player plays with AI, AI should build walls to prevail over early attacks from human player. Till this day many RTS games lack this ability due to which human player take advantage of this flaw and beat even hard level AI.

One needs to understand the basic structure of wall and implementation of wall to get to the solution of this problem. As described above wall is a static object which is hard to break and separate you from the enemy.

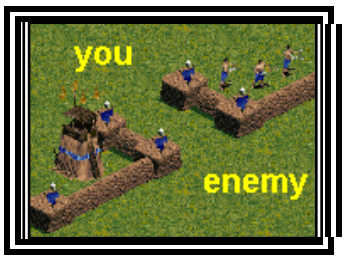


Figure 2: Wall separate enemy and you

Wall segment: normally all RTS games are based on map which can be divided into grid; even in 3d games each polygon of ground can be treated as the cell of grid irrespective to its orientation or height map. The smallest part of wall which covers a single cell of map is known as wall segment. Each wall segment is wall in itself. After passing some qualifying criteria collection of wall segments can be treated as wall.

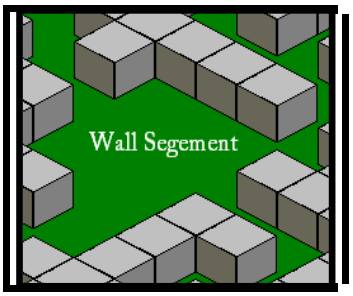


Figure 3: Each block is a wall segment

Interior Area: Any walled off area can be treated as an interior area in other words area which is surrounded by wall from all the sides is called interior area. Interior area should be continuous which means any position in interior area should be reachable from any other position in interior area. Interior area normally used for building town and military units. Interior area provides shelter from enemy attacks.

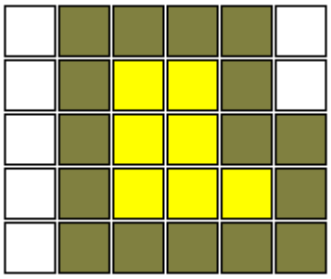


Figure 4: Light color cells are part of interior area

Exterior Area: Area which is cut-off from the interior area by the wall is called Exterior area.

Rules or criteria for good wall can vary from game to game. Some of the major and common rules are mentioned below.

- Main aim of any wall building algorithm is to minimize the resource use and maximize the interior area. Depending on game this criteria changes or varies to give more priority to one of them.
- Wall should give freedom to player and create a barrier for opponent. This means wall should be constructed such a way that any future expansion should be possible.
- Wall should make town as self sufficient as possible. In any RTS game resource gathering important since everything requires some sort of resource to build. Algorithm should also make sure that wall will cover all necessary resources which are in vicinity.
- Wall should not have any gaps or wholes which can give a bypass to opponent.

These are ideal requirements but in actual implementation have to do tradeoff with one or two criteria's. Like keeping whole in the wall is not the best solution but due to some objects like water or trade road some wholes might get created.

Aim for this thesis is to do research on existing methods and try to implement algorithm which will try to satisfy most of the criteria.

Chapter III: Techniques Needed In Wall Building

This chapter explains the algorithms which are used for wall building. All these algorithms are sub part of main wall building algorithms used by industry.

Graham's scan

Graham's scan is a method to create a convex hull from given set of points in the plane with time complexity O(n log n). It is named after person called Ronald Graham who implemented the original algorithm 1972. Basic output of this algorithm is to give a convex hull created by line joining outermost points of given set of points. This algorithm is used by some RTS games to generate convex hull which can be used as a basic structure of wall or mapping the area which needs to be operated on.

Algorithm starts with a set of co-planer points for which we need to find a convex hull. First step for this algorithm is to find the point with smallest y co-ordinate value. This operation has O(n) time complexity where n are the number of points.

Next we start to sort all other points with respect to the angle they make with X axis. Any sorting algorithm can be used for the sorting purpose e.g. heap sort O (n log n). In order to speed up the process we can find the tangent to each point angle with x axis rather than finding the acute angle they make with x-axis which is heavy on calculation. All the points are stored in an array to be used for further procedure.

Next step is to consider each point on the sorted array and find out that if moving from two previous points to current point is "Left turn" or "Right turn". If it is a "Right turn" then second last point is not a part of convex hull. As soon as algorithm encounters

the “Left turn” algorithm continues and keeps points for convex hull creation. Refer the following image to understand it visually.

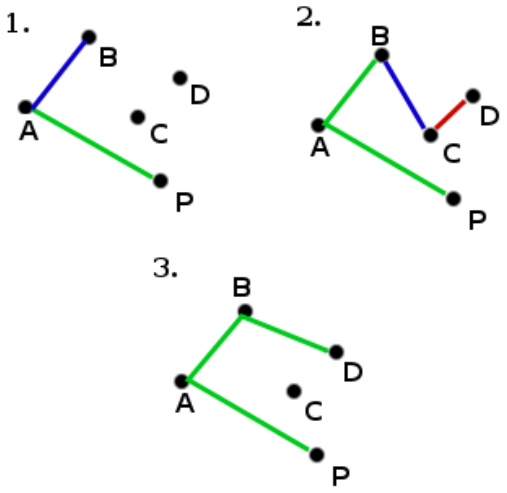


Figure 5: Image taken from http://www.answers.com/topic/graham-scan

If three points are co-linear we can discard the middle point or keep it since it does not affect the final convex hull in its shape but adds extra point.

Time complexity for this algorithm is O (n log n) even if it seems like O (n^2). This is due to each point is considered only once when each point goes back to check if any of previous points make right turn. This procedure either considers the point for convex loop, or it is removes it from the array and thus never considered again. The overall time complexity is therefore O (n log n), since the time to sort dominates the time to actually compute the convex hull.

Formal Algorithm

1. Let p_o be the point in given set Q with the minimum y-coordinate or leftmost point.
2. Let $<p_1, p_2 \ldots p_m>$ be the remaining points in Q, sorted by polar angle in counterclockwise order with respect to p_o.
3. TOP [S] = 0
4. PUSH (p_o, S)
5. PUSH (p_1, S)
6. PUSH (p_2, S)
7. **for** i = 3 **to** m **do**
    i. **while** {angle between NEXT_TO_TOP[S], TOP[S], and p_i makes a non left turn} **do**
    ii. PUSH (S, p_i)
8. **return** S

Return S will be collection of points which will create convex hull.

Flood fill

Flood fill algorithm is used also known as seed fill. Aim of the algorithm is to finding all connected objects of an array which has similar properties. This algorithm is heavily used in the graphics part of any application. We can see its implementation in any basic paint application which has bucket fill tool. This bucket fill tool utilizes the flood fill algorithm and replaces the color in certain area to other color.

This algorithm is also used in games like mine sweeper, Puyo Puyo, Luminous and Tetris. In all these games the aim is to find cells having same properties and they are connected to each other. This algorithm is mainly used for 2D games but can be used for 3D by doing some modification.

What is flood fill and how does it works? As name suggested it floods certain area of array with certain value or property. There are 2 basic types of flood fills which are 4 directional or 8 directional flood fill. These two types does the same thing except 4 way fill respects the boundaries where as 8 way fill can expel in other areas depending on boundary width.

Any basic flood fill algorithm takes 3 parameters first two are row and column and third one is the property to be used for flood that area, e.g. new color value is used as third variable in the bucket fill. Algorithm starts with the row and column value provided by the function, also it maintain a stack or queue for keeping track of visited cells.

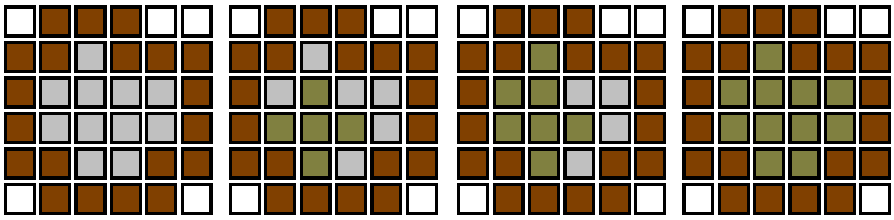


Figure 6: Flood fill used as a bucket fill

Algorithm starts with first cell and checks all connected cells which has same property as the first cell, if it is true that cell is pushed inside stack or queue depending on implementation. Now the connected cell can be 4 way connection with starting cell or 8 way connections. At the end of this iteration the property of current cell is change to target property. Now new cell is popped from the stack or queue for next iteration, these iterations continues till we found stack is empty. At the end of all the iterations we change all the connected cells properties to target property.

