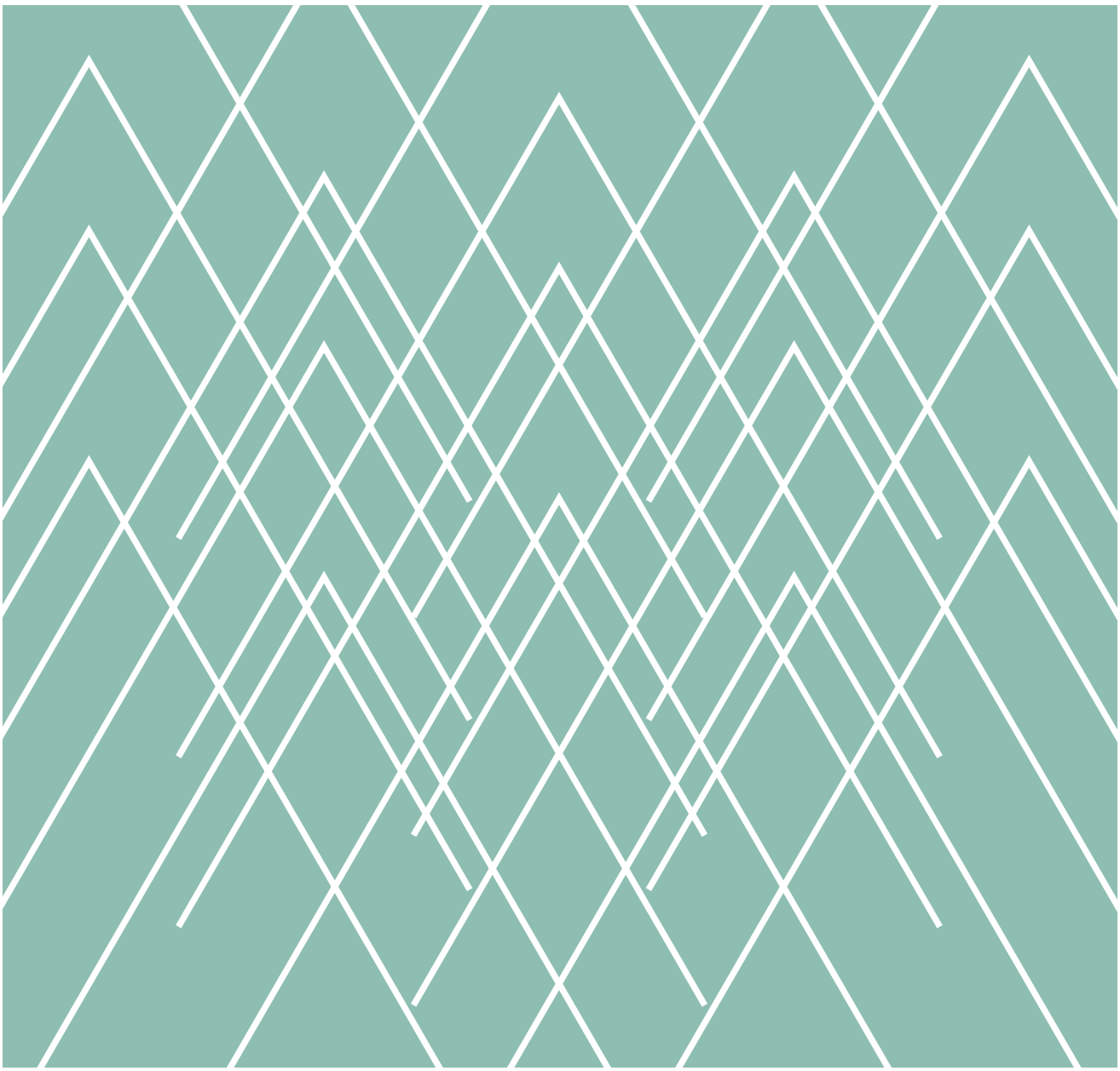
Erkennung von Regionen und Engpässen in Spielfeldern des Echtzeit-Strategiespiels Iron Harvest

Tanja Witke



Hochschule Harz
Hochschule für angewandte Wissenschaften

WAIT

Impressum

Inhaltlich verantwortlich

Autor/-in der Abschlussarbeit

Institution

Der Fachbereich Automatisierung und Informatik ist ein Fachbereich der Hochschule Harz. Die Hochschule Harz ist eine Körperschaft des öffentlichen Rechts. Sie wird durch den Rektor Prof. Dr. Folker Roland gesetzlich vertreten: info@hs-harz.de.

Umsatzsteuer-Identifikationsnummer

DE231052095

Adresse

Hochschule Harz
Fachbereich Automatisierung und Informatik
Friedrichstraße 57-59
38855 Wernigerode

Kontakt

Dekanin des Fachbereiches Automatisierung und Informatik
Prof. Dr. Andrea Heilmann
Tel.: +49 3943 659 300
Fax: +49 3943 659 300
E-Mail: dekanin-ai@hs-harz.de

Aufsichtsbehörde

Das Ministerium für Wirtschaft, Wissenschaft und Digitalisierung des Landes Sachsen-Anhalt (MW), Hasselbachstraße 4, 39104 Magdeburg, ist die zuständige Aufsichtsbehörde.

ISSN 2702-2293

Haftungsausschluss

Die Hochschule Harz weist auf Folgendes hin:

Die Hochschule Harz ist lediglich für die Veröffentlichung der einzelnen Werke zuständig, sie übernimmt keinerlei Haftung. Vielmehr gilt Folgendes:

- für den Inhalt der Publikation ist der/die Autor/-in verantwortlich
- mit der Erfassung in der Schriftenreihe Wernigeröder Automatisierungs- und Informatik-Texte verbleiben die Urheberrechte beim Autor/bei der Autorin
- die Einhaltung von Urheber- und Verwertungsrechten Dritter liegt in der Verantwortung des Autors/der Autorin

Vor Veröffentlichung bestätigte der/die Autor/-in,

- dass mit der Bereitstellung der Publikation und jedes Bestandteils (z.B. Abbildungen) nicht gegen gesetzliche Vorschriften verstoßen wird und Rechte Dritter nicht verletzt werden
- dass im Falle der Beteiligung mehrerer Autoren am Werk der/die unterzeichnende Autor/-in stellvertretend im Namen der übrigen Miturheber/-innen handelt
- im Falle der Verwendung personenbezogener Daten den Datenschutz (durch Einholen einer Einwilligung des Dritten zur Veröffentlichung und Verbreitung des Werks) zu beachten
- dass im Falle einer bereits erfolgten Veröffentlichung (z.B. bei einem Verlag) eine Zweitveröffentlichung dem Verlagsvertrag nicht entgegensteht
- dass die Hochschule Harz von etwaigen Ansprüchen Dritter (z.B. Mitautor/-in, Miturheber/-in, Verlage) freigestellt ist

Erkennung von Regionen und Engpässen in Spielfeldern des Echtzeit-Strategiespiels Iron Harvest

Bachelorarbeit
BA AI 14/2019

von Tanja Witke

März – Mai 2019

1. Prüfer: Prof. Jürgen Singer, Ph.D.

2. Prüfer: Prof. Daniel Ackermann

▲ Hochschule Harz
Hochschule für angewandte Wissenschaften

Hochschule Harz
Fachbereich Automatisierung und Informatik
Tanja Witke

Thema und Aufgabenstellung der Bachelorarbeit
BA AI 14/2019

Erkennung von Regionen und Engpässen in Spielfeldern des Echtzeit-Strategiespiels Iron Harvest

Bei der räumlichen Orientierung und dem taktischen Einbeziehen der Umgebung sind Menschen den herkömmlichen Künstlichen Intelligenzen aus Echtzeit-Strategiespielen wie z.B. StarCraft oder Command & Conquer weit überlegen. Um diesen Nachteil auszugleichen und die Herausforderung für den Spieler langfristig aufrecht zu halten, ist die Akquisition von Wissen aus komplexen und dynamischen Spielwelten von zentraler Bedeutung.

