Automated Terrain Analysis in Real-Time Strategy Games

Chen Si
Games Studio
University of Technology, Sydney
chen.si@student.uts.edu.au

Yusuf Pisan
Games Studio
University of Technology, Sydney
yusuf.pisan@gamesstudio.org

Chek Tien Tan
Games Studio
University of Technology, Sydney
chek@gamesstudio.org

ABSTRACT

Real-time strategy (RTS) games represent a mainstream genre of video games. They are also practical test-beds for intelligent agents, which have received considerable interest from Artificial Intelligence (AI) researchers, in particular game AI researchers. Terrain knowledge understanding is a fundamental issue for RTS agents and map decomposition methods can help AI agents in representing terrain knowledge. These contributions support AI agents' path finding and combat strategy. In some RTS games, such as StarCraft, all terrain information is provided to AI agents at the beginning of the game. This presents an unfair advantage, as human players do not have access to this information. We propose a terrain analysis framework, in which AI agents gather terrain knowledge by managing scouts to explore game maps. This framework is part of my Ph.D. study that is investigating scouting strategies for RTS games. We developed an extension to the StarCraft system, called terrain engine that releases terrain information in small chunks rather than providing the full map, to investigate human-like techniques for scouting. Within the terrain analysis framework, we present a reconnaissance (recon) algorithm to guide individual scout units in recon tasks. Then, we identify the factors for terrain exploration planning model, which will be implemented as part of our future work.

1. INTRODUCTION

Real time strategy (RTS) games pose interesting challenges for both human players and artificial intelligence (AI) bots. In RTS games, a player need to consider high-level game issues such as building order, unit composition and troop formations, and at the same time micro-manage individual units to attack efficiently in packs and to minimize damage. Unlike traditional turn-based board games such as Chess and Go, in which players are able to get access to full game-state, RTS games require players to make decisions with partial game-state information. Actively gathering information about the opponent is a crucial strategic component for RTS games.

Terrain information collection and analysis is an important foundational part of RTS games, due to the vital information it contains. The analysis results [12] are used in strategy making for AI bots (e.g. ambushing in narrow paths) [3] and in advanced path-finding algorithms (e.g. navigating unit groups) [13].

A lot of research has been performed in terrain information analysis. AI agents can recognize general terrain features such as regions, chokepoints and base-locations by applying computer vision techniques to game maps [3][11]. These techniques rely on having access to the full game-map at the beginning of the game, and would be considered cheating for games where human players do not have access to this data. Although strong players know the popular maps, most novice players are unfamiliar with the game maps. A cheatings AI [8], which unfairly gets access to global game information, destroys the game experience for players.

As part of my Ph.D. study, we present a dynamic terrain analysis framework that collects map information and recognizes terrain features by managing scout units tactically and strategically. StarCraft Brood War is a suitable platform to test our idea about scouting in unknown territory. We, thus, describe a terrain engine that has been implemented as an extension of StarCraft Brood War. The terrain engine releases terrain information based on what AI agents scouting units would be allowed to detect in their immediate vicinity. We present a novel tactical navigation strategy for scouting. As a part of our future work, we are also exploring a planning module to determine when to scout, where to scout and which units are chosen to scout, by considering a variety of factors such as size, movement speed and price of units. Our novel terrain analysis method reasonably avoids cheatings, and contributes to build up human-like AI robot in RTS games. Scouting in unknown territory presents a lot of challenges, such as balancing resource-consuming between exploring terrain and attacking enemy. In this paper, we are investigating how to explore terrain features. Overall scouting strategies for exploration are left for our future work.

2. RELATED WORK

Terrain understanding is a fundamental task for RTS AI agents to extract essential information for other decision-making sub-systems. Efficient analysis methods have been developed to represent game maps in different ways. A title-based terrain representation system has been used in BANG – an RTS game [12]. Game maps are divided into tiles and classified into two types – convex areas and non-convex areas, which are used in tracking routes. According to the requirements of combat scenarios, game space [3][11] was divided into regions (free movement areas), chokepoints (narrow areas) and obstacles, by using image processing techniques and the Voronoi diagram. The Hale *et al.* [5] also presented an automatic growth mesh technique to perform region partition – DEACCON, in which small squares are seeded and grown to detect edges. This automated navigation-mesh-generation method has also been extended into 3D scenarios [6].