Flood fill is used for finding similar property area from the map such as forest area or enemy influence areas. Terrain analysis use flood fill to separate all the areas to perform analysis and set certain values for the area. Flood fill also used for path finding in some games, if map has cells with walk able property value assign to it flood fill can find out that area is reachable or not.

Greedy Algorithm:

Greedy Algorithms are simple and straightforward. They are shortsighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future. They are easy to implement and simple to use and most of the time efficient than any other approach. Most of the time greedy algorithms are used for getting solution and if possible optimize solution. Due to their shortsighted approach most of the times they do not give optimize solution

Example of greedy algorithm:

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
- A greedy algorithm would do this would be:

  At each step, take the largest possible bill or coin that does not overshoot
  - Example: To make $6.39, you can choose:
    - a $5 bill
    - a $1 bill, to make $6
    - a 25¢ coin, to make $6.25
    - A 10¢ coin, to make $6.35
    - four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution

A Path Finding Algorithm*

A* algorithm is most often used algorithm in Game AI. A* can be used for lot of purposes but in RTS game it is mostly used for path finding. Algorithm works on tree or graph to find the optimize solution.

How does A* work? A* basically traverse the graph or tree structure based on a heuristic value.

$f(n) = h(n) + g(n)$

This is heuristic function; basically it considers current cost to reach goal and cost till the current node from the starting node.

h(n) is the cost required to read the goal.

g(n) is the cost from the starting node.

This algorithm begins with a start node and adds all the nodes accessible from this node to an open list. The nodes on this list are then assigned a heuristic which is used to sort them in likelihood of providing the optimal route to the destination. The algorithm then moves the best node on the open list to a closed list. All the nodes accessible from that node added to the open list and their heuristics are calculated or recalculated if they were already on the list. This process repeats until a path to the destination has been found.

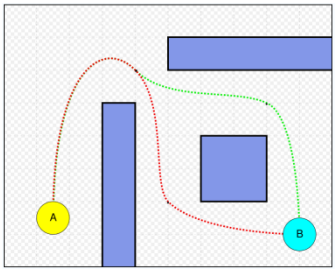


Figure 7: Path finding from A to B .

Wall building requires A* for creating gates in the wall. Whenever a wall structure is created it acts as a barrier to the opponent as well as player who created the wall. Wall restricts player's movement across the wall, to solve this player need to put some gates. Gates give freedom to player to move across the wall at the same time they stop enemy coming in.

When wall structure is complete Game AI can use brute force to put gates in the wall. This method works fine but does not guarantee that those gates will help in smooth movement across the wall. Gate might face natural barrier which effectively block the path. Sometimes brute force can put gates in wrong direction and this might lead to delay the attack.

Solution for all these problems is finding optimal path from the center of town to the opponent town. Intersection of this path with the wall will give us the best position to put gate. A* comes handy in finding optimal path that we need.

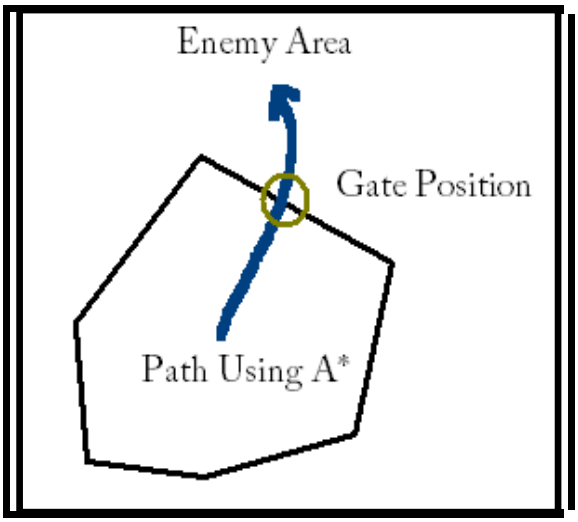


Figure 8: Finding the path from interior area to enemy area.

Chapter IV: Techniques Used by Industry

This chapter explains the techniques used by the industry and there positive and negative points. This chapter also has the detail explanation of wall building method based on Greedy Algorithm which is base for my improvements.

Predefine Wall Structure:

Many old RTS games which use to run on very low processing power computers adapted strategy that the entire AI wall will be defined before hand to save the calculation time. In this method all the maps use to have map points which were defined using map generator tool by the programmer. This was very effective process since all these points were defined by human which gives some advantage to AI.

Recently games adapted a policy to develop a wall dynamically because of extra processing power and also game requirement. Still some games use predefine structure because most of the algorithms fail to work as efficiently as human can work. This is the major plus point of this method.

This method does not work with the games which have ability to generate random maps. Since random maps are generated dynamically they do not have predefine points.

Also predefine wall structure helps human player more than AI since human player can understand the pattern or positions where AI normally builds the wall. This gives human advantage by making his moves depending on wall structure that AI is going to build.

Convex hull method:

Basic definition of Convex hull is; for a {set of points} S the smallest {convex set} which encloses all the points of set S. Let's us consider a set of points on a plane and we start with stretch rubber band enclosing all the points. Now when rubber band is release and it starts shrinking, the smallest shape when it still encloses all the points is called convex hull.

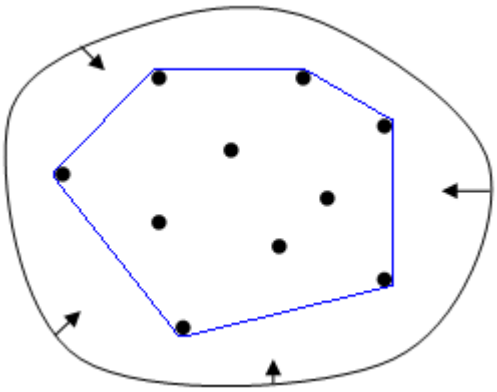


Figure 9: Convex Hull Fig taken from http://en.wikipedia.org/wiki/Convex_hull

Now consider this convex polygon is used for outlining the basic shape of wall in RTS game. Since Convex shape satisfies all the requirement or conditions to be a wall it is very useful to generate good wall structure.

Convex polygon method is used by various games. Most prominent use of this method can be seen in Empire Earth II [3]. This game makes use of convex hull method to its best and gets the decent wall structure for the AI player.

According to paper Wall is defined as a static barrier or defense system which surrounds the AI town. Ideal wall will be which surrounds the town while taking advantage of terrain and also keep space for growing. Empire Earth does not build the best wall since the game AI targets to build wall which is tight around the town so that building wall consume minimum resources.

This method starts with a box with a fix distance from the town center or starting location given to AI. Each side of the box is considered to be a wall edge and each corner will be considered as a node. Algorithm goes over each edge to check that edge is valid or not. Valid is defined as the edge which does not intersect with any buildings or farms. If the edge is invalid then algorithm finds the intersection point and creates a new node which is located at the ½ the length of the original edge from intersection point in the normal to the original edge point away from center.

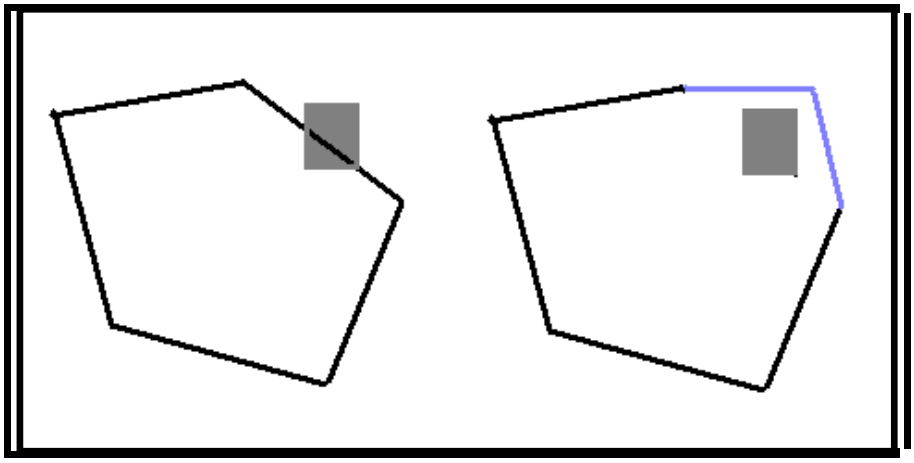


Figure 10: Addition of new node to solve collision creates two child edges

Once we step through all the original edges to check that every edge is valid or not we get lot of new child nodes and edges. Now we repeat the process till we find that all the edges are valid.