Ziel der Bachelorarbeit ist es dabei, den Teil der Spielfeldanalyse zu entwickeln, welcher Regionen und wichtige Engpässe findet, die später von der Künstlichen Intelligenz für taktische Manöver genutzt werden können. Die Berechnung soll ausschließlich im Unity-Editor stattfinden und könnte den Gamedesignern so bereits Hinweise für die Wahl günstiger Positionen strategischer Punkte geben. Da das Aktualisieren zur Laufzeit des Spiels nicht geplant ist, wird die Dauer der Berechnung kein relevantes Kriterium sein. Die Implementierung soll in der Programmiersprache C# erfolgen.

Die Bachelorarbeit beinhaltet folgende Teilaufgaben:

- Recherche bestehender Ansätze zur Findung von Engpässen
- Analyse zur Bedeutung dieses Wissens für spätere mögliche Entscheidungen der Künstlichen Intelligenz
- Erprobung eigener Ansätze bzw. Abwandlungen
- Entwicklung eines Prototyps
- Bewertung der Ergebnisse

Prof. Jürgen Singer, Ph.D.
1. Prüfer

Prof. Daniel Ackermann
2. Prüfer

Inhaltsverzeichnis

Glossar

Basis In der Basis können in Iron Harvest wichtige Gebäude errichtet werden, die verschiedene Einheiten bauen, ausbilden und heilen. Dort befindet sich auch das Headquarter, welches gegen den Feind verteidigt werden muss. (siehe auch Abschnitt 1.3 und [1]).

Corner-Graph Ein Corner-Graph soll nicht durch den Spieler gesteuerten Einheiten die Navigation durch das Spielfeld ermöglichen. Es handelt sich dabei um stark miteinander vernetzte Punkte, die an den Ecken von Hindernissen entstehen und oft genutzt werden, wenn es Einheiten mit unterschiedlichen Größen geben soll (siehe auch Abschnitt 1.4 und [2]).

Echtzeit-Strategiespiel Ziel bei einem Echtzeit-Strategiespiel ist es, eine militärische Stärke aufzubauen, um den Gegner beispielsweise durch das Zerstören seiner Basis zu besiegen. Dafür werden unter anderem Rohstoffe gesammelt, Gebäude gebaut oder verbessert und neue Einheiten trainiert. Alle Mitspieler handeln dabei gleichzeitig, also in Echtzeit (siehe auch Abschnitt 1.2 und [3]).

Engpass Engpässe dienen in dieser Arbeit als Übergänge zwischen Regionen und bieten unterschiedliche taktische Vorteile (siehe Abschnitt 2.3).

Influence Map Influence Maps sollen in Computerspielen nicht durch Menschen gesteuerten Agenten eine Entscheidungsgrundlage für räumliches Denken und Planen geben (siehe [4]).

Iron Harvest Iron Harvest ist das Echtzeit-Strategiespiel von KING Art Games, für das die Regionen und Engpässe gefunden werden sollen. Es spielt in einer alternativen Realität nach dem ersten Weltkrieg und zusammen mit der Infanterie, sprich den Fußsoldaten, ziehen dort auch große, mit Diesel betriebene Mechs in den Kampf (siehe auch Abschnitt 1.3 und [1]).

Künstliche Intelligenz (KI) Mit KI ist in Computerspielen eher das Modellieren von Verhalten oder das Design eines Agenten gemeint (siehe Abschnitt 1.1 und [5, S. 9F])

Mech Im Fall von Iron Harvest ist Mech ein Sammelbegriff für Exo-Skelette und große Maschinen (siehe auch Abschnitt 1.3 und [1]).

NavMesh	Ein NavMesh besteht aus mehreren miteinander verbundenen, konvexen Polygonen und soll nicht vom Spieler gesteuerten Agenten eine Navigationsgrundlage liefern. Somit kann das Finden von Wegen durch komplexe Umgebungen ermöglicht werden (siehe Abschnitt 1.4, Kapitel 6 und [6]).
Point-Of-Interest (POI)	POI's sind wichtige Punkte in Iron Harvest wie Flaggen oder Ressourcengebäude, die von Einheiten erobert und verteidigt werden können. Flaggen ermöglichen über Zeit das Sammeln von Siegpunkten während in Ressourcengebäuden Rohstoffe für neue Mechs, Gebäude oder Waffen gewonnen werden (siehe Abschnitt 1.3)
Priority-Queue	Eine Priority-Queue ist eine nach Priorität sortierte, abstrakte Datenstruktur. Während bei einer normalen Queue bzw. Schlange das erste eintretende Element auch als erstes die Struktur wieder verlässt, hat hier das Element mit der höchsten Priorität den Vorrang.
Region	Regionen sollen in dieser Arbeit zur Abstraktion des Spielfeldes dienen und durch Engpässe miteinander verbunden werden (siehe Abschnitt 2.2).
Spielfeld	Mit dem Begriff Spielfeld ist in dieser Arbeit immer die Umgebung gemeint, in der die Einheiten sich bewegen und gegeneinander kämpfen. Im Inneren befinden sich zahlreiche, teilweise auch zerfallene, Gebäude und andere Hindernisse, die die begehbare Fläche unterteilen. Einige Hindernisse können auch durch Mechs oder Minen zerstört werden (siehe auch Abschnitt 1.3).
Squad	Squads sind kleine Gruppen aus Infanterieeinheiten (siehe [1]).
Umweg-Bedingung	Die Umweg-Bedingung ist für einen Engpass erfüllt, wenn die Länge des Umweges ohne den Engpass eine gewisse Mindestlänge erreicht (siehe Abschnitt 2.3.4).
Unity	Unity ist die Spiel-Engine mit der Iron Harvest entwickelt wird.

Abbildungsverzeichnis

1 Einleitung

1.1 Künstliche Intelligenz (KI) in Computerspielen

Im Gegensatz zur KI in der Forschung, bei der es vor allem darum geht, intelligente Entscheidungen zu treffen und besonders schwierige Probleme wie das Verstehen von Sprache etc. zu lösen, ist jede Entscheidung einer KI in Computerspielen einem einzigen Ziel untergeordnet und das ist Spaß [5, S. 9]. Während es zum einen naheliegend erscheint, dass Spaß vor allem dadurch entsteht, ein Spiel zu gewinnen, erläutert Koster in seinem Buch „A Theory of Fun for Game Design“, dass das Meistern eines Spieles den eigentlichen Spaß bringe [7, S. 40]. Wäre das Spiel aber erst einmal gemeistert, gehe der Spaß an selbigem wieder verloren. Setzt man beispielsweise im Spiel Tic-Tac-Toe das erste Symbol in eine Ecke und der Gegner setzt anschließend nicht in die Mitte, kann man mit hoher Wahrscheinlichkeit gewinnen. Sind diese Muster erst einmal durchschaut, verliert das Spiel seinen Reiz. Entweder man gewinnt immer oder der Mitspieler kennt die Muster ebenfalls und das Spiel endet immer unentschieden. Spaß hängt also sehr stark mit der Herausforderung zusammen und diese kann von Spieler zu Spieler unterschiedlich sein und sich mit zunehmender Erfahrung des Spielers verändern. Folglich ist das Besiegen des Spielers bzw. das Treffen der bestmöglichen Entscheidung nicht das Ziel einer KI in Computerspielen. Durch das Einsetzen von beispielsweise unterschiedlichen Schwierigkeitsgraden, sollte eine besonders niedrige KI eher eine weniger gute Entscheidung treffen, um dem Spieler eine Chance zu geben. Trotzdem sollte das Verhalten intelligent wirken und die KI daher nicht den gleichen Fehler ständig wiederholen oder trotz einer Übermacht vor den Feinden fliehen. In vielen Fällen ist es für ein intelligentes Wirken vollkommen ausreichend, sehr einfache Strukturen zu verwenden, die trotzdem sehr komplexes Verhalten modellieren können. Ein guter Überblick über verschiedene Techniken ist in [8] gegeben.

Gerade in den letzten Jahren, kam es immer häufiger zum Einsatz modernerer KI Techniken wie dem Maschinellen Lernen (ML). Aufgrund der hohen Zustands- und Aktionsräume (siehe auch Abschnitt 2.1) in beispielsweise Echtzeit-Strategiespielen, werden diese hauptsächlich zum Lösen von Subproblemen eingesetzt. Da es bei der Strategie vor allem darauf ankommt, die Strategie des Gegners zu entschlüsseln und dann eine entsprechende Gegenstrategie zu entwickeln [9, S. 45], konnte beispielsweise in [10] durch ein Bayessches Model mit einer Wahrscheinlichkeit von über 70% nach den ersten 10 Minuten die Eröffnung des Gegners vorhergesehen werden. Ein guter Überblick über ähnliche State of the Art Ansätze ist auch in [11] gegeben. Da sich Echtzeit-Strategiespiele zu einem geeigneten Testbett in der KI-Forschung entwickelt haben, ist es schwer abzugrenzen, welche Ansätze in der Anwendung für das Spiel tatsächlich Verwendung finden können und welche rein für die Forschung relevant sind. Denn die Forschung konzentriert sich weitgehend darauf, den menschlichen Spieler zu besiegen und hat dies Anfang des Jahres mit AlphaStar auch geschafft [12], womit an die Erfolge von Deep Blue [13] und AlphaGo [14] angeknüpft werden konnte. Da momentan die menschlichen Spieler die herkömmlichen KI's in Echtzeit-Strategiespielen schnell durchschauen [3, S. 12], ist langfristig oft nicht die notwendige Herausforderung gegeben. Aus diesem Grund wird der KI oft auch mehr Wissen über das Spiel zur Verfügung gestellt, was ihr den nötigen Vorteil verschaffen soll. Ein besseres Planen von taktischen Manövern und alternativen Herangehensweisen, kann den Spieler positiv überraschen und das Spiel wieder interessanter gestalten. Dabei ist es nicht notwendig, dass der Plan auch wirklich aufgeht. Vielmehr wird ein hochwertiges Spielerlebnis dadurch kreiert, dass der Spieler eine gut durchdachte Taktik erfolgreich abwehrt oder von der KI sogar lernt, welche Positionen in der Karte besonders wertvoll sind. Alternativ zum Vorteil durch mehr Wissen scheint es auch naheliegend von der Forschung zu profitieren und Ansätze für die KI zu übernehmen. Allerdings bringen gerade Ansätze des ML neben den offensichtlichen Limitierungen durch Rechenleistung und Speicher von aktuellen Konsolengenerationen in der

Entwicklung von Spielen noch weitere Probleme mit sich [5, S. 9]:

1. Beim ML wird genau das gelernt, was durch die Daten vorgegeben wird. Wird eine KI nach den Daten eines inkompetenten Spielers trainiert, wird diese schlechte Spielweise auch übernommen. Das bedeutet im Umkehrschluss aber auch, dass professionelle Spieler für das Training benötigt werden.
2. Natürlich kann man KI's auch gegeneinander spielen lassen und sie anhand einer Fitnessfunktion lernen lassen. Diese ist für Spiele allerdings deutlich schwieriger zu implementieren. Während es in der Forschung ausreichend sein kann, nach Sieg und Niederlage zu bewerten, lässt sich „Spaß“ einer Funktion nicht wirklich zuordnen. So kann ein durchschauter und dadurch abgewehrter Hinterhalt dem Spieler eine Menge Spaß bringen, aber die Fitnessfunktion diese Taktik durch die Niederlage als unerwünschtes Verhalten werten.
3. Während der Entwicklung eines Spiels ändert sich das Gamedesign in den meisten Fällen ziemlich oft. Das liegt vor allem daran, dass es keine Formel gibt, die Spaß garantiert und dies somit nicht ausreichend planbar ist. Ansätze mit ML lassen sich nach dem Training aber kaum noch verändern, testen oder debuggen. Daher müsste der Lernprozess regelmäßig von vorne beginnen. Außerdem ist beim ML oft nicht bekannt, warum die KI ein bestimmtes Verhalten zeigt. Käme vom Gamedesign dann der Wunsch, dass die Einheiten beim Angriff beispielsweise etwas natürlicher und weniger geometrisch positioniert sein sollen, ist es kaum bis unmöglich die Daten entsprechend abzuändern und das Lernen somit gezielt zu beeinflussen.
4. Während des Spiels ist weiteres Lernen außerdem nicht notwendig, da die Einheiten oft nicht lange genug leben und das Balancing nicht verändert werden sollte.

1.2 Echtzeit-Strategiespiele

Folgend werden Echtzeit-Strategiespiele basierend auf der Arbeit von Ontañon et al. in [3] definiert. In Echtzeit-Strategiespielen übernehmen Spieler die Rolle eines Kommandeurs und erleben eine militärische Simulation. Im Gegensatz zu rundenbasierten Spielen, geht es vor allem um Schnelligkeit, da alles in Echtzeit geschieht und jede Aktion auch eine gewisse Dauer in Anspruch nimmt. Durch das Erobern von Ressourcengebäuden können Rohstoffe gesammelt und dadurch neue Gebäude gebaut oder bessere Ausrüstungen für Einheiten produziert werden. Häufig sind Teile des Spielfeldes, die noch nicht von Einheiten erkundet wurden, durch einen sogenannten „fog-of-war“ verdeckt. Im Laufe des Spiels geht es um die Verteidigung der eroberten Gebiete, das Aufbauen einer eigenen Basis, das Trainieren der Einheiten sowie das Zerstören der gegnerischen Armee und deren Basis. Um diese Ziele zu erreichen, kann nach verschiedenen Strategien vorgegangen werden. Diese gibt den grundlegende Plan an und wird auch Macro-Management genannt. Im Micro-Management hingegen geht es um ein detaillierteres Umsetzen von Taktiken oder Manövern. So könnte es beispielsweise Teil einer Strategie sein, einen wichtigen Stützpunkt des Gegners einzunehmen. Die Taktik könnte anschließend festlegen, dass von drei verschiedenen Richtungen aus angegriffen werden soll. Darunter gibt es noch die reaktive Kontrolle. Diese steuert das Verhalten einzelner Einheiten wie Zielen, Ausweichen, Bewegen etc.

Neben dem Problem der räumlichen (und auch zeitlichen) Orientierung, welches auch in dieser Arbeit betrachtet wird, werden in [15] noch fünf weitere Herausforderungen in der KI-Forschung für Echtzeit-Strategiespiele identifiziert. Diese sind das Management von Ressourcen, das Treffen von Entscheidungen unter Unwissenheit, das Zusammenarbeiten zwischen mehreren KI's, das Modellieren des Gegners und Lernen sowie die widersprüchliche

Echtzeitplanung.

1.3 Iron Harvest

Alle folgenden Informationen und Bilder sind der offiziellen Webseite des Spiels entnommen [1]. Das sich noch in der Entwicklung befindende Echtzeit-Strategiespiel Iron Harvest wird von KING Art Games entwickelt und spielt nach dem Ende des ersten Weltkriegs in einer alternativen Realität. Die ersten Alpha-Versionen wurden bereits veröffentlicht und somit kann das Spiel stetig direkt durch das Feedback der Spieler verbessert werden. Neben den Infanterieeinheiten, die in Gruppen bzw. Squads kämpfen, gibt es ca. 30 verschiedene große mit Diesel betriebene Mechs, Exo-Skelette und laufende Maschinen (vgl. Abbildung 1). Von gigantischen ultra-schweren und damit auch langsamen Kolossen, bis hin zu eher leichteren und schnellen Typen, werden den Spielern zahlreiche strategische Möglichkeiten zur Verfügung gestellt. Die gesamte Umgebung des Spielfeldes lässt sich unter anderem durch Minen und Mechs zerstören, wodurch die Struktur der Hindernisse sich ständig verändern kann (vgl. Abbildung 2).

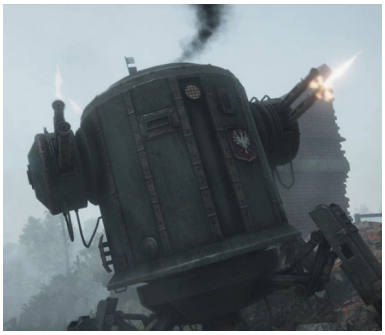


Abbildung 1: Ein typischer Mech in Iron Harvest

Abbildung 2: Zerstörung eines Mechs und der umliegenden Umgebung

Um eine geeignete Verteidigung gegen solch übermächtige Maschinen aufzubauen, gibt es neben Bunkern (vgl. Abbildung 3) auch Anti-Mech Kanonen sowie schwere Maschinengewehre

Abbildung 3: Bunker in Hellgrün bei der Platzierung vor einer Brücke (Engpass)

und Mörser. Im Laufe des Spiels wird versucht die Points-Of-Interest (POI's) zu erobern, um Ressourcen- und Siegpunkte zu sammeln und somit schnell eigene Mechs und Gebäude bauen zu können. Die meisten Gebäude können ausschließlich in der Basis gebaut werden. Diese muss besonders gut verteidigt werden, da nur von dort aus neue Einheiten nachrücken können.

In den Abbildungen 4 und 5 ist ein typischer Aufbau dieser Spiefelder zu sehen. Oft gibt es auch Flüsse, die mehrere Inseln voneinander abgrenzen. Zum anderen Ufern gelangt man in solchen Karten lediglich über Brücken, die nicht immer von allen Einheitenbreiten passiert werden können. Obwohl die Spielfelder alle in 3D aufgebaut sind, finden Simulationen und die komplette Wegfindung lediglich in 2D statt.

Abbildung 4: Typisches Spielfeld in Iron Harvest

Abbildung 5: Draufsicht eines weiteren Spielfeldes

1.4 Wegfindung in Echtzeit-Strategiespielen

Bevor mit Hilfe von verschiedenen Suchalgorithmen wie dem A* [16] nach Wegen in Spielfeldern gesucht werden kann, muss der begehbare Bereich in einer Struktur repräsentiert werden. Von dieser Struktur ist stark abhängig, wie genau aber auch wie schnell die Wegfindung erfolgen kann. Folgende Varianten werden häufig als Struktur verwendet.

1. Raster [2, S. 88ff]

   Durch quadratische, rechteckige, hexagonale oder dreieckige Raster-Strukturen lässt sich ein Spielfeld besonders leicht repräsentieren. Je nachdem wie stark die Auflösung jedoch ist, wird mehr Speicher benötigt und das Finden von Wegen dauert länger. Ein starker Vorteil ist hingegen die schnelle Zugriffszeit (O(1)) um herauszufinden, welches Kästchen der aktuellen Position am nächsten liegt. Alle folgenden Ansätze müssen einen Graphen nach dem Node durchsuchen, welches der aktuellen Position am nächsten liegt.
   Da beispielsweise durch ein rechteckiges Raster außerdem nur 90°-Drehungen möglich sind, um in eines der angrenzenden vier Felder zu kommen, würden diagonale Wege immer im Zickzack verlaufen. Dieses Problem wird durch sogenanntes Fadenziehen [6] gelöst. Da jedes Kästchen innerhalb des Rasters nur entweder begehbar oder nicht begehbar sein kann, ist oft nicht die gesamte Fläche abgedeckt, wie man in Abbildung 6 erkennen kann.

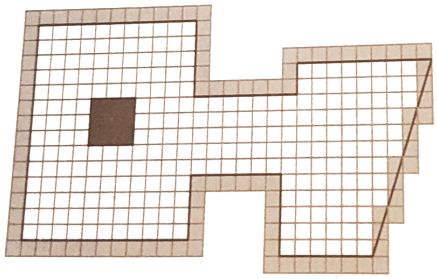


Abbildung 6: Rechts ist an der schrägen Kante sind einige Kästchen nicht begehbar. Die Fläche ist also nicht komplett abgedeckt [2, S. 88].

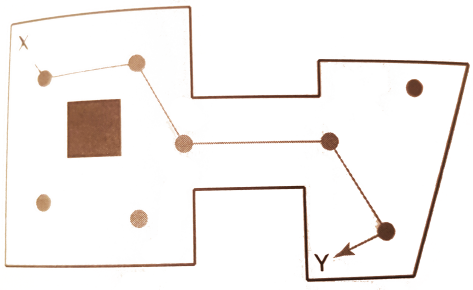


Abbildung 7: Der Weg von X (oben links) nach Y verläuft durch die Positionen der Waypoints zackig [2, S. 93].

2. Waypoints [2, S. 92ff]

   Waypoints können beispielsweise von Leveldesignern gezielt gesetzt und so an jeder Stelle einer 3D Welt platziert werden. Anschließend ist das Testen der Verbindungen aber oft mit einer hohen Zeitkomplexität ($O(n^2)$) verbunden, da dies für jedes mögliche Punktpaar notwendig ist. Als Ergebnis erhält man einen Graphen, über den anschließend navigiert werden kann. Dieses findet bei Waypoints etwas schneller statt, weil es in der Regel deutlich weniger Waypoints als Kästchen in einem Raster gibt. Auch hier ist der gefundene Weg am Ende nicht optimal, weil die Wege sich entlang der Punkte orientieren müssen (vgl. Abbildung 7).

3. Corner-Graph [2, S. 90ff]

   Diese Variante wird zur Zeit in Iron Harvest verwendet und eignet sich vor allem für die Wegfindung von unterschiedlichen Einheitengrößen. Dabei handelt es sich ebenfalls um Waypoints. Diese werden allerdings gezielt an den Ecken von Hindernissen platziert. Für jede Einheitengröße kann man den Abstand zur Ecke durch den jeweiligen Radius der Einheit nach außen setzen. Am Ende hat man für jede Einheitengröße auch einen eigenen Graphen. Da Corner-Graphen wie in Abbildung 8 jede mögliche Kante zu anderen sichtbaren Punkten bilden, ist das Erstellen der Verbindungen ziemlich teuer.

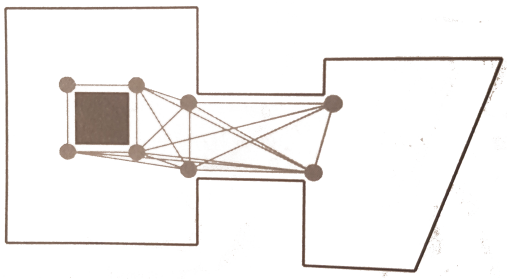


Abbildung 8: Corner-Graph mit allen gefundenen Verbindungen [2, S. 90]

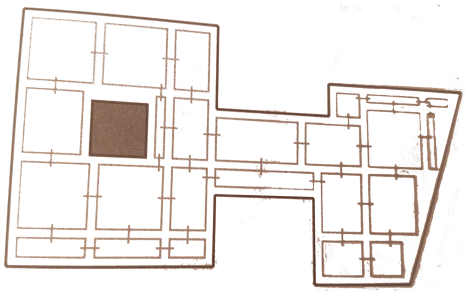


Abbildung 9: Die Rechtecke können die Fläche an der rechten schrägen Kante nicht komplett abdecken [2, S. 95].

4. Space-Filling Volumes [2, S. 95]

   Das Platzieren kleiner Rechtecke (oder Quader in 3D) in regelmäßigen Abständen innerhalb der begehbaren Fläche und anschließendes Ausdehnen stellt eine weitere Variante dar. Das Ausdehnen stoppt für jede Box sobald diese an ein Hindernis oder eine andere Box stößt. Alternativ können auch Raster verwendet werden, die sich anschließend zu größeren Rechtecken verbinden. In komplexen Umgebungen (mit nicht an den Achsen orientieren Kanten der Hindernisse) ist allerdings oft keine genaue Abdeckung gegeben (vgl. Abbildung 9). Durch das Verbinden mit allen angrenzenden Rechtecken, entsteht auch hier wieder ein Graph.

5. NavMesh [2, S. 95-98]

   Ein NavMesh [6] besteht aus verbundenen konvexen Polygonen und kann durch verschiedene Varianten erstellt werden. Da jede konkave Flache in konvexe Polygone unterteilt werden kann, ermöglicht diese Struktur auch für komplexe Spielfelder eine optimale Abdeckung. In [17, S. 173] beschreiben Hale et al. verschiedene Vorteile durch das Einteilen des Spielfeldes in ein NavMesh. So könne man in den einzelnen Polygonen Informationen speichern, die von den Agenten bei der Navigation mit berücksichtigt werden können. Durch die Konvexität könne man sich außerdem darauf verlassen, dass für einen Agenten alle Objekte innerhalb des Polygons seines Standortes für ihn sichtbar sind. Außerdem könne das NavMesh als Ersatz für die Kollisionserkennung verwendet werden. Dafür müsse man lediglich überprüfen, ob die Position des Agenten innerhalb eines der Polygone liegt. Da diese Position sich von einem Frame in den nächsten kaum verändert, könne man sich das zuletzt gefundene Polygon merken und von dort eine Breitensuche starten. Das NavMesh wird als beste der genannten Varianten beschrieben, weil man ähnlich wie beim Corner-Graphen auch hier entlang der Verbindungskanten abhängig vom Radius der jeweiligen Einheit den gefundenen Weg nach außen „schieben“ kann. Verschiedene Varianten zur Erstellung eines NavMeshs werden in Kapitel 6 genauer erläutert.

2 Grundlagen

Bevor die Ansätze zur Findung von Regionen und Engpässen näher erläutert werden können, muss geklärt werden, warum dieses Wissen für die KI und vor allem auch für das Spiel überhaupt wichtig ist. In diesem Kapitel werden wichtige Grundlagen und Definitionen vorgestellt, die für das Verständnis des Hauptteils der Arbeit von Bedeutung sind.

2.1 Wissensakquisition und Wissensabstraktion

Die Anzahl legaler, erreichbarer Spielpositionen lässt sich für sehr einfache Spiele berechnen und wird als Zustandsraum bezeichnet. Für komplexere Spiele hat es sich hingegen angeboten, lediglich eine obere Grenze anzugeben, die auch illegale Werte mit in Kauf nimmt. In [18, S. 158] wird der Zustandsraum am Beispiel des Spiels Tic-Tac-Toe beschrieben. Als obere Grenze wird zunächst 3^9 = 19 683 bestimmt, da an jedem der neun Felder jeweils ein Kreuz, ein Kreis oder gar kein Symbol stehen könnte. Grenzt man dies etwas weiter ein, erhält man 6 046, da die Anzahl von Kreisen und Kreuzen sich nur um einen Wert unterscheiden darf. Zieht man anschließend noch alle Zustände ab, die erst nach dem Gewinnen erreicht werden würden und somit illegal sind, erhält man den Wert 5 478. Bis zu einer gewissen Komplexität können auch einfach alle Möglichkeiten auf ihre Legalität überprüft werden. Ansonsten sind Standard KI-Techniken wie Suchbäume ohne eine Abstraktion nicht praktizierbar [3, S. 2]. Vergleicht man nun den Zustandsraum für Spiele wie Schach (10^{47} [19]) oder Go (10^{171} [20]) mit dem Zustandsraum von beispielsweise StarCraft (10^{2791} [21, S. 4]), erkennt man, warum eine Abstraktion notwendig wird, um die Anzahl der möglichen Aktionen zu verringern. Diese Anzahl kommt durch die Menge an möglichen Einheiten, den möglichen Aktionen für jede Einheit, den möglichen Positionen jeder Einheit auf dem Spielfeld und zuletzt auch durch die hohe mögliche Frequenz der Entscheidungen aufgrund der Echtzeit zustande.

Lediglich das Reduzieren des Zustandraums für die Entscheidungsfindung ist allerdings nicht ausreichend. Gleichzeitig sollte auch geprüft werden, welches Wissen für eine Entscheidung überhaupt notwendig ist. Möchte man beispielsweise entscheiden, ob es sinnvoll wäre, den Gegner jetzt anzugreifen, wäre das Wissen über die Stärke und Anzahl von gegnerischen aber auch eigenen Einheiten hilfreich. Darüber hinaus können auch Informationen über die Umgebung und eventuelle taktische Vorteile nützlich sein. Während menschliche Spieler aufgrund ihrer Erfahrung und der Fähigkeit des räumlichen Denkens solche Vorteile sehr einfach erkennen können, muss der KI dieses Wissen für eine Interpretation zur Verfügung gestellt werden.

2.2 Regionen

2.2.1 Definition

Eine Region wird unter anderem als „durch bestimmte Merkmale (z.B. Klima, wirtschaftliche Struktur) gekennzeichneter räumlicher Bereich“ beschrieben [22]. Im Falle von Iron Harvest soll eine Region als Abstraktion der begehbaren Fläche eines Spielfeldes dienen. Weiter oben wurde bereits erläutert, warum eine Abstraktion überhaupt notwendig ist. Ziel ist es hierbei, Regionen mit möglichst großer Aussagekraft über die begehbare Fläche zu erzeugen. Diese lässt sich auf unterschiedliche Weise darstellen. Zum einen könnte es sich um ein Raster handeln, bei dem es begehbare und nicht begehbare Kästchen gibt. Zum anderen könnte es sich auch um ein oben beschriebenes Polygon mit Löchern handeln, bei dem jedes Loch für ein Hindernis in der Karte steht.

In [23, S. 168] beschreibt Perkins Regionen auch als verbundene Komponenten aus begehbaren

Kästchen. Diese sollten die Bewegung von Einheiten innerhalb nicht einschränken und somit durch Engpässe voneinander abgegrenzt werden.

2.2.2 Informationen durch Regionen

Für die räumliche Orientierung ist es wichtig zu wissen, wie die Regionen miteinander verbunden sind. Mit Hilfe dieses Wissens könnte man Wege sehr schnell planen, da der Suchraum extrem verkleinert wäre und man beispielsweise nicht den detaillierten Weg von Berlin nach Bangkok suchen müsste, sondern durch die Abstraktion zunächst nur die Flugrouten von Berlin über Frankfurt nach Bangkok finden würde. Die detaillierte Suche könnte dann schrittweise vom aktuellen Standort bis zum Berliner Flughafen und dann von dort wieder bis Frankfurt erfolgen. Dieses Vorgehen wird als hierarchische Wegfindung bezeichnet [24].

Dill weist in [25, S. 372f] darauf hin, dass man die gefundenen Wege auch speichern könne und somit nicht ständig neu suchen müsse. So würde ein zweidimensionales Array Schritt für Schritt den Weg zum Ziel verraten, wenn es immer speichern würde, in welche der angrenzenden Regionen man als nächstes gehen müsste, um langfristig den kürzesten Weg zum Ziel zu finden. Das heißt, würde ein Weg wie in Abbildung 10 bei Region 2 starten und dann über 4, 1, und 5 nach 3 zum Ziel führen, würde das Array an der Position |2||3| die Id 4 beinhalten. Als nächstes müsste man also überprüfen, wie man von der Region 4 dann zu 3 gelangen könnte und an der Position |4||3| wäre entsprechend der Wert 1 gespeichert. Dies geht solange weiter, bis die Position |5||3| dann schließlich den Wert 3 beinhaltet. Laut Dill könne eine solche Tabelle wie in Abbildung 11 sowohl dynamisch zur Laufzeit, als auch bereits davor berechnet werden. Dill weist in [25, S. 373] außerdem darauf hin, dass man sich die Hälfte der Speicherungen sparen könnte, sofern die Struktur es erlaubt und der umgekehrte kürzeste Weg von Region 2 nach 3 dem kürzesten Weg von 3 nach 2 entspräche.

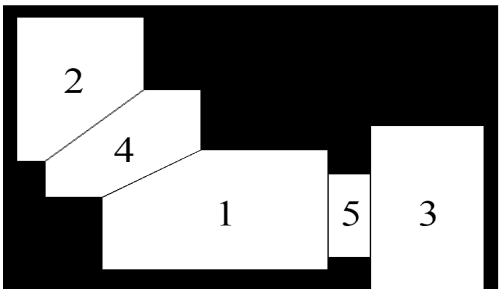


Abbildung 10: Der Weg von Region 2 nach 3 führt über 4, 1 und 5

	1	2	3	4	5
1		4	5	4	5
2	4		4	4	4
3	5	5		5	5
4	1	2	1		1
5	1	1	3	1	

Abbildung 11: Die Tabelle zeigt den kürzesten Weg zum Ziel

Durch die Verbindungen zwischen den Regionen erhält man außerdem Informationen über die Anzahl der Zugänge. Dies kann bei taktischen Manövern helfen, bei denen der Feind beispielsweise von mehreren Seiten angegriffen werden soll, gibt darüber hinaus aber auch an, an wie vielen Stellen man die Region verteidigen müsste. Letzteres ist für Iron Harvest hauptsächlich von Bedeutung, wenn die Region interessante Ressourcengebäude oder Flaggen enthält. Auch diese Information sollte entsprechend gespeichert werden.

In [25, S. 370f] beschreibt Dill ebenfalls, dass ein Einteilen in Regionen die Komplexität für die Entscheidungsfindung stark reduzieren kann. Statt in einem Raster beispielsweise beim Erkunden der Karte jedes Kästchen einzeln durchzugehen, könne durch die Abstraktion in Regionen aus deutlich weniger Möglichkeiten gewählt werden. Auch die Entscheidung, wo und wann ein Angriff sinnvoll wäre, ließe sich über Regionen treffen, die Werte wie die Stärke der Truppen des Gegners, Stärke der eigenen Truppen und den taktischen Wert der jeweiligen Region speichern würden. Man wisse so genau, ob ein Senden von Unterstützung notwendig wird, weil man den Ausgang des Kampfes bereits abschätzen könne.

Eine Möglichkeit die dafür notwendigen Werte zu berechnen, seien Influence Maps [4]. In Iron Harvest wurde bereits versucht, solche zu implementieren und in Entscheidungen mit einzubeziehen. Allerdings waren die Karten zu groß, um die Werte mit einer Rasterstruktur zur Laufzeit regelmäßig updaten zu können. Nimmt man nun aber Regionen als Grundlage, gibt es durch die Abstraktion weniger Felder und sowohl der Speicher wäre entlastet, als auch die Rechengeschwindigkeit für das Ausbreiten der Einflüsse wäre verringert.

Da Regionen die Fläche vereinfachen/abstrahieren sollen, ist es wenig hilfreich bestimmte Formen oder Größen festzulegen und dadurch eine Unterteilung der Fläche künstlich zu erzwingen. Eine große Region verrät, dass das Spielfeld an dieser Stelle sehr groß und offen ist und man dort eine größere Anzahl an Einheiten sammeln oder größere Schlachten führen kann. Wenn es außerdem nur zwei schmale Zugänge zu der Region gibt, ist es vollkommen egal, ob es sich um eine riesige Insel mit Brücken oder eine kleine eigene Basis handelt. Beides lässt sich sehr gut an diesen zwei schmalen Zugängen verteidigen. Wäre die Insel aber künstlich unterteilt, weil sie eine maximale Größe überschreitet oder andere Eigenschaften nicht erfüllt, hätte die KI diese nützliche Information verloren. Die kleinen schmalen Zugänge weisen außerdem darauf hin, dass Engpässe ideale Stellen sein könnten, um die Fläche in Regionen zu unterteilen. Das Finden von Regionen und Engpässen ist damit kein „getrennter Vorgang und sollte entsprechend zusammen erfolgen.

2.3 Engpässe

2.3.1 Definition

Ein Engpass wird als „schmale, verengte Stelle auf einem Weg, einer Straße, einem Durchgang o.Ä.“ [26] definiert. Allein das Verhältnis der Breite einer Stelle im Vergleich zur Breite der anliegenden Fläche ist also entscheidend. Als Enge wird außerdem ein „Mangel an Raum“ oder „räumliche Beschränktheit“ beschrieben [27]. In Bezug auf Verkehr spricht man bei Engpässen auch von einer „Stelle, die nur langsam oder von nur wenigen Verkehrsteilnehmern gleichzeitig passiert werden kann“ [28]. Muss der Gegner mit seinen Truppen einen solchen Engpass in einem Echtzeit-Strategiespiel durchqueren, kann er also durch die räumliche Beschränktheit im Nachteil sein. Damit eine solche Beschränktheit auch garantiert werden kann, kann es sinnvoll sein, eine Maximalbreite zu definieren. Diese wird in [25, S. 374] von Dill auch als „reasonably narrow“, also ziemlich oder angebracht eng beschrieben.

In [23, S. 168] beschreibt Perkins Engpässe auch als Korridore, die exakt zwei Regionen miteinander verbinden. In seinem Ansatz sucht er lediglich nach der engsten Stelle dieses Korridors und unterscheidet zwischen zwei verschiedenen Typen von Engpässen. Zum einen gäbe es Engpässe, die zugebaut zwei Flächen komplett voneinander trennen würden und zum anderen gäbe es welche, die zugebaut zwei Hindernisse miteinander kombinieren und somit vergrößern würden.

2.3.2 Taktische Vorteile

Wo genau ein Engpass wichtig ist, ob es eine Maximalbreite geben muss und wie groß diese sein sollte, hängt jeweils von folgenden taktischen Vorteilen ab:

1. Vorteil im Nahkampf (Nahkampf-Vorteil)

   Ist ein Engpass sehr eng und hat somit eine kleine Maximalbreite, sind die Einheiten des Gegners räumlich stark beschränkt und können nicht alle gleichzeitig an der Front kämpfen. Sie sind gezwungen sich aufzureihen und den Engpass hintereinander zu durchqueren, wie im Stau auf einer Autobahn. Die eigenen Einheiten hingegen sollten auf der anderen Seite stehen und genug Platz haben, um die Front der gegnerischen

Einheiten dann mit voller Kraft bekämpfen zu können. Die vordersten Einheiten des Gegners wären also direkt nach dem Verlassen des Engpasses von den eigenen Truppen umkreist. Da weitere Gegner durch den Engpass nur verlangsamt dazu kommen können, ist es möglich, selbst große Übermächte stark zu schwächen, aufzuhalten oder sogar zu besiegen. Ein dazu passendes berühmtes Beispiel ist die Schlacht bei den Thermopylen, in der eine im Verhältnis kleine griechische Armee die große Übermacht der Perser eine Zeit lang aufhalten und so den Rückzug decken konnten.

2. Vorteil im Fernkampf (Fernkampf-Vorteil)

   Für beispielsweise einen Bunker oder eine Kanone wäre es am besten, wenn der Einflussbereich die komplette Breite des Engpasses abdecken würde und die gegnerischen Einheiten somit nicht entkommen könnten. Da manche Einflussbereiche allerdings kegelförmig sind und daher mit zunehmender Entfernung immer breiter werden, lässt sich eine geeignete Maximalbreite dafür nur schwer festlegen. Außerdem ist ein solcher Engpass natürlich nur dann wichtig, wenn es überhaupt möglich ist, an dieser Stelle einen Bunker oder eine Kanone zu platzieren. Wie dicht diese dann an dem Engpass platziert wären und wie gut die Abdeckung tatsächlich wäre, könnte als Kriterium für die Berechnung von Priorisierungswerten der Engpässe dienen. Dabei muss unbedingt auch bedacht werden, dass ein Bunker, der zu dicht an einem Engpass steht, diesen gleichzeitig verschließen könnte. Dies sollte vermieden werden, falls der Gegner den Weg dann gar nicht mehr in Betracht ziehen würde oder auch, falls eigene Truppen den Engpass noch nutzen sollen.

3. Vorteil beim Einsatz von Fallen (Fallen-Vorteil)

   Für das Platzieren einer Mine o.ä. müsste ein Engpass so schmal sein, dass diese beim Durchqueren gegnerischer Einheiten mit einer sehr hohen Wahrscheinlichkeit auch ausgelöst wird und es nahezu keine Möglichkeit gibt, der Falle auszuweichen. Bei einer Mine muss zusätzlich natürlich bedacht werden, dass die angrenzenden Hindernisse bei einer Explosion mit zerstört werden könnten, wodurch ein taktisch wertvoller Engpass unter Umständen verloren ginge.

4. Vorteil bei der Flucht (Flucht-Vorteil)

   Da es in Iron Harvest Einheiten mit deutlichen Größenunterschieden gibt, kann ein besonders kleiner Engpass bei einer Flucht sehr hilfreich sein, wenn beispielsweise ein verfolgender Mech nicht hindurchpassen würde. Natürlich sollte man dabei trotzdem beachten, dass bestimmte Mechs ihre Umgebung zerstören können. Allerdings würde es wahrscheinlich zumindest einen zeitlichen Vorteil geben.

5. Vorteil durch Verschließen von Zugängen (Verschluss-Vorteil)

   Bei besonders wichtigen Regionen kann es sinnvoll sein, einen Teil der Zugänge nicht nur zu verteidigen, sondern mit Mauern und Gebäuden komplett zu verschließen. Das kann gerade zu Beginn des Spiels einen sehr entscheidenden Vorteil bringen, da es zu diesem Zeitpunkt meistens noch keine Mechs gibt, die diese Barrikaden einfach wieder zerstören könnten. Die gegnerischen Einheiten wären also gezwungen, einen anderen Zugang zu wählen, was unter Umständen einen großen Umweg bedeuten würde. Damit könnte man den Gegner außerdem gezielt in bestimmte Fallen oder generell in taktisch ungünstige Gebiete locken. Die Maximalbreite kann hierbei an sich beliebig groß sein. Man sollte nur beachten, dass man für breitere Engpässe natürlich eine längere Mauer benötigt, deren Bau mehr Zeit kostet und dass der Engpass nach dem Verschließen auch von eigenen Truppen nicht mehr passiert werden kann.

Man kann also festhalten, dass es höchstwahrscheinlich sinnvoll sein wird, mehrere verschiedene Maximalbreiten für Engpässe festzulegen und somit mehrere Abstufungen der

Breiten zu schaffen. Während es beim Fernkampf-Vorteil sehr breite Engpässe geben kann, die dann beispielsweise von Bunkern komplett unter Beschuss genommen werden können, muss die Maximalbreite bei einer Flucht sehr klein sein. Beim Verschluss-Vorteil ist sogar überhaupt keine Maximalbreite notwendig. Dort könnte die schmalste Stelle des gesamten Engpasses am besten geeignet sein.

2.3.3 Unterscheidung zwischen Engpass als Region oder als Stelle

Die Verwendung der Worte Stelle und Durchgang in der Definition deuten darauf hin, dass es sich bei einem Engpass sowohl um eine Region (nachfolgend Engpass-Region genannt), wie einen schmalen Gang, als auch um einen bestimmten Ort handeln könnte. Da im Zusammenhang mit Engpässen außerdem von schmal und eng die Rede ist, besitzt jeder Engpass eine Breite und ist somit am besten als Kante mit der Länge dieser Breite darstellbar. Beim Nahkampf-, Fernkampf- und Flucht-Vorteil aus Kapitel 2.5.2 sind lediglich die Übergänge der Engpass-Region entscheidend. Aus diesem Grund kann man diese auch als eigenständige Engpässe bezeichnen.

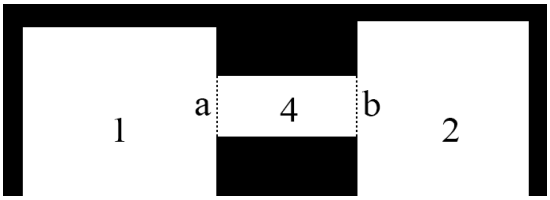


Abbildung 12: Die Regionen 1 und 2 sind durch die Engpass-Region 4 und den Übergängen a und b miteinander verbunden.

So ist die Region 4 in Abbildung 12 zwar eine Engpass-Region zwischen Region 1 und 2, aber Übergang a ist gleichzeitig ein Engpass von Region 1 und Übergang b gleichzeitig ein Engpass von Region 2. Nur für den Fallen- und Verschluss-Vorteil kann zusätzlich auch jede beliebige Stelle innerhalb der Engpass-Region den gleichen Vorteil bieten.

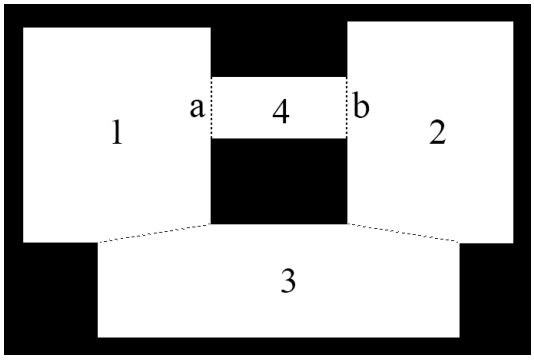


Abbildung 13: Die Engpass-Region 4 ist in diesem Fall bedeutungslos, weil es über Region 3 einen deutlich besseren Weg von 1 nach 2 gibt.

2.3.4 Kriterien für die Wichtigkeit

Die Überlegung, dass gegnerische Truppen auch ausweichen oder entkommen könnten, führt zu einem sehr entscheidenden Kriterium für die Wichtigkeit eines Engpasses. Betrachtet man in

Abbildung 13 einen größeren Ausschnitt der Karte aus Abbildung 12, kann man deutlich erkennen, dass man statt über Region 4 mit Region 3 deutlich einfacher von Region 1 in Region 2 kommen kann. In diesem Fall ginge die Taktik komplett verloren und solche Engpässe müssen in der Priorisierung aussortiert werden. Das bedeutet nicht, dass der Engpass der einzige Weg sein muss. Aber der Umweg, den man sonst einschlagen müsste, sollte so weit sein, dass dieser einen zu großen Nachteil für den Gegner bedeuten würde. Im späteren Text wird hierauf als Umweg-Bedingung verwiesen. Auch kleine Umwege können in Iron Harvest bereits einen entscheidenden Effekt haben, da besonders große Mechs sich sehr langsam bewegen und aufgrund ihrer Größe auch nicht immer den kürzesten Umweg nehmen können.

Allerdings kann ein Engpass, der alleine unter Umständen nicht die Umweg-Bedingung erfüllt, innerhalb eines Sets aus mehreren Engpässen, welche einen wichtigen Bereich der Karte abgrenzen und/oder einkreisen, wiederum sehr wichtig sein [25, S. 374]. Dies aber natürlich hauptsächlich in Kombination mit den anderen Engpässen des Sets. Gibt es beispielsweise insgesamt drei Zugänge zur eigenen Basis, aber zwei davon liegen direkt nebeneinander, haben diese zwar jeweils einen kleinen Umweg, sind für die Verteidigung aber trotzdem von Bedeutung. Auf der anderen Seite kann ein Engpass, der zwar die Umweg-Bedingung erfüllt, aber an einer unwichtigen Stelle wie dem Rand der Karte oder an einer Sackgasse steht, dadurch trotzdem unwichtig sein.

Dill empfiehlt in [25, S. 374] sich für jeden Engpass zu merken, wie viele gegnerische Einheiten diesen im Laufe des Spiels passieren und an den am stärksten besuchten Regionen dann entsprechend Fallen aufzustellen. Außerdem solle man besonders an den Engpässen innerhalb eines Sets Verteidigungen aufbauen, da man sie so gezielter und an weniger Punkten einsetzen kann. Ansonsten könne es sich auch lohnen, dort Wachen aufzustellen und so einen Angriff des Gegners rechtzeitig zu bemerken. Damit wäre auch dies ein entscheidendes Kriterium für die Wichtigkeit eines Engpasses.

3 Problem und Anforderung

Der Grund, weshalb für Iron Harvest die Engpässe ursprünglich von Bedeutung waren, war der Fernkampf- und der Verschluss-Vorteil. Engpässe bieten durch ihre taktische Lage ideale Stellen für die Verteidigung und sind somit interessante Punkte für die Platzierung von Bunkern oder auch Mauern. Natürlich ergeben sich wie in Kapitel 2.5.2 beschrieben darüber hinaus noch viele weitere Vorteile. Welche davon am Ende für Iron Harvest jedoch Verwendung finden sollen, ist noch nicht festgelegt. Aus diesem Grund sollte der Algorithmus nach Möglichkeit maximal modular und flexibel sein und über zahlreiche Parameter und Flags Einstellungsmöglichkeiten liefern, die das Ergebnis entsprechend beeinflussen können. Die Platzierung von allen Gebäuden durch die KI wird momentan über Hinweise der Leveldesigner gesteuert, die damit die taktischen Positionen vorgeben. Mit Hilfe eines Tools im Unity-Editor sollen den Designern unter Verwendung meiner Implementierung alle Engpässe angezeigt werden. Diese können dort noch gelöscht, umplatziert oder hinzugefügt werden, bevor sie als Point-Of-Interest (POI) dem Spiel und damit der KI übergeben werden. Diese Arbeit beschäftigt sich mit dem Problem der Abstraktion eines Spielfeldes und soll Regionen und Engpässe finden, die der KI für die Entscheidungsfindung zur Verfügung gestellt werden. Die Umsetzung dieser KI und das Nutzen dieses Wissens ist nicht Teil dieser Arbeit.

Um einer KI durch ML beizubringen, welche Engpässe in einer Karte von Bedeutung sind, müsste ihr ein großer Datensatz zur Verfügung stehen, der Wissen über die richtigen Positionen der Engpässe und Regionen bereitstellt. Da dieser Datensatz mit entsprechendem Wissen nicht vorhanden ist, wird in dieser Arbeit auf den Einsatz des ML verzichtet.

Damit verschiedene Ansätze schnell und einfach getestet werden können, wird zunächst lediglich ein Prototyp verwendet, der völlig unabhängig von Unity funktionieren soll. Ein solcher Prototyp bietet außerdem auch den Vorteil, dass man getrennt vom Spiel mit anderen Entwicklern aus der Branche über die Ansätze und Implementierungen sprechen kann, ohne noch unveröffentlichte Inhalte aus dem Spiel zeigen zu müssen.

Da der Algorithmus nicht zur Laufzeit des Spiels aktualisiert werden soll, ist die Zeit- und Speicherkomplexität nicht von Bedeutung.

Zur Verbesserung des bestehenden Systems in Iron Harvest soll außerhalb dieser Arbeit außerdem der aktuell verwendete Corner-Graph durch ein NavMesh ersetzt werden. Die Herausforderung dabei bilden vor allem die unterschiedlichen Einheitengrößen. Entsprechend sollte der Ansatz zum Finden von Engpässen nicht auf die Wegfindung durch den Corner-Graphen angewiesen sein und im Idealfall später sogar mit dem neuen NavMesh zusammenarbeiten können.

4 State of the Art

Sowohl beim Finden von Engpässen als auch bei der Unterteilung einer Karte in Regionen gibt es bereits zahlreiche Ansätze und Überlegungen.

Besonders hervorheben muss man das Spiel StarCraft, welches sich als interessantes Testbett für die Forschung im Bereich der KI erwiesen hat. Im jährlichen Wettbewerb AIIDE treten Forscher und Studenten aus aller Welt mit ihrer KI gegeneinander an und testen so, welche Fähigkeiten im Spiel besonders entscheidend sind. Ein großer Teil der Bots nutzt für die Terrain Analyse den von Perkins entwickelten BWTA [23], der 2009 zur offiziellen BWAPI [29] hinzugefügt wurde. In [30] wurden die dadurch gefundenen Regionen mit taktischen Werten aus einer Analyse der Wiederholungen von professionellen Spielern verbessert. In [31] stellen Uriarte und Ontañón die Verbesserung BWTA2 vor, die nicht nur 26 Mal so schnell, sondern auch wesentlich genauer Engpässe finden soll. Auch der von Dimitrijevic implementierte BWEM [32] bietet einen zeitlichen Vorteil gegenüber dem BWTA und speichert die kürzeste Entfernung zum nächstgelegenen Hindernis in einer Rasterstruktur.

Ebenfalls durch die Verwendung eines Voronoi-Diagrams wurde von Forbus et al. [33] die freie Fläche charakterisiert und für die Wegfindung verwendet. Die mittlere Axe verwenden Bidakaew et al. in [34], um schmale Region Engpässe zu finden.

Higgius [35] nutze zum Finden von Engpässen und Regionen die Idee von Auren, die sich um Hindernisse herum bilden und dann an den Engpässen überlappen oder zusammenwachsen. Ein ähnlicher Ansatz wurde in [36] von Obelleiro verfolgt. Er stellte außerdem fest, dass ein einziger Wert für die Größe dieser Auren zum einen nicht für jedes Spielfeld gleich gut funktioniert und zum anderen auch innerhalb einer Map mehrere verschiedene Größen sinnvoll wären. Halldórsson et al. [37] suchten Engpässe mit Hilfe eines Wasserlevel-Ansatzes und bauten diese als Tore in ihre Heuristik der Wegfindung mit ein. In [21, S. 15] wird durch Uriarte angemerkt, dass dieser Ansatz zwar ebenfalls schneller als der BWTA sei, er aber zu falschen Ergebnissen tendiere.

Ein Unterteilen des Spielfeldes in Rechtecke, aus denen dann Regionen gebildet werden, nutze Kevin Dill unter anderem für das Erkennen von Engpässen [25]. Diese Struktur bietet darüber hinaus auch viele andere Vorteile und Möglichkeiten, die der KI bei der Entscheidungsfindung helfen sollen.

In dem Echtzeit-Strategiespiel „Company of Heroes“ wurde zum Finden von Regionen in einer Rasterkarte ein Floodfill-Algorithmus verwendet [38].

Pottinger nutzte eine Technik mit konvexen Hüllen, um im Echtzeit-Strategiespiel „Age of Empires“ Zonen zu erzeugen [39].

Ein Ansatz durch Waypoints wurde außerdem von van der Sterren [40] beschrieben.

5 Analyse vielversprechender Ansätze

In diesem Kapitel werden zwei der vielversprechendsten Ansätze für das Finden von Regionen und Engpässen in Echtzeit-Strategiespielen im Hinblick auf ihre Anwendbarkeit für Iron Harvest analysiert. Einer davon soll später als Grundlage dienen, um das Finden von Engpässen und Einteilen in Regionen für Iron Harvest zu ermöglichen.

5.1 Brood War Terrain Analyser (BWTA)

Der Brood War Terrain Analyser (BWTA) aus [23], um den es im gesamten Abschnitt gehen soll, wurde 2010 von Perkins entwickelt und seit dem von zahlreichen Bots im AIIDE Wettbewerb verwendet. Uriarte und Ontañón entwickelten 2016 die Erweiterung BWTA2 [31]. Der Algorithmus ist grundlegend in acht Schritte eingeteilt.

1. Das Spielfeld wird vektorisiert, damit die Hindernisse als Polygone zur Verfügung stehen. Dieser Schritt könnte bei Iron Harvest eingespart werden, da die Hindernisse ihre Außengrenzen als Polygon bereits zur Verfügung stellen. Auch wurden für den Corner-Graphen zur Laufzeit des Spiels Überlappungen durch ein Clipping-Verfahren bereits zusammengefasst.
2. Durch die Implementierung aus [41] wird aus den Kanten der Polygone ein Voronoi Diagramm erstellt. Das Ergebnis ist in Abbildung 14 zu sehen.