The works mentioned above focused on the analysis of pre-loaded game maps. In this paper, we take a different approach: building map information incrementally. Navigation of scout units is the primary challenge for our terrain exploration agent. Potential field technique [4] has been used to deal with fog of war in Wargus (a clone of WarCraft 2), evaluating unexplored terrain tiles to navigate scouts to explore terrain. Park *et al.* [10] presented a heuristic navigation tactic for scout units in collecting opponents' information. They devised a navigation method where a scout walks around the enemy base.

In terms of planning in uncertain game states, several tactical planning algorithms provide references for us to develop a strategic terrain-exploration planner. For example, Chung *et al.* [2] employed the Monte Carlo method to create a random planning, evaluation and evolving planner for combat scenarios. Another promising algorithm [1] – UCT (a kind of Monte Carlo planning) was used in making tactical assault plans, concentrating on group-based plan making. On the resource collection aspect, Naves and Lepes [9] presented a stochastic search and planning method to solve the resource production-planning problem. Many strategies for RTS games play an important role. All of these strategies now still rely on full map information being available. Being forced to do scouting as well as choosing and modifying a strategy makes it more challenging.

3. METHOD

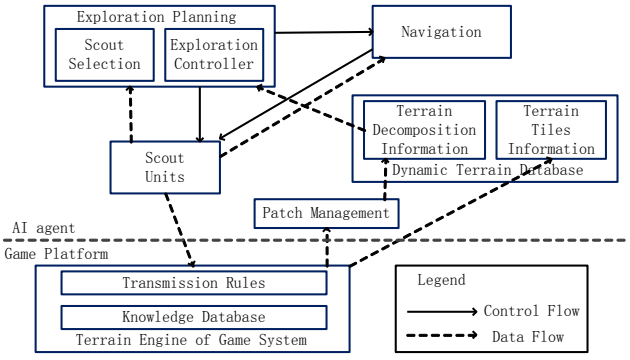

Figure 1. An overview of terrain analysis.

Our proposed AI agent framework consists of four main parts as shown in Figure 1. In order to plan the recon task, the *Exploration Planning* module first performs *Scout Selection,* using known terrain data and other relevant game information. Then it assigns the scouts to appropriate destinations and informs the *Navigation* component to conduct low-level guidance. Next, the *Dynamic Terrain Database* module is an extended version of the *Knowledge Database*, which adds various other types of exploration-guidance data. The *Patch Management* component is utilized to manage explored patches (small map chunks), and to recognize decomposed terrain information from these patches. The *Navigation* component then leads the way of a scout unit to explore a certain area efficiently.

In many RTS games such as StarCraft: Brood War, the game system does not provide the full maps. Therefore, analyzing the full map, which is not normally available to human players, provides an unfair advantage to AI agents. We have hence implemented a *Terrain Engine* that releases limited information to AI agents cumulatively, rather than releasing the full map at once.

The *Terrain Engine* provides a platform for our framework to explore scouting strategies for AI agents in a fair setting, similar to how human players employ scouting. The *Terrain Engine* will only release information about parts of terrain that are in the visible area of a scout. There are two kinds of information in the *Knowledge Database* of the *Terrain Engine*. One is a *regional* database, which maintains the results of terrain decomposition, including regions, chokepoints, and locations of resources. The other is a *tile-based* database, in which a game map is represented by square tiles. Each tile is labeled passable or impassable, depending on whether a ground unit can pass or not.

The game system releases terrain information to AI agents based on the four transmission rules below:

(1) The engine will release information, including passability of tiles, points in edges of regions (edge-points), end-points of chokepoints and center-points of resources, when they are in the visible area of a scout.

(2) If all the edge-points of a region have been detected by scouts, information of the entire region will be released.