At the end when we find all the edges are valid we can see that the output is very jaggy wall plan. To get the more mature plan for wall building the next step is to smooth the rough plan which we got from the first part of algorithm.

Using all the nodes and edges which are created during previous process we create a convex hull. Graham's scan is used for getting the convex hull. Graham's scan considers all the nodes as points on 2D plane and runs the algorithm as explain previous topics. Smooth wall plan generated using Graham's scan needs to undergo edge validation to make sure that final plan do not intersect with any buildings or farms. This process iterates till we get a smooth plan with all valid edges.

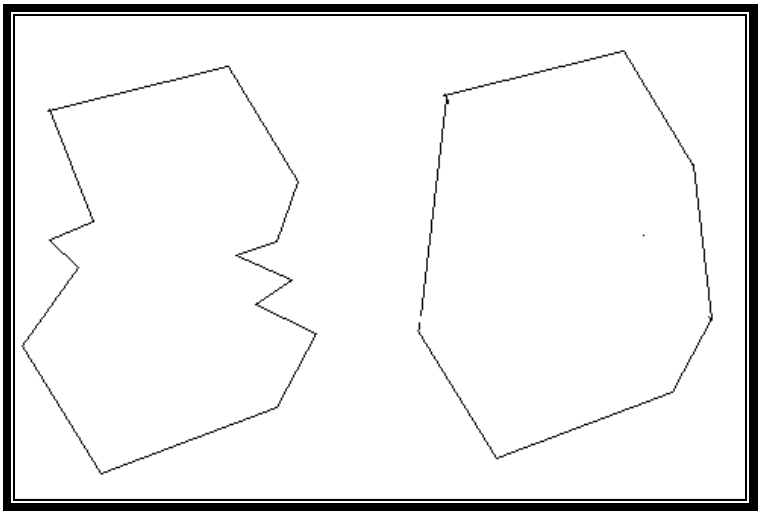


Figure 11: After applying Graham's scan we get better wall structure.

Next step is actual wall building process for this Empire Earth algorithm starts with one edge at a time since this reduces work force required and gives AI time to generate enough resources.

Problems faced by this method

- This method can iterate for long time depending on size of town.
- Algorithm can generate very costly wall plan with repetitive smoothing algorithm.
- Roads pose a particular challenge to wall plans because a road is a continuous obstacle, the usual solution of stepping away from obstacle will not work unless the road is parallel to required wall plan.

Benefits of using this method:

- This method is simple to implement and easy to tweak depending on games requirement.
- This method allows AI player change the plan dynamically depending on the situation of game. This requires modification in Graham's scan algorithm which is explained below.
    i. Since this method builds wall one segment at a time it takes long time to build whole wall. During this time other AI manager can put building right in the middle of edge making it invalid.

ii. Solution for this problem is keeping already build wall edges fixed create new convex hull.

iii. Now Graham's Scan needs some modification to run properly and generate smooth wall.

New Approach to Graham's Scan

1. Find bottom most edge node.
2. Sort all nodes at their angle from this minimum point.
3. Loop through all edges in wall plan
4. If edge is already built, move the two nodes of the edge adjacent in the sorted list.
5. Loop through sorted node list.
6. Look at nodes in groups of three.
7. While the angle between these nodes is counter-clockwise and the middle node is not on a started edge drop the middle node and use last node and two nodes before the middle as the group of three.

After applying this modification we are able to generate new plan for wall building without disturbing existing wall structure. Output of this may not be the best looking wall but it saves time and resources since we keep existing wall as it is.

According to the paper it seems that this method is good for wall building when we only think wall as a defensive structure. This algorithm may not give the best strategic advantage by wall location and shape.

Terrain Analysis:

What is terrain analysis? Terrain analysis is the analytical techniques that quantify terrain parameters (type, slope) or the effect of terrain on a particular operation. Terrain analysis is used by lot of programs to do different thing but the common goal of any terrain analysis is to gather information about any of given terrain and utilize that for achieving different goals.

Terrain analysis is probably used in every RTS game, whether you know it or not. Terrain analysis can exist in many forms from simple path finding to advanced area decomposition. Whatever method is employed, it has the single goal of supplying information about the map to various systems in the game. [3]

Computer player is the biggest customer of the terrain analysis since human player does terrain analysis on his own. Terrain analysis can provide important insight of the map for computer player; using which he can decide where to build walls, where to attack etc. etc. Good terrain analysis also provides a framework for a location-based knowledge base to the Game AI. This knowledge is useful in processes like wall building.

Implementation of terrain analysis: We need to understand all the terms which are used in terrain analysis to implement it. Terrain analysis is heavily used in Age of

empire game; we are considering this game as case study to understand this method. Terrain analysis is based on terrain properties but it can use influence map to do better understanding of terrain.

Terrain Analysis Tools: Developing a sophisticated terrain analysis system is a lot like developing a good image processing toolbox. [3] We need to cover some basic definitions to understand how terrain analysis is used for wall building.

- **Tile based maps**: Maps which are based on 2D height field are tile base maps. In this algorithm we consider XZ plane as ground and Y direction as upward direction.
- **Tile**: Single cell in tile based map is called as tile.
- **Arbitrary poly map**. A terrain map made up of arbitrary polys.
- **Area**. A collection of terrain (either in a tile or arbitrary poly map) that shares similar properties.
- **Area connectivity**. The concept of knowing how areas are connected and being able to traverse that connection network.

Influence map:

Influence mapping is a common terrain analysis tool. Influence maps are 2D arrays which are of same size as actual game map. The influence map is initialized to an initial value and then attracting and detracting influences are applied to the map based on some game-specific heuristics. Let's consider a gold mine in the map; tiles in nearby area

gets a gold influence value. The values change as we go away from the actual gold mine. Influence can overlap each other to produce stronger influence.

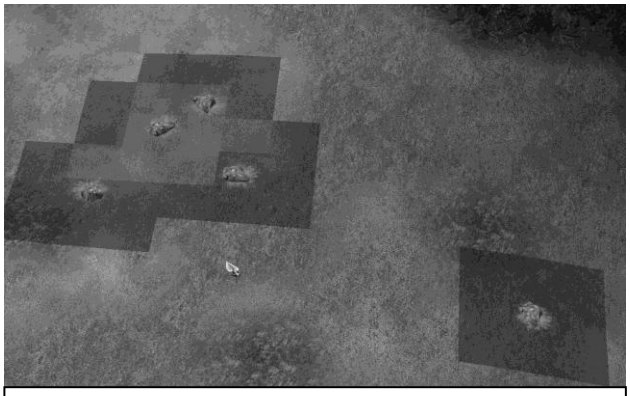


Figure 12: Influence map around gold mines.

Influence map can be a simple 2D array or can be made complex enough to have stack of values at each cell. Age of empire newer version used multilayered influence map [3] each layer represent different influence and can be used for different purposes in the game.

Wall building: Now algorithm makes use of influence map and terrain properties to divide areas into subareas. To do this flood fill is used, which is common practice in area subdivision. Once all the subareas are formed they are linked based on their neighbors and traversal feasibility. This structure is then used for wall building.

Normal way to define wall structure is to take wall building points from the subarea edge points. Now selection of subareas for wall building is based on that particular game. Subarea accuracy and size is very much important to find wall building points. Consider twisty forest passage may be a hundred meters long, but if it's made up of five subareas, then we already have our several usable wall endpoints determined for

us. Algorithm can simply take the subarea boundaries and use those as wall locations to block that passage. [3]

This way terrain analysis allows wall building on any kind of terrain even if whole map is made-up of grass field it will be divided into subareas and wall can be build by selecting proper subareas.

