S191251

AI: Interactive Agents Report
Spaceship Wars

Version 1.0

S191251

Sections

Introduction

Spaceship Wars is an AI driven game where two spaceships engine in space based combat with each other. Each spaceship must use its limited ammo to try and destroy the opposing spaceship. When a spaceship is low on health and ammo it searches for either ammo or health pickups in the game world. If a spaceship destroyed a winner is declared and the game ends.
Spaceship wars is an incomplete project, so this document analyzes theoretical techniques that would have been used if spaceship wars was a complete project.

S191251

Spaceship Wars Game

This section lists the AI techniques used to create the Spaceship Wars game.

Each spaceship is controlled by a state machine that has three states, patrol state, attack state and flee State.

A navigation graph is used by the two spaceships to find the most optimal path to patrol randomly, find the quickest path to the enemy spaceship and find ammo and health powerups in the game world.

Asteroids are driven by steering behaviors, in particular the wandering behavior. If an asteroid collides with a spaceship, the spaceship takes damage.

Finite State Machines

Spaceship wars uses a finite state machine to control the behavior of the two spaceships. The state machine for the spaceships has only three states, a patrol state, attack state and flee state.

- In patrol the spaceship will search for a random navigation path via the A* search algorithm.
- In attack state the spaceship will fire at the enemy spaceship, if within sight range and if it has enough ammo.
- In flee state the spaceship will find health and ammo pickups, if low on health and ammo.

The finite state machine is based on the state machine implementation in Programming Game AI by Example. This implementation uses templates which enables a generic type to be used, which make this implementation extendable to other AI controlled entities if desired.

Pathfinding

Navigation in spaceship wars is achieved via a graph data structure.. The navigation used in spaceship wars is based on the AI pathfinding implementation in Game Coding Complete. The navigation graph used in spaceship wars contains a set of navigation nodes, these navigation nodes are connected together via navigation links. These navigation links then form a navigation graph which then can be used by a pathfinding algorithm to create a navigation path. The navigation path is used by the spaceships to find a random patrol path.

Spaceship wars uses the A* or A star pathfinding algorithm for its navigation graph pathfinding. A star is a computer algorithm used in pathfinding, which is finding a path between multiple nodes. There are there are other pathfinding algorithms spaceship wars could use, the two alternatives are Dijkstra's Algorithm and Breadth First Search. Breadth first search explores nodes in all directions, which isn't really useful for our needs. Dijkstra's algorithm explores specific nodes in a graph via their assigned cost, lower cost paths will be considered first. A star (which is a modification of Dijkstra's algorithm) on the other hand is optimized for a finding a single target node in a graph. A star finds paths navigating to one target node, prioritising paths leading to the target node.

A star is prefered because its performance and accuracy, which why I choose A star over other search algorithms.

An in depth discussion of the A star search algorithm with sample code is omitted for brevity. This link shows a C++ implementation of the A star search algorithm, *C++ Implementation* [link].

It must be noted that A* is not the most efficient algorithm for grid-based pathfinding. A more efficient for algorithm for grid-based pathfinding is the JPS+ algorithm. JPS+ is a specially designed algorithm for grid-based pathfinding. Spaceship isn't really grid-based, but if it was I would have implemented the JPS+ algorithm for the pathfinding instead. An article on the JPS+ algorithm is linked here, *JPS+ An Extreme A* Speed Optimization for Static Uniform Cost Grids* [link].

There are alternatives data structures for navigation and pathfinding that I could have used for spaceship wars. I discuss two alternatives to navigation below and give examples of their usage. The two alternatives to graph based structure are path nodes and navigation meshes.

Path nodes also known as waypoint graphs. Sanjay Madhav (2018, p. 115) states that waypoint graphs were popular in first-person shooter games during the 90s. This approach involves the placement of nodes in the game world the AI can navigatio to.

Any nodes placed in the game world are represented within the nodes within the graph.

Navigation meshes are another abstract data structure that can be used for AI navigation. In a navigation mesh data structure each node in the graph corresponds to a convex polygon. Any adjacent nodes in the graph are adjacent convex polygons, these convex polygons can then be used to cast entire game levels. This approach enables the AI to have more maneuverability within the game world, because the AI can traverse a single convex polygon node. Navigation meshes are usually generated automatically. Unfortunately the algorithms used to generate navigation meshes are far too complex to discuss here.

S191251

Steering Behaviors

Steering behaviors in spaceships wars are used to improve the spaceships pathing through the game world. Seek behavior is used when a spaceship is within sight range of another spaceship.
The seek steering behavior uses the steering force to smoothly adjust the spaceships velocity to reach the target spaceship.

Spaceships wars also uses steering behaviors to control the asteroids wandering behavior. To generate truly random wandering behavior I used an alternative implementation proposed by Craig W. Reynolds.
Reynolds idea was to produce small random displacements to the boids current velocity every frame. Since the boids velocity determines its direction and its speed every frame any small deviation with the velocity vector will change the boids path. Small displacements added to the boids velocity vector prevents the boids from abruptly changing their path, which does not make for a very convincing wandering behavior.
This approach is implemented in spaceship wars via a circle in front of the boid or asteroids in our case. The displacement force is then calculated from the circle's center and constrained by the circle's radius, a larger circle radius will increase the force the asteroids receives each frame. The displacement force is then used to calculate the asteroids wandering force, which will then create deviations in the asteroids velocity.

Conclusion

The AI techniques used in spaceship wars are sufficient for creating the AI driven game desired. However there are alternate techniques which could have been used instead.

- The finite state machine could be replaced with a behavior tree or replaced with a utility AI system.
- The navigation system could be replaced with a 2D navigation mesh.

A behavior tree could be a viable replacement for the state machine, if the AI grew in complexity. A utility AI system is not really suitable here since the AI spaceship wars is not that complex.
A 2D navigation mesh is a more ideal solution than the navigation graph currently used in spaceship wars. Instead of using nodes in a graph to represent a path for the spaceship, a 2D mesh could be generated which will represent an area a spaceship is allowed to move around in. I think this would be a far better approach than the navigation graph that ended up using in spaceship wars.
Creating a 2D navigation mesh is a rather complex task, which is not discussed here. However this article here talks about creating 2D navigation meshes, *Generating 2D Navmeshes by Christopher Romstöck* [link]. An article on using navigation meshes with the A* algorithm is also linked here, *A* with navigation meshes by Matheus Cansian* [link].

Bibliography

McShaffry, MS, Graham, DG. (2012) *Game Coding Complete*. Fourth Edition. USA. Cengage Learning PTR.

Rabin, SR. (2013) *Game AI Pro: Collected Wisdom of Game AI Professionals*. USA. A K Peters/CRC Press.

Madhav, SM. (2018) *Game Programming in C++*. USA. Addison-Wesley Professional.

Buckland, MB. (2005) *Programming Game AI by Example*. USA. Wordware Publishing.

Fernando Bevilacqua, FB. (2012) *Understanding Steering Behaviors: Wander*. Available at: Link

Amit Patel, AP. (2019) *Implementation of A**. Available at: Link

Amit Patel, AP. (2018) *Amit's A* Pages*. Available at: Link

Amit Patel, AP. (2019) *Introduction to the A* Algorithm*. Available at: Link

Christopher Romstöck, CR (2014) *Generating 2D Navmeshes*. Available at: Link

Steve Rabin, SR. Fernando Silva FS. (2015) *JPS+ An Extreme A* Speed Optimization for Static Uniform Cost Grids*. Available at: Link