(3) If two end-points of a chokepoint have been explored, information of the chokepoint will be released.

(4) If center-points of resources are "seen" by scouts, we assume that they know the entire information about the resources (e.g. locations and reserves).

3.1 Exploration Planning Module

The purpose of the *Exploration Planning* module is to collect the most valuable terrain data for the current game state as soon as possible with minimal resources. Therefore, deciding which locations to explore and choosing appropriate scouting units in corresponding game states are significant challenges for RTS agents.

Some areas are more valuable than others during certain game periods. For example, areas around the straight-line path from a player's start location to the enemy base are frequently visited (hence valuable) by units during beginning 10-15 minutes. Therefore, scouts should preferably consider exploring these areas first.

The role of the *Exploration Controller* is to generate the exploration plans. In an exploration plan, the general procedure of exploration starts by assigning a scout to explore the possible start-locations of the enemy base, right when the game starts. After that, the scout walks along its movement direction, until one edge of the region is "seen". Then, the *Navigation* component (Section 3.5) takes over control of the scout in order to explore the region. Next, the following steps are repeated: (1) Search a valuable area for current game state, based on obtained terrain information. (2) Manage a scout to move to one edge of the area, and active the domination of the *Navigation* component.

The role of the *Scout Selection* component is to consider which units should be assigned for recon tasks, in which several properties of the scout units need to be considered.

(1) Visibility: Different units have different sight ranges. Units with larger sight ranges are able to "see" further.

(2) Speed: Units with higher movement speeds can perform scouting faster.

(3) Armor: Stronger armor can help scout units live longer in dangerous scouting areas.

(4) Cost: The cost for producing scout units varies. The cost should be balanced between producing more un-expensive scouts to do multiple units scouting and using less advanced scouts to complete the tasks.

(5) Air or ground: Air units, when available, can be used for scouting when the situation requires.

3.2 Data Structure

We combine two kinds of data structures, region-based structure and tile-based structure, to represent RTS game terrain for exploration planning and navigation purposes separately. Region-based structure classifies areas into three different types: regions, chokepoints and obstacles. In this representation method, we employ *region*, *chokepoint*, *baselocation* and *startlocation* to record essential elements of a game map, like BWTA [11] did. A *region* structure contains the data of its border polygon and indexes of connected *chokepoints.* In a *chokepoint* structure, two endpoints of the chokepoint and the indexes of the two connected *regions* are presented. A *baselocation* is a kind of *region* with additional information about resources contained in that region. A *startlocation* is a *baselocation* with a player's information.

This structure is an efficient way for the *Exploration Planning* module to infer the next valuable area to be explored. For example, when the *startlocation* of the opponent is found, areas, which can be entered via *chokepoints* from the *startlocation*, are more likely to be explored, if resources exist in these areas. Furthermore, data contained in this structure also plays an important role in making strategies. For example, a player would tend to choose a rush strategy if the distance between *startlocations* are short, and *chokepoints* around them are relatively wide.

The tile-based structure divides a game map into tiles with the size of the smallest unit. Based on obstacle information, a tile is also flagged with passable or impassable. In this paper, our navigation algorithm employs the tile-based information to guide a scout in exploring an area. Data in tiles also contributes to more efficient path finding.

3.3 Navigation Component

The purpose of the *Navigation* component is to guide a scout unit to collect data in an area. Generally, the navigator takes control of a scout unit, and begins to navigate it when the unit arrives at the designated spot. The navigator follows two principles. (1) Scout units need to avoid damage from enemy units and keep alive. The resources for scouting are extremely precious, since AI bots may weaken their economy, if they assign too many units to scout. Furthermore, scout can potentially collect more data if they can stay alive longer in unexplored areas. (2) Units should explore designated areas as soon as possible.

To adhere to these two principles, a scout unit explores an area by mimicking human to scout for practical purpose in our system. It will walk around a region whilst keeping an optimal distance from the edge. We employ the tile-based structure to design the navigation algorithm. A tile can be one of three types – *passable*, *impassable* and *unknown*. Initially, all tiles are flagged *unknown*. When they are gradually uncovered, their types change to *passable* or *impassable*, and this change only happens once.