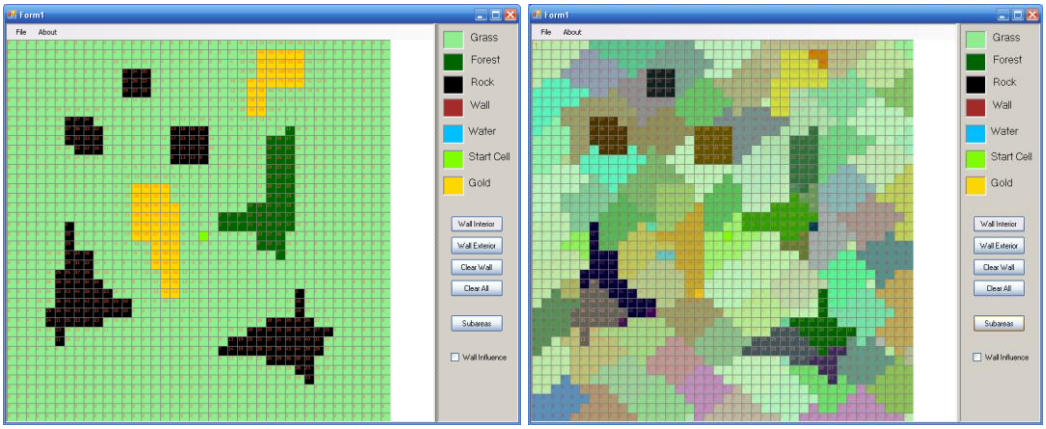


Figure 13: Map before and after subarea division

Wall building using terrain analysis is very hard to implement since one has to subdivide whole map into subareas and link them to create structure out of it. Selection of subareas heavily relies on linking of subareas. Implementation of subareas selection process (to build wall) is open to developer as per his (game's) needs. This selection of subareas can be as simple as collecting all connected subareas near players town or can be as complex as finding bottleneck within the subarea structure which we generated. One can definitely implement better subarea collection algorithm to make wall more strategic than just a defensive structure.

Greedy Algorithm

As defined earlier greedy algorithm is an algorithm that tries to find an optimal solution by using a series of locally optimal steps. [1] Using this way algorithm just sees immediate gain and hence the name. Since locally optimal solutions are easier to calculate these algorithms are faster in nature. This wall building algorithm is based on some basic definitions and rules; we need to understand those to implement this algorithm.

- **Tile based map**: Algorithm uses simple 2D tile based decomposition of map which is called tile based maps.
- **Wall segment**: Wall segment is the smallest part of a wall which covers on tile on the map.
- **Wall** is collection of wall segments. Group of wall segments need to satisfy certain criteria to be called as wall.

  Definition: Wall is a collection of segments that divides map into exactly 2 connected components.
- One of the components is called interior area – based on the location of a special point, usually the town-center. Notice that wall should never contain 2x2 squares of wall segments - it's always possible to remove one of the 4 segments without breaking the wall property, thus making wall more efficient/cheaper.

The above remark allows us to reformulate wall definition using more practical rules:

- Each wall segment has exactly two unique wall segments adjacent to it.
- All wall segments should be linked in a circular manner in such a way that if we start from one wall segments; while moving clockwise or anti clockwise we should reach the starting point never passing the same segment more than once.
- There should be at least one interior area.
- All interior areas should be connected by their edges, diagonal connections will not count.

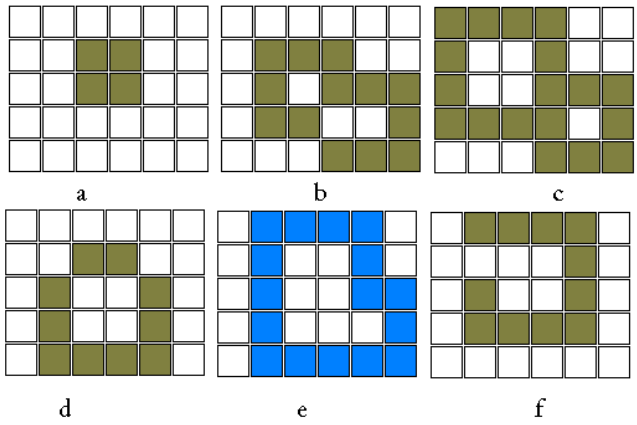


Figure 14: Different wall structures.

- 'a' is not a wall because it has no interior area.
- 'b' & 'c' are not walls since they don't have connection between interior areas.
- 'd' diagonal connection of wall is not counted as a valid connection wall.
- 'e' satisfy all the conditions so it is a valid wall
- 'f' is not a wall since there is a hole present in wall .

Building walls using greedy method:

Starting location is needed to initiate the algorithm. Starting location is a location which we need to protect by building wall around it. Approach for this method starts with basic wall around the starting location and then start growing outwards by applying series of greedy moves. **Move** in the wall building context is an action that

- removes an existing wall segment
- and then builds a wall around the space where wall was present before the removing it. There are two conditions to satisfy by move
  i. There should be net gain of at least one tile.
  ii. Wall should still meet all the criteria's which define the wall.

First approach: This approach needs a set of moves which will be used for building the wall. With given rules and conditions we can have 4 groups of wall structures which are basically made-up of three wall segments. Group 2 3 4 are rotation and flipped version of first group.

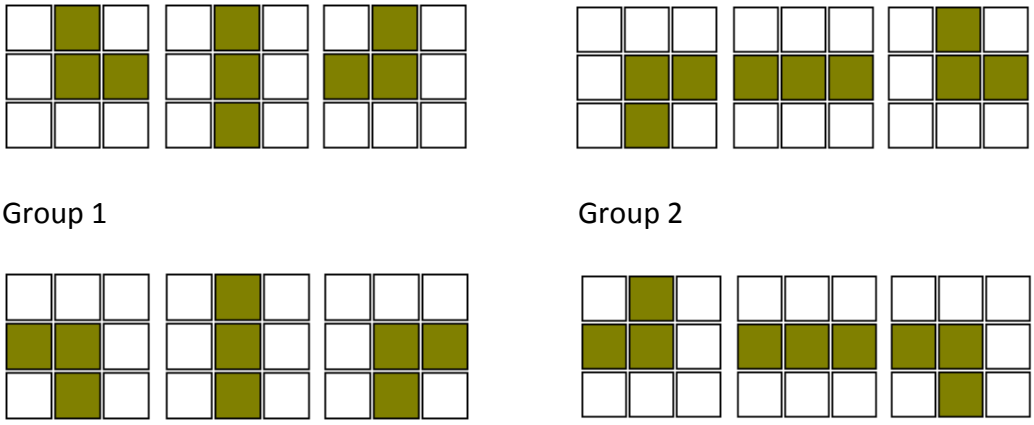


Figure 15: all the possible wall structures made up of 3 wall segments

Now in this given groups each case can produce set of outputs which satisfy the wall definition after a single move operation is performed on them.

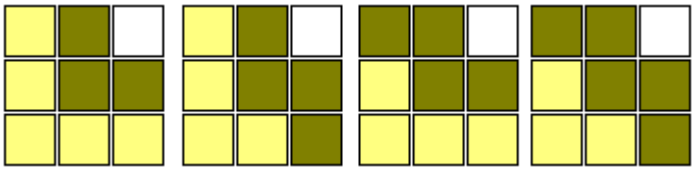


Figure 16: Group 1: Case 1 Solution set

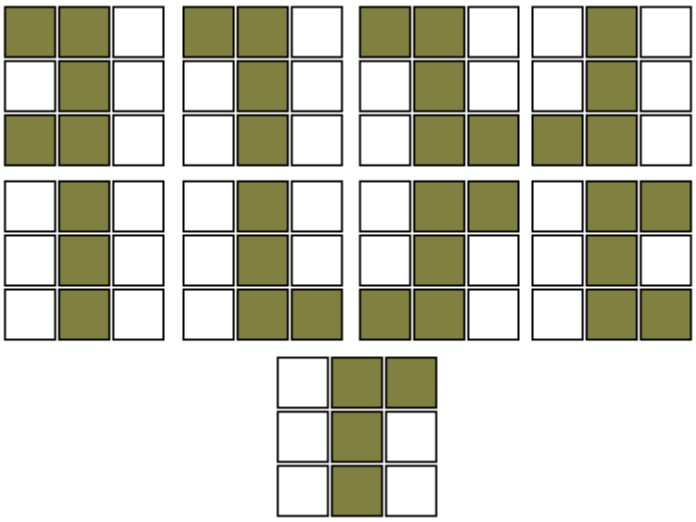


Figure 17: Group 1: Case 2 Solution set

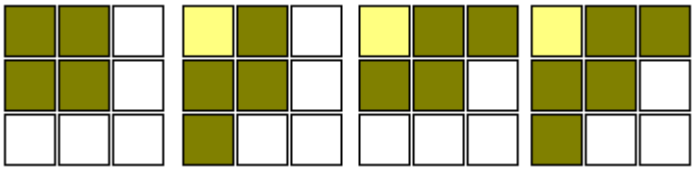


Figure 18: Group 1: Case 3 Solution set

As we can see the solution set for number of output for first group is 17 so for all four it will be 68. Even though this seems ok when we consider natural barrier which we cannot move, situation gets worst. This approach requires handling a big and complex solution set to find the wall after each move. These restrictions are applied due to tile map or grid base solution set system.

Second Approach: to simplify this we will convert this problem in a graph and apply greedy algorithm to find the solution. We need to define some terms to do this paradigm shift.

- **Node**: Node is a representation of smallest area on the map. If we consider a map as a grid then a single cell on the grid is node.
- **Graph**: A graph is a data structure consists of nodes and set of edges which join them.
- **Edge**: Edge is a link which joins two adjacent nodes, also if defines the link only if the area is walk able for a RTS unit.

Now since each node can have a wall or wall segment attach to it the problem gets converted to graph problem and restriction due to grid are no longer present.

Implementation:

Algorithm starts with the starting location, starting location can be set by other module of Game AI which makes strategic decision. In this approach move corresponds to adding a node to an inner area and creating new wall to cover that new interior node. This approach needs implementation of openlist and closelist.

- **Openlist** by definition is inner area which at the end considered as location where wall should be build. Each open node is represented by wall segment.
- **Closelist** is the list of nodes that represents inner area which will be walled off. It is called closelist since all the nodes are closed from any conversion and they will remain as interior nodes.

At the start there will be a single node in closed list and number of openlist nodes required to surround the staring area. Algorithm begins with removing a single node from openlist and adding it to the closelist. Then appropriate nodes from the map are added to openlist such that those nodes surround the node added to inner area. Selection of openlist node that is to be added to inner area depends on heuristic function. Before selecting any node from the openlist each openlist node gets a heuristic value, depending on the heuristic value nodes are sorted and smallest heuristic valued node is added to inner area.

There are three basic functions used in this algorithm they are as follows:

1. Traversal function: Traversal function is the function which traverses through the openlist and finds which one it to be converted from openlist node to closelist.

2. Heuristic Function: Heuristic function ranks the nodes present in openlist. Heuristic function is heart of the algorithm since it decides which nodes to rank and using what properties. Since ranking decides which node will be used for interior area heuristic function actually controls the shape of wall.

This function has to maximize the wall-off area while minimizing the cost of walling off. So this function starts with cost of walling off for each node. Now this condition can create wall stretched in one direction and then it can form abnormal wall. Generating wall which grows in proper way, one more variable is added; distance from the starting location. This variable gives preference to node that is nearest to the starting point. This addition is useful to create circular shaped walls which maximize the area with given perimeter.

To make the distance variable less significant we need to multiply cost of walling off with a large constant which is equal to maximum distance from center of town. Maximum distance is taken so that cost of walling off will be more significant even at maximum distance.

f (n) = c x u (n) + d (n)

- ***c*** is the large constant (Mostly equal to max distance)
- ***u (n)*** is the function for calculating cost of walling off.
- ***d (n)*** is the function for calculating distance from the starting location.
- ***n*** is the cell or node on which we are working.

3. Acceptance function: This function decides when to stop processing. Acceptance function can change from game to game depending on the goal of game. Usual acceptance function will be at certain area algorithm should stop.

Pseudo code for algorithm is as follows:

1. Start with adding starting area node to close list.
2. Add surrounding area nodes to the openlist.
3. While Acceptance function is not satisfied
    a. Using Traversal function find the node to be added.
    b. Add node to closelist and remove from openlist.
    c. Add nodes need to wall of current node
    d. Recalculate heuristic values and sort.
4. Create basic wall structure from nodes present in openlist.

Once acceptance function is satisfied all the openlist nodes will be taken as basic wall structure. Growing nature of this algorithm is shown in figure 19 with help of steps 1-6.

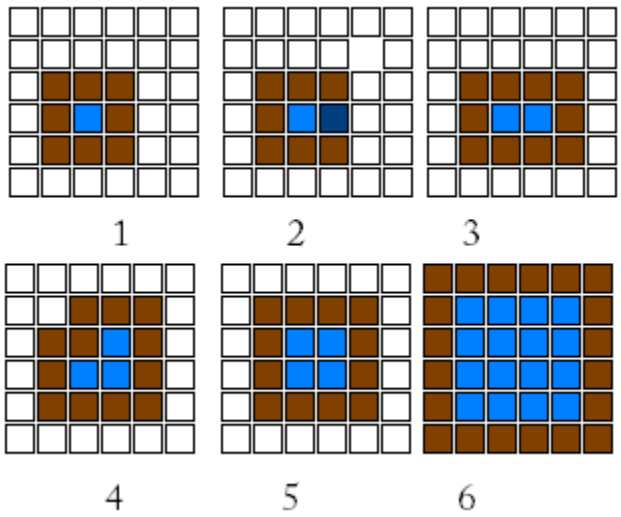


Figure 19: Growing approach

This algorithm is simple to use and gives better result. This algorithm also handles advance issues like map edges, maximum distance, minimum distance. All walls need a gate for moving units across it and this can be solved by brute force. Brute force approach is not good when it creates gates facing natural barrier or opposite side to the enemy. So we can use A* path finding algorithm as explained in previous chapter to create gates.

Chapter V: Improved Wall-Building Algorithm

In this chapter we will analyze the existing wall building method that uses greedy algorithm. After discussing its pros and cons we will suggest improvements that address some of deficiencies of the original method.

Analysis of Exiting Method:

Existing wall building algorithm is very fast and produces a good wall structure, the main characteristic is that the wall looks aesthetically appealing to a human player. Algorithms also handle all the advance issues like map edges, taking advantage of natural barriers and generating fixed radius wall by specifying max or minimum distance from the staring location.

But it does not give us the wall which is best suited for certain games. As we defined in chapter III, wall has to accomplish certain goal to be called as good wall. After careful analysis we found that existing algorithm may be improved in certain areas to make a better use of natural resources.

Utilization of natural barriers: Existing algorithm utilizes natural barriers whenever they come across; this is undoubtedly a very good practice, but it has some flaws. Algorithm can use barriers that are close to starting point (town center), but it does not consider the help which it can receive from natural barriers that are a little distance apart. This is a feature common to all greedy algorithms – usually referred to as "short-sightedness".

Another problem with the original algorithm is that it just incorporates natural barriers into the wall structure without making sure that which part of the barrier should be used. In some scenarios doing that produces very inefficient solution, by starting the wall from outermost edge of barrier. Due to this major part of barrier remains inside the walled off area and keeping barriers inside walled off area will reduces useful part of interior area. Use of natural barrier makes wall building better since most of them are unbreakable and cost nothing. So algorithm should consider all the natural barriers including those which are at some distance from center of location. Algorithm should try to shift towards the area where it can receive help from natural barrier.

Wall location: Current algorithm uses a fixed staring point (town center) which is used as a reference point for calculating heuristic functions. Due to this implementation wall cannot deviate from staring position to a better position. When it comes to random maps static starting point might not be the best solution since it can produce wall which is not optimize for that area. To overcome this we need to consider a dynamic reference point which could allow us to shift the wall in good region of map.

Making town self sufficient: Algorithm completely ignores the fact that player needs to make use of resources for building the town. As it was defined in the previous chapter the wall should be build in such a way that it encloses all/most needed resources which are in near vicinity. This makes town self sufficient since all the requirements can be field inside the walled off area. Notice that resources that are not inside the wall are still reachable, but

- to collect those resources workers have to go through the gate which connects interior with the rest of the world and gate may not be located on the straight-line line path from town center to the resource location, thus the time to collect resource may be significantly higher compared to the straight-line path.
- also when it comes to RTS game all worker units including resource gathering workers are defenseless and they should remain inside the wall to keep them safe from enemy.

Since current algorithm ignores all these facts it needs an improvement in procedure such that whenever we build a wall it should take all the resources inside the wall.

Implementation Improvement:

As we found out, there are some flaws or places to improve in the current (greedy) wall building algorithm. So we decided to implement some improvements which will overcome the flaws present in the current algorithm. We will keep the graph base approach as it is since it gives freedom over grid based solution and makes easy to define valid wall. All the definitions for the wall will also be same.

Implementing Shrinking algorithm:

As we know greedy algorithms just sees the local maxima and finds the solution based on it. This approach stops the process of finding good wall structure as soon as current solution meets the acceptance criteria. Algorithm fails to find a better solution

which may be available just few iteration from current position. We will keep the main algorithm same but shift its approach from growing the wall structure to shrinking. Shrinking algorithm preserves the main benefits of the original method like aesthetically appealing structure, but also overcomes some of the short-coming of the original method. The most important is that shrinking algorithm is able to utilize natural barriers more efficiently.

New Shrinking Approach: In this approach rather than giving a starting location and starting with a minimal wall, we start with a fixed size wall that encompasses area near the reference point. We make sure that initial wall size will be more than enough to cover basic town facilities as well as keep some free space for town growth. This size will change from game to game according to requirement and the nature of map.

We start with openlist and closelist as original algorithm. The definition for openlist is same but closelist is slightly changed from original algorithm.

- **Openlist**: Collection all the wall nodes in the map. (Dark color)
- **Closelist**: Collection of all the nodes outside walled off area. (Light Color)

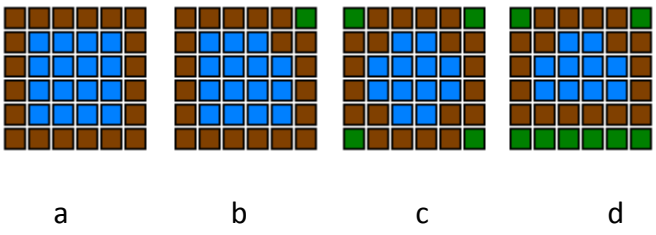


Figure 20: Shrinking approach

We start with openlist which will be all the nodes from initial wall. This wall will be used for shrinking. We change all walled off node's type to interior area. Now our aim is to remove a wall node from openlist and put it to closelist.

Move in context of this approach will be to remove only one wall node at a time and find the nodes from interior area which can be used as wall to make sure that wall satisfy all the conditions. We can see in figure 20 the progress of wall shrinking which moves towards center. Move requires ranking of openlist nodes to decide which node to be removed. This ranking is based on heuristic function which ranks the nodes in ascending order of heuristic value. Then node having maximum heuristic value will be removed. Heuristic function for shrinking is combination of cost of walling off and distance from reference point. **Reference point** is center of initial wall structure; currently it is fixed. (we make it dynamic in next improvement)

Implementation of new heuristic function for shrinking:

f (n) = c x -u (n) + d(n)

- **c** is the large constant (Used for minimizing the significance of distance variable)
- **u(*n*)** is the function for calculating cost of walling off.
- **d(*n*)** is distance from fix reference in interior area
- ***n*** is the node on which we are working.

Our aim for calculating the heuristic value is such that we should consider the node which has lowest walled off cost and maximum distance. So to achieve proper heuristic we multiply cost of walling with large constant to minimize impact of distance variable. Also we multiply the cost of walling value with negative one so as to achieve max heuristic value for lowest walling cost.

In this approach traversal function and acceptance function remains the same as in the original algorithm. Acceptance function can be set to satisfy maximum wall segments used or max interior area. Once we get the acceptance function value as true we can take all the nodes from openlist and generate basic wall structure.

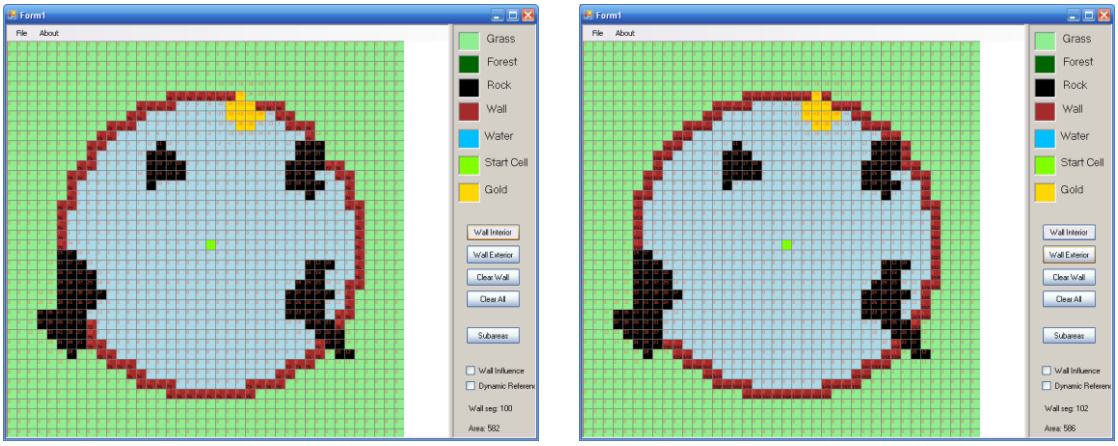


Figure 21: Left Growing and Right Shrinking algorithm with same acceptance function.

Implementation of influence map:

Use of influence map is a major part of improvement. Since original algorithm do not take into consideration that the resources should be walled off to make town self sufficient. Also it does not try to relocate wall to area where more natural barrier assistance is present. To overcome these flaws we suggest an implementation of basic influence map which will help us in better calculation of heuristic function. This new heuristic function will direct the wall such a way that wall will try to make sure that it keeps the resources inside the wall and utilize more number of natural barriers. Also make sure that major part of natural barrier will be outside walled off area.

Implementation of influence map is very simple; we are attaching an influence value to each cell depending on its neighbor. This influence value will be aggregate

influence calculated from surrounding cells up to second layer. Influence value also varies with distance from the reference point inside the walled off area.

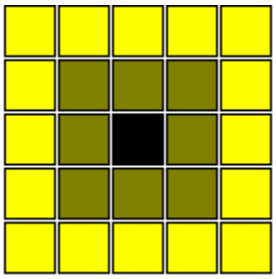


Figure 21

In figure above black cell will calculate influence value from the outer two layers. This influence value will be different for different type of cell and also change for each layer. Rock or forest will have positive influence where as resource like water gold will have negative influence. Since our aim is to keep all the resources inside we give more negative influence value to resources. This way node with is nearer to any resource will get large negative influence.

Also cell will multiply all the influence values with distance between cell and reference point in the map. This make sure that influence will be assigned in such a way that farthest node from reference point will get maximum value it may be positive or negative depending on type of influence. This makes sure that nodes surrounding resources are eliminated last and nodes surrounding natural barrier gets eliminated first. This way we keep the resources inside walled off area and try to use natural barriers such that maximum part of them stays outside while we use them.

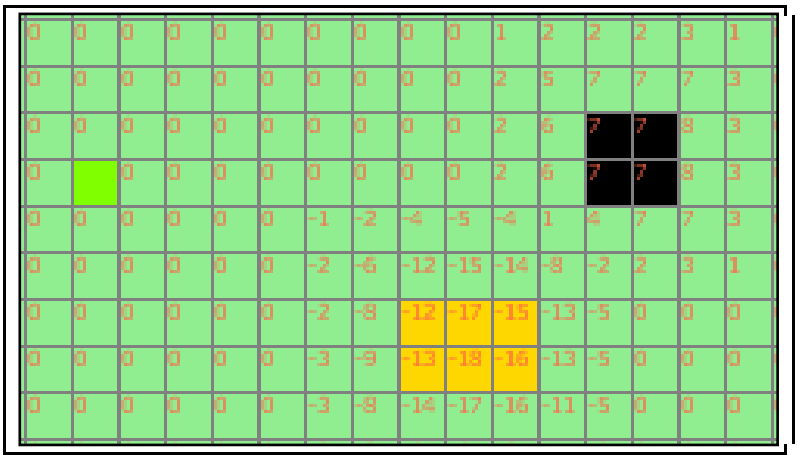


Figure 22: Influence value with respect to reference point

Code for calculating influence value for each cell in the map is as given below

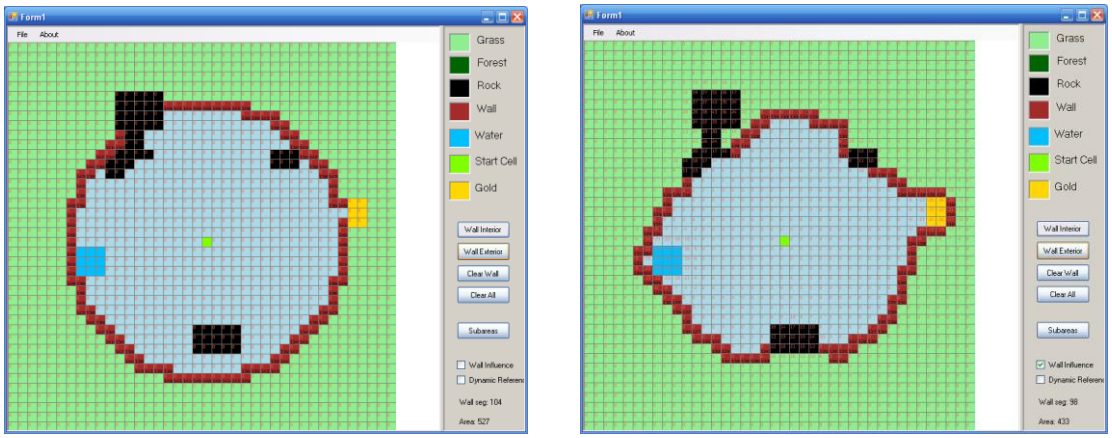
```
for (int row = 0; row < m_Size; row++)
for (int col = 0; col < m_Size; col++)
   {
     int Influence = 0;
     for (int i = -2; i < 3; i++)
     for (int j = -2; j < 3; j++)
      {
        int tRow = row + i;
        int tCol = col + j;

        if (i < -1 || j < -1 || i > 1 || j > 1)
        {
         if (tCol >= 0 && tCol < m_Size && tRow >= 0 && tRow < m_Size)
         if (m_cells[tRow, tCol].m_Type == BARRIER)
             {
               influence ++;
             }
         else if (m_cells[tRow, tCol].m_Type == GOLD)
             {
               influence -=2;
             }
        }
        else
        {

         if (tCol >= 0 && tCol < m_Size && tRow >= 0 && tRow < m_Size)
         if (m_cells[tRow, tCol].m_Type == BRARRIER)
            {
              influence +=2;
            }
         else if (m_cells[tRow, tCol].m_Type == GOLD)
            {
              influence -= 4;
            }

        }

        double dist = Math.Sqrt((col - m_StartRow) *(col - m_StartRow)
                     + (row - m_StartCol) * (row - m_StartCol));

        //50 and 20 are constants used for normalization
        m_cells[row, col].m_RockInfluence = influence / 50 * dist/20;

       }

    }
```

Implementation of the new heuristic function for shrinking:

f (n) = c x{ -u (n) + i(n) } + d(n)

- **c** is the large constant (Mostly equal to max distance)
- **u(*n*)** is the function for calculating cost of walling off.
- **d(*n*)** is distance from fix reference in interior area
- ***i(n)*** influence value of node.
- ***n*** is the node on which we are working.

We can see that this heuristic added *i(n)* "influence value" to it. Depending on influence value heuristic value for node gets affected. Now influence value also plays a role in ranking of nodes and eventually shaping process of wall.



A **B**

Figure 23: A: shrinking approach without influence map.

B: Shrinking approach with influence map.

As you can see the output of original algorithm gives solution which incorporates most of the natural barriers inside the wall where as new algorithm utilize the natural barrier more efficiently. Also both the resources are kept inside the walled off area by new algorithm while making sure that it also uses natural barrier.

Implementation of dynamic reference position:

As we saw original algorithm needs a starting position and it has some disadvantages. Starting position not necessarily gives the best location for wall to be built in that particular map. Since we depend heavily on other AI module to give staring point we lack the advantage of deciding where to build the wall. So we decided to improve this situation by adding dynamic reference point. According to first improvement we are considering shrinking approach in which we start with a large initial wall and shrink it to get final wall structure. In this shrinking approach we are considering the center point of the wall as our reference point which is used in heuristic functions. Currently this reference point is fixed so wall shrinks in circular manner towards the initial center.

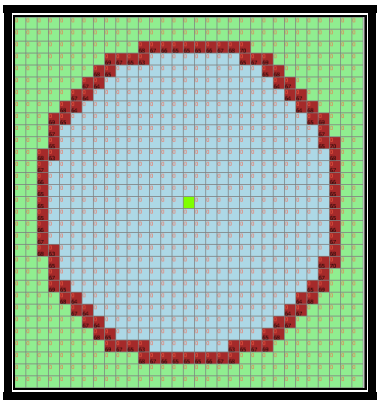


Figure 24: Wall shrinking towards center.

Now if we make this reference point dynamic it will add flexibility to the wall shrinking according to the map. We will calculate minimum and maximum position of wall segments available in both co-ordinate axis. Then we find the center in terms of x axis and y axis. This will be our new reference point and all the functions will use this dynamic reference point. This will be re-calculated each time we shrink the wall.

Since heuristic heavily uses reference point we can see a large change in the ranking of wall segment for each iteration. This change of ranking will eventually shifts the wall towards more influential area. Due to dynamic reference point shrinking takes place towards influential area rather than center of initial wall.

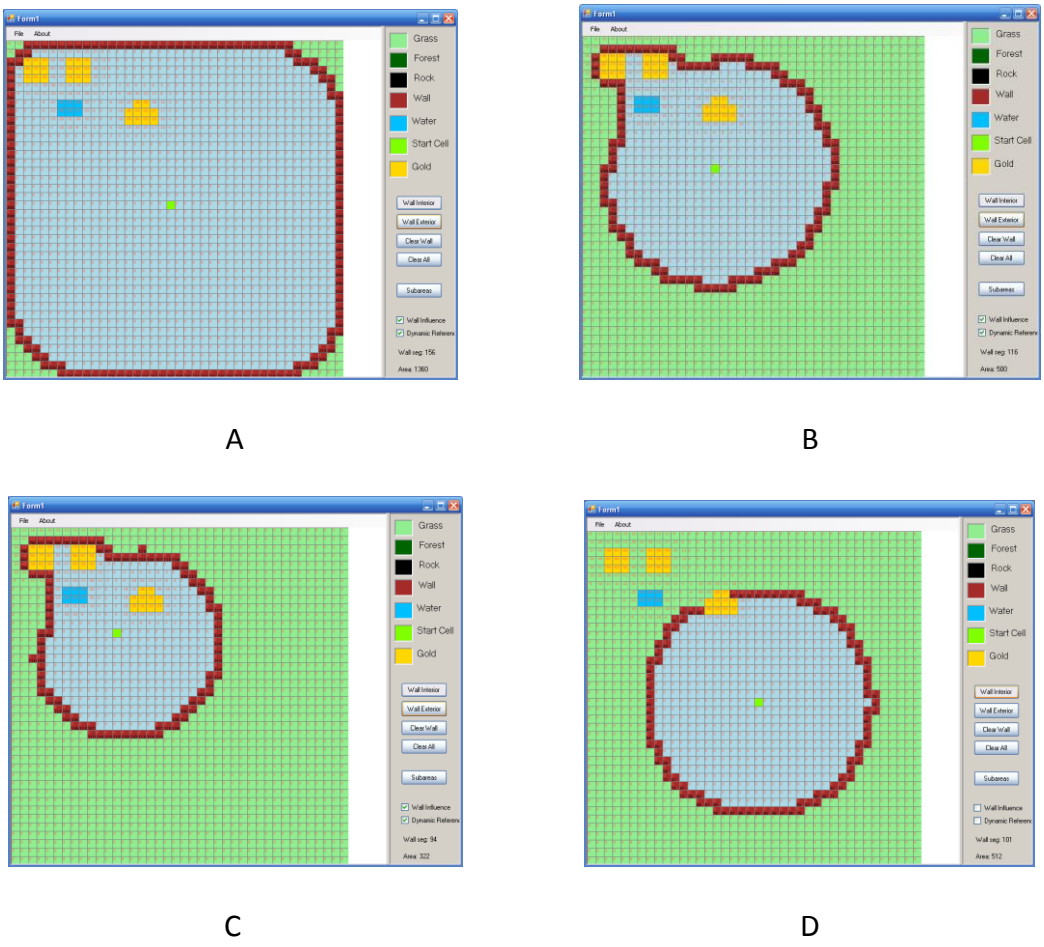


Figure 25: A,B and C shows the process of wall shifting due to dynamic reference point with influence. D shows the output with original greedy algorithm.

Examples of implementation

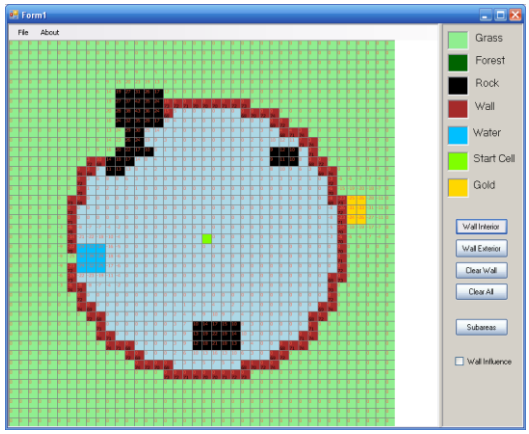


Figure 26 a: Original Algorithm
Wall segments: 100 Inner Area: 532

As we see original algorithm produces good wall but it ignores the gold which it should take inside the wall.

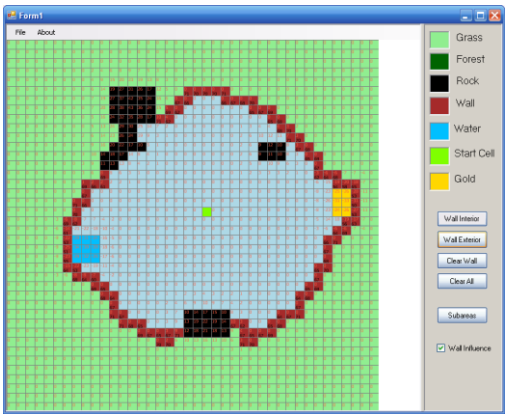


Figure 26 b: New Algorithm
Wall segments: 103 Inner Area: 478

Our Algorithm with improvement gives better result than original algorithm since it makes sure that it will take all resources inside the wall as well as utilize the natural barriers more efficiently. Even though inner area is comparatively less that original algorithm area we can see our wall will be strategically better than original one. This result is combine affect on shrinking and influence map.

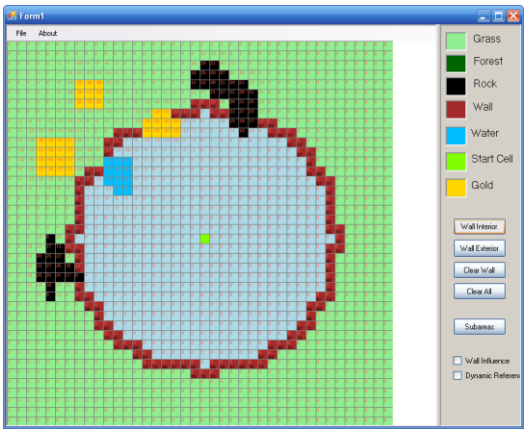


Figure 27 a: Original Algorithm
Wall segment use: 100 Inner Area: 507

In this example we can see resources are completely miss manage by the original algorithm. Wall completely ignores the fact that resources are present in near vicinity.

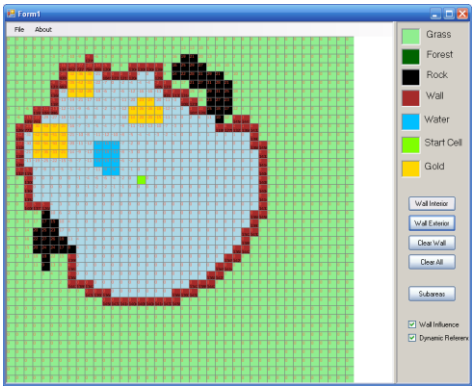


Figure 27 b: New Algorithm
Wall segment use: 104 Inner Area: 456

As we can see new improved algorithm gives better output. Here we can see the affect of reference point shifting. Shifting of reference point makes sure that all the resources are walled off and we still take help from natural barriers as far as possible. As you can see upper mountain is not fully utilize this happens due to influence value distribution and distance from reference point. In future we could implement better influence distribution technique to overcome this flaw.

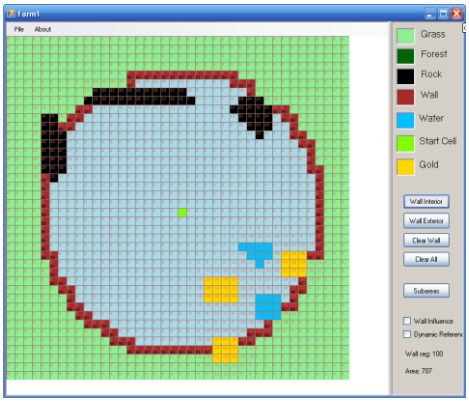


Figure 28 a:Original algorithm
Wall segments: 100 Inner Area: 707

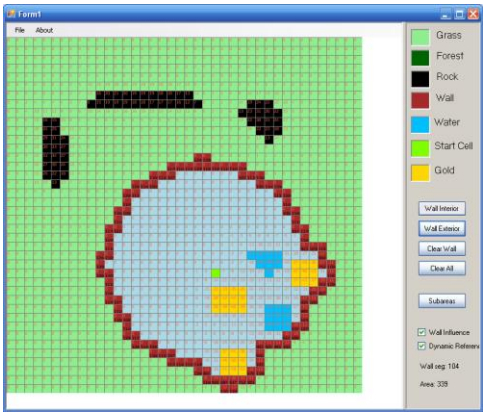


Figure 29 b: New Algorithm
Wall Segments: 104 Inner Area: 339

In this example we can see that the output of original algorithm is far better than our new algorithm. This is due to polarization of resources; since we give preference to resources more than natural barriers our algorithm shifts towards area which has maximum resources present and completely ignores the fact that there are good natural barriers present which can give better wall output.

Conclusion and Future Work

As results shows our improvements to the greedy algorithms gives better result than original greedy algorithm. Main aim of improvement was to make wall building more strategic and useful for player rather than creating just a defensive wall around the town. Original algorithms major focus was on minimizing cost of wall building and maximizes the area gain. New algorithm considers this but also try to make wall building more strategic. Even though cost area ratio don't always compliment strategic implementation of wall but strategic wall building gives a edge when it comes to building better and self-sufficient town. Game

New algorithm does not give best solution every time as we saw in third example but most of the time wall build by new algorithm will give better stability to economic as well as strategic in longer run.

Regarding future scope one can implement better influence map which will give correct solution even though map is polarized with resources and natural barrier. Currently normal greedy algorithm has advantage when we compare cost of building to interior area ratio with new algorithm. We need to address this issue so that we can keep the same ratio or give better ratio than original algorithm has. In future we can use Mip-Map method which can significantly reduce the number of calculation. Also currently resources are used as impassable structures which might not be the case every time so in future algorithm should handle this case.

References

[1] Wall building for RTS Games Mario Grimani Sony online AI Game Programming Wisdom 2 (ed. S. Rabin), Charles River Media

[2] Terrain Analysis in Realtime Strategy Games Dave C. Pottinger

[3] AI Wall Building in *Empire Earth® II* Tara Teich, Dr. Ian Lane Davis

[4] Ramsey, M (2004), Designing a Multi-Tiered AI Framework, AI Game Programming Wisdom 2 (ed. S. Rabin), Charles River Media, pp. 457-466

[5] Graham's Scan: http://en.wikipedia.org/wiki/Graham_scan

[6] AI for real-time strategy games, Master thesis By Anders Walther 2006

[7] A* algorithm implementation: http://en.wikipedia.org/wiki/A%2A.