AI Tech Team Part II
Improvements to Pathfinding Algorithms

Presenting:
Mario Carniero
Franz Kurniawan
Clayton Mallory
We Will Be Covering…

- Improvements to the A* algorithm, specifically in terms of:
  - Reducing the search space
  - Reducing the calculations needed

- Other Game-Oriented Algorithms and Comparisons
Spatial Grid A*

- Devised by Bobby Anguelov of the University of Pretoria
- Goal: develop "a set of techniques for the improvement of the A* search algorithm's performance and memory usage when used to search grid-based navigational graphs."
- Smaller systems with tighter memory constraints can benefit greatly
The Problem

- A* is a widely-used and much-discussed pathfinding algorithm for many reasons
  - Ease of use
  - Speed Tradeoff

- But when it is applied to 2D/3D space, it suffers some performance drops
  - Space Race, for example

- Large area to cover = large search space
Dijkstra's → "Generic" A*

Figure 5.2: A comparison of search space exploration between Dijkstra’s algorithm and A*
Improvements to A* - Discrete

- **Iterative Deepening A**\(^*\) (IDA\(^*\))
  - Recursive DFS with increasing limit
  - Only searches nodes within a threshold value determined by a Heuristic
  - Memory Cost: Linear to number of nodes
    - Cheaper overall than A\(^*\), but performance loss
    - Heuristic needs calculated on a per-node basis
    - Storing the heuristic in each node increases cost
Memory Enhanced IDA* (ME-IDA*)
- IDA algorithm that, when a node with a lower Cost So Far (CSF) value than the current is examined, will ignore it.
- Compared to IDA*, reduces total nodes explored by >50% and computational time by >30%.
- In testing, more than 1900% slower than and explores nodes orders of magnitude larger than A*!
- NOT worth the memory savings.
Improvements to A* - Continuous

- Dynamic A* (D*)
- Focused D*
- Lifelong Planning A* (LPA*)
- Hybrid of D* and LPA* - "D* lite"
- Many more variations
Improvements to A* - Continuous

- **D* lite**
  - **Advantages:**
    - Simpler to use and understand compared to D*
    - Easier to extend, hence the many derivatives
    - Worst case efficiency is that of focused D*
  - **Disadvantages:**
    - Memory cost similar to A*
    - Moderate performance gain
Improving A* - Hierarchical

- Common Algorithms:
  - Hierarchical Pathfinding A* (HPA*)
  - Dynamic HPA* (DHPA*)
  - Partial-Refinement A* (PRA*)
  - Minimal Memory (MM) abstraction
Hierarchical approaches to pathfinding reduce processing and memory costs in exchange for increased algorithm memory cost and the chance of suboptimal paths.

- Subdivide the problem to develop a path
  - Less search space in total
Improving A* - Hierarchical

a) The search space explored to solve the overall pathfinding problem

b) The combined search space required to solve the same problem using subdivision

c) The search space required to solve each of the smaller sub-problems

Figure 7.1: The subdivision of a large pathfinding problem into smaller sub-problems
Improving A* - Hierarchical

- Post-processing smoothing needed
- Overall, memory usage and search efficiency improved greatly, but potentially drastic loss in optimality
Spatial Grid A* Algorithm

- Aims to combine the low cost of hierarchical algorithms with the performance of continuous algorithms (like D* lite)
a) Initial line segment creation stage

a) Connection of line segments using A*

Figure 8.3: SGA* sub-goal creation and refinement
Figure 8.4: Search space comparison between A* and SGA* for a simple search problem.
SGA* Problems

- Situations where dividing up the problem creates impossible paths

- A pre-processing step can label "bad" areas, preventing this issue
SGA* Variations

a) A* search space exploration for an example problem

b) SGA* Variant 1 search space exploration

a) SGA* search space exploration with dynamic sub-goaling and refinement path trimming

b) SGA* Variant 2 search space exploration
SGA* Variations

- Version 3 (the final version), makes a single modification to Version 2, in that unnecessary line segments are removed from the path and only used to select sub-goals

- Greatly improves path optimality while maintaining low cost
A* Applied to Triangulations
Path Planning in Triangulations
Benefits of TA* and TRA*

- Robotics: Non-point object, needs to avoid obstacles by some margin
- Games: Needs to be very fast and use minimal memory
Are Optimal Paths Required?

- Yes! There are many reasons:
  - Complexity of the game
  - Speed of Algorithm
  - Accuracy of Algorithm
Triangulation

- Start with an area and a collection of points
- Add non-crossing edges between the points
- Continue connecting points until all connected
Delaunay Triangulations

Triangulations in which the minimum interior angle of all triangles is maximized

Better triangulation: tends to avoid thin and skinny triangles

Can be done locally by edge flipping diagonals across quadrilaterals
Triangulation-Based Pathfinding: Advantages

- Represent detailed areas better
- Triangulations have much fewer cells and are more accurate than grids
Triangulation Reduction A* (TRA*)
TRA*

- Abstract triangles with 3 constrained edges as degree-0
- Abstract triangles with 2 constrained edges as degree-1
- Put the triangle adjacent to the unconstrained edge on a queue
TRA*

- Go through the queue
- If the triangle is now degree-1, abstract it as one
- And put the un-abstracted face across the unconstrained edge onto the end of the queue
- Otherwise, just remove it
- Sometimes a connected component is “collapsed” into all degree-1 triangles
TRA*

- Go through the other triangles
- Determine which ones have neither constrained edges nor adjacent degree-1 triangles
- Abstract these as degree-3
- There are $2n - 2$ for a component with $n$ obstacles
TRA*

- From degree-3 triangles, move through the corridors of un-abstracted triangles to the next degree-3 triangles
- Abstract these triangles as degree-2
- If there are still any un-abstracted nodes, abstract them into one or more “rings” of degree-2 triangles
TRA* Degree-3 Node Search

- Start on a degree-3 node: search queue initialized with a state using that node
- Goal on a degree-2 corridor: degree-3 nodes on both ends of that corridor are possible goals for the search
**TRA* Degree-3 Node Search**

- Start in degree-1 tree: search queue initialized with states using degree-3 nodes at ends of corridor at the root of the tree
- Goal is one degree-3 node
- Now search moves only between degree-3 nodes
Example - Video

http://www.youtube.com/watch?v=9dCUTbj2rxY&feature=endscreen&NR=1
Experimental Results

- Execution times of standard A* and TRA*

![A* Execution Time Graph](image1)

![TRA* Execution Time Graph](image2)
Triangulation Reduction
Conclusion

- Triangulations can accurately and efficiently represent polygonal environments
- Triangulation-based pathfinding finds paths very quickly and can also find optimal paths
- Our abstraction technique identifies useful structures in the environment: dead-ends, corridors, and decision points
- This abstraction can be used to find paths even more quickly, only depending on the number of obstacles
Tree-search Algorithms and an AI for TBS
Graph Search vs. Tree Search

- Graph searches are commonly used for pathfinding in physical space (i.e. get from point A to point B)
- Tree searches are more commonly used in decision-making for finite-branch games
  - Decision trees are not always trees!
    - Commutative games
    - Multiple ways to win
- Although trees are graphs, algorithms for graph search are much less efficient than tree search algorithms
Minimax

- Minimax theorem:
  - For every two-person, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that
    - Given player 2’s strategy, the best payoff possible for player 1 is V, and
    - Given player 1’s strategy, the best payoff possible for player 2 is −V.

- Minimax Algorithm gives this optimal strategy.
Minimax

- For two-player games with alternating moves and finitely many choices for each move
- Heuristic needed for intermediate stages
- The Algorithm:
  - Evaluate tree to depth $n$
  - Evaluate heuristic at leaf nodes
    - Heuristic returns $\infty$ for a player 1 win and $-\infty$ for a player 2 win
    - Intermediate values measure which player is favored
Minimax

- The Algorithm
  - Label each node on even layers with the maximum of its children and on odd layers with the minimum of its children
  - Chosen path is the one whose number was propagated to the root
Minimax

- The algorithm is optimal if the tree can be explored completely (to wins and losses), but otherwise depends on the strength of the heuristic.
Alpha-beta

- An optimization to the minimax algorithm
- Since only one node is propagated to the root, most of the tree is not used.
- Maintain two values (conventionally called alpha and beta) for internal nodes
  - Alpha is a bound on the best possible score player 1 (the maximizer) can get from this position
  - Beta is a bound on player 2's best score
- Initially alpha is $-\infty$ and beta is $\infty$, and the interval gets smaller as the tree is searched
  - The subtree need not be explored if beta < alpha
Minimax with alpha-beta pruning on a two-person game tree of 4 plies

The current state of the game gets a pair of alpha and beta values, which are initially empty:
- the alpha value will be used as the initial record when the player maximizes on the beta values from the next potential game states (PGSs)
- the beta value will be used to check against the beta values from the next PGSs (which are alpha candidates) for potential pruning
Applications & Drawbacks

- Designed for two-player games with alternating moves and finitely many choices for each move
  - Three players?
  - Multiple moves per turn?
  - Continuous games?
- Minimax is $O(b^d)$, where $d$ is the depth and $b$ is the branching factor
  - Explores the entire tree, so not very fast
- Alpha-beta is $O(b^d)$ in worst case, $O(b^{(d/2)})$ in best case, and $O(b^{(3d/4)})$ on average
  - Only has to explore player 1's moves
Applications & Drawbacks

- Dijkstra applied to tree search is $O(b^d)$
- A* depends on heuristic
  - Both need a specific marked goal node, rather than a number of win leaves
- Tree search algorithms such as alpha-beta are often used for games like Tic-Tac-Toe and chess
  - But they rely on low branching factor
- Alpha-beta for Realm of Kaodith?
  - ~8 to 10 branches per move, ~5 to 20 moves per turn \( \Rightarrow b \approx 1000000000000000 \)
  - Highly commutative within a turn
Applications & Drawbacks

● Alpha-beta for Realm of Kaodith?
  ○ No obvious minimax heuristic (that depends on piece positions)
  ○ Fog of War means incomplete information
  ○ A few minor continuous elements

● TBS is too open-ended for tree searches at the individual move level
  ○ Hierarchical planning?
An AI for TBS using Resource Assignment (RAA)

- Scenario: 4X game
- How should resources (the ships) be allocated to the objectives?

1. Colony defense
2. Attacking enemy colonies
3. Colonizing planets
4. Attacking ships
5. Repairing ships
6. Exploring space
RAA

- Solution: generate scores for each ship/task combination, and assign the best task for each ship
  - The scoring heuristic weights the general priority of a task (protect your colony before exploring) as well as local considerations (this ship is weak or damaged or far away)

$$\text{assignment score} = \frac{6 - \text{general priority} + \text{modifier}}{\text{distance to ship that is assigned}}$$
● **Pseudocode**

● **When to call algorithm?**
  ○ Discard all old tasks at the start of turn (to reflect new state)
  ○ Sometimes need to reallocate resources in mid-turn if exploration yields new tasks

● **Results**
  ○ "Unexpected tactic changes"
    ■ New units at base can affect allocation everywhere

"One got the impression that there was some sort of real intelligence behind the control of the enemy fleet"
Conclusion

- TBS games are too complicated for naïve tree search or graph search algorithms
- Algorithms for TBS must use hierarchical planning to be at all tractable
- Best results from local (this turn) considerations based on current threats and resources
- RAA is simple enough to implement in a week
Questions?