We use a potential field value recorded in each tile to help units find an optimal path in exploring. The method is based on [4][7]. The value is calculated when passability of tiles has been explored in the visibility area of a scout for each frame. The algorithm for calculating this value is shown below:

Algorithm 1 Calculate Tile Potential Value

```
Require : t ∈ gridmap and sr ∈ S
square ← t
pv ← 0
if passable(square) then
  while passable (square) do
    pv ← pv + 1
    square ← expand (square)
    if pv > (sr - 2) then
      pv ← 0
      break
else
     pv ←-1
return pv
```

t refers to a tile in the map. *pv* represents the potential value of tile *t*. *sr* is the sight range of current scout unit. *passable()* is used to check whether a square is passable or not. *expand()* is to expand a square into its adjacent tiles in eight directions. The higher the potential value of a tile is, the more possible that the scout will move to the location of the tile in next step.

To prevent scout units visiting tiles repeatedly, we include a *visited* flag to record tiles that have been reached in the last several steps. Visited tiles are excluded when choosing the tile with the highest potential value for the next destination of a scout (among the eight adjacent tiles). Sometimes, a scout may be confused when there are two or more tiles with the same highest potential value. In this situation, the previous directions of movement will be used as tiebreakers. Vectors from the current tile to these candidate tiles are first generated. The eventual selected destination tile is then the candidate tile with the smallest angular difference between its vector and the previous direction vector.

3.4 Patch Management Component

The terrain data that is captured by a single unit in each frame will be known as a patch in our system, since the visible area of a single unit can only cover parts of a region in most cases. As data is often incomplete, per-frame queries for recognizing terrain features might be a waste of time. Moreover, terrain data in sequential frames almost always overlap with each other. These two vital problems, (1) how to organize patches properly without overlaps and (2) how to query terrain features efficiently, are handled in our *Patch Management* component.

In our system, we develop a hierarchical *Patch Management* component. In each frame, points of terrain data that a unit has collected in its visible area are organized into patches, according to their spatial connectivity between each other. The patches are stored in a local patch buffer. Patches are merged, if they are spatially connected, when they are pushed into the local patch buffer. The patches in the temporal patch buffer are transmitted and merged into a global patch buffer, in which global patches are stored, for a shot period. After merging, data in the local patch buffer is cleared. This method hence helps to reduce redundant

data, since overlapped patch data is merged and cleared when necessary, and to improve efficiency, because the system do not need to traverse the global patch buffer to merge the new patch in each frame. For each long period, terrain features are recognized by comparing patch data in the dynamic patch buffer and region data in the *Terrain Region Database*. Recognized terrain features are finally stored in the *Terrain Region Database*, and corresponding patches are deleted in the dynamic patch buffer.

4. IMPLEMENTATION

Our framework is implemented in StarCraft: Brood War, one of the most popular RTS game genres in the last decade. It is coded in C++, based on the Brood War API (BWAPI)[1] – a library to provide interfaces for game AI development. The terrain game engine uses the Brood War Terrain Analyser (BWTA) [11] to analyse game maps, and thus obtain all of its processed terrain information. This system can be easily embedded into other AI agents, which are able to play full games. We have finished designing and implementing the *Terrain Engine* and *Navigation* component. The implementation of *Patch Management* component and *Dynamic Terrain Database* is expected to be completed soon. When these low-level and foundational components are completed, we will design the exploration planning algorithm and embed it into the *Exploration Planning* module.

5. EXPERIMENT

Our framework will be evaluated by comparing its performance with the BWTA. We will first choose three AI bots from previous AIIDE StarCraft competitions[2], which rely on BWTA to obtain terrain data. These bots will then be modified into three different versions: (1) the original version, (2) a version that knows nothing about the game terrain, by cutting its link with BWTA, and (3) the version that contains our scouting framework. For each bot, its three versions will compete against each other with 15 games in each map. Nine ICCup (International Cyber Cup) maps will be chosen for this experiment. We will then compare the win-rate to identify the practical applicability of our detection model in RTS game agents. Moreover, we will invite 20 novice players, who know nothing about maps that we choose above, but they have basic skills to play StarCraft: BroodWar. These human players will be organized to play against version (1) bots and version (3) bots five times separately in each map. The questionnaires will be made to test whether our method helps to increase the game-playing experience.

6. CONCLUSIONS

Terrain knowledge plays a pivotal role in helping RTS game players win by using it to infer possible enemy locations, determining opponent strategies and devising strategies that take advantage of terrain features. Scouting is a believable way to collect terrain data for AI players. In this paper, we demonstrate a terrain analysis framework to effectively explore RTS game map, and to distribute information interactively. Potential field method and heuristic information are employed for navigation. The patch management technique is used to manage explored terrain data and to identify features from current data.

Our current terrain analysis framework does not take into account hostile units, whether they should be avoided or engaged. Similarly, obstacles placed by the opponent that block exploration have to be taken into account when scouting. The design and implementation of the exploration planning algorithm are the primary future work for this PhD.

7. ACKNOWLEDGMENTS

The authors would like to thank the financial support provided by the China Scholarship Council, the University of Technology, Sydney, and the Centre of Human Centred Technology Design.

8. REFERENCES

[1] Balla, R.-K. & Fern, A. 2009. UCT for Tactical Assault Planning in Real-Time Strategy Games. In *Proceedings of the International Joint Conferences on Artificial Intelligence* (Pasadena, California, USA, July 11-17, 2009). IJCAI '09. AAAI, 40-45.

[2] Chung, M., Buro, M. & Schaefer, J. 2005. Monte Carlo planning in RTS games. CIG '05. IEEE, 117-124.

[3] Forbus, K. D., Mahoney, J. V. & Dill, K. 2002. How qualitative spatial reasoning can improve strategy game AIs. *Intelligent Systems*, 17, 4, 25-30.

[4] Hagelback, J. & Johansson, S. J. 2008. Dealing with fog of war in a real time strategy game environment. CIG '08. IEEE, 55-62.

[5] Hale, D. H., Youngblood, G. M. & Dixit, P. N. 2008. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. AIIDE '08. AAAI, 173-178.

[6] Hale, D. H. & Youngblood, G. M. 2010. Automated navigation mesh generation using advanced growth-based techniques. In *Game programming gems 8*, A. Lake, Ed. Course Technology Press, USA, 244-255.

[7] Harabor, D., & Botea, A. (2008, December). Hierarchical path planning for multi-size agents in heterogeneous environments. *CIG'08.* IEEE, 258-265.

[8] Laird, J. & VanLent M. 2001. Human-level AI's killer application: Interactive computer games. *AI magazine* 22, 2: 15.

[9] Naves, T. F. & Lopes, C. R. 2012. Maximization of the resource production in RTS games through stochastic search and planning. *IEEE International Conference on Systems, Man, and Cybernetics* (Seoul, Korea, October 14-17, 2012). SMC '12. IEEE, 2241-2246.

[10] Park, H., Lee, K., Cho, H.-C. & Kim, K.-J. 2012. Prediction of early stage opponents strategy for StarCraft AI using scouting and machine learning. In *Proceedings of the Workshop at SIGGRAPH Asia* (Singapore, Nov. 28 – Dec. 1, 2012). WASA '12. ACM, 7-12.

[11] Perkins, L. 2010. Terrain analysis in real-time strategy games: an integrated approach to choke point detection and region decomposition. AIIDE '10. AAAI, 168-173.

[12] Pottinger, D. C. 2000. Terrain analysis in realtime strategy games. CGDC '00.

[13] Preuss, M., Beume, N., Danielsiek, H., Hein, T., Naujoks, B., Piatkowski, N., ... & Wessing, S. 2010. Towards intelligent team composition and maneuvering in real-time strategy games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2, 2, 82-98.

[1] http://code.google.com/p/bwapi/

[2] http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp