CAS - (Complex Adaptive Systems)

- mehrere zusammenhängende Elemente
- adaptiv (Anpassungsvermögen an Umfeld)
- Aus Erfahrung lernend

Links

- Buch: Kompexität von Klaus Mainzer
- An Introduction to Multiagent Systems - Michael Wooldridge. John Wiley & Sons 2009+
- Multiagent Systems: A modern approach
- Multiagent Systems: Algorithmic, Game-Theoretic and Lofical Foundations

Notes

deklarativ vs iterativ programmieren
Große Systeme != Komplexe Systeme
Agenten: Interne Verarbeitung, Externe Wahrnehmung
bdi - beliefs-desires-intention
Multiagentensysteme - Bottom Up, Sozialität, Kohärenz
Preis der Anarchie: Verhältnis zwischen keine Absprache und vollkommene Vorabsprache
Lineare-Systeme:
zum Faktor *n
Beispiel: 2*Rauschen auf 2*Sinuskurve
Nicht-Lineare-Systeme:
zum Faktor ^n
Beispiel: Rauschen² auf Sinuskurve²

Agentenentscheidungen

- 1: ad-hoc
- 2: Regelwerke (Komplexität der Regeln * Anzahl der Regeln)
- 3: Adaption

Nutzen

Belohnungsbilanz mit Regeln/Belohnungsbilanz ohne Regeln

Sprachen

- Logo
- Net Logo

Ideen:

Space Piracy Simulation

Entwickler:
Tim Beier

Software:
Engine: Unity 3D
Sprache: C#
IDE: JetBrains - Rider
Dokumentation: atom

Ausgangssitation:

- Eine begrenzte Spielwelt (z.B. 8x8 Felder)
- Rundenbasiert
- Piratenschiffe verschiedener Fraktionen erscheinen in der Spielwelt (2-10 Fraktionen)
- Transportschiffe fliegen durch die Spielwelt

Regeln:

- Neutrale Transportschiffe fliegen regelmäßig durch die Spielwelt
- Schiffe lagern Ressourcen
- Schiffe verlieren mit der Zeit Ressourcen (1 pro Tick)
- Besitzt ein Schiff keine Ressourcen mehr, so wird es zerstört
- Wracks enthalten sammelbare Ressourcen (50)
- Schiffe besitzen einen begrenzten Lagerraum für Ressourcen (100)
- Schiffe erhalten Ressourcen beim Zerstören von Schiffen
- Schiffe können schießen und verursachen Schaden in Ressourcen (50)
- Schiffe mit 100 Ressourcen spalten sich in 2 Schiffe mit je 25 Ressourcen (Spawn auf freiem, angrenzendem Feld)
  Neue Schiffe kopieren ihre Werte von ihrem Erschaffer (Fraktion, v-werte)
- Schiffe besitzen eine begrenzte Sichtweite
- Schiffe sehen die Ressourcenanzahl von sichtbaren Fraktionen/Flotten
- Ein Schiff, dass sich auf ein besetztes Feld bewegt erleidet Schaden(25) und bewegt sich nicht

Verbleibende Rohstoffe werden aufgeteilt)

(Optional: Schiffe hinterlassen nicht passierbare Wracks)
(Optional: Das Zerstören von Wracks gibt Ressourcen)
(Optional: Schiffe verursachen Flächenschaden bei Zerstörung)
(Optional: Neue Schiffe kosten 50 Ressourcen und Zeitpunkt wird vom Agenten gewählt

(Optional: Waffensysteme)
(Optional: Upgrades)
(Optionale Variable: Waffenschaden)
(Optionale Variable: Geschwindigkeit)
(Optionale Variable: Zielgenauigkeit)
(Optionale Variable: Schätzgenauigkeit)

Variablen

- Sichtweite(angrenzend)
- Scanner(angrenzend)
- Schießen(angrenzend)
- Schaden(50)

Selbstorganisation:
Jedes Schiff will seinen Ressourcengewinn maximieren

Aktionen
Übergänge zwischen allen Aktionen mittels Markov und Q-Werten möglich.
Eigene Matritzen für: Verhältnis von Freundanzahl - Feindanzahl -> (<0, 0, >0)

Aktionsmatrix: (-1, 0, +1)
1: Schießen -> Schießenmatrix(angrenzende Felder: V-Werte)
2: Scannen -> Errechnet für eine Runde Freund-Feind Verhältnis
Gibt eine Runde lang Boni auf Schießenmatrix & Bewegenmatrix abhängig für Felder mit Feind/Freund:
Schießenmatrix: Bonus auf Feld von Feind durch Paramerer v_schießenFeind (Startwert 0)
Bewegenmatriy: Bonus auf Feld von Feind durch Paramerer v_bewegenFeind (Startwert 0)
Schießenmatrix: Bonus auf Feld von Freund durch Paramerer v_schießenFreund (Startwert 0)
Bewegenmatrix: Bonus auf Feld von Feind durch Paramerer v_schießenFreund (Startwert 0)

3: Bewegen -> Bewegenmatrix(angrenzende Felder: V-Werte)
4: Nichts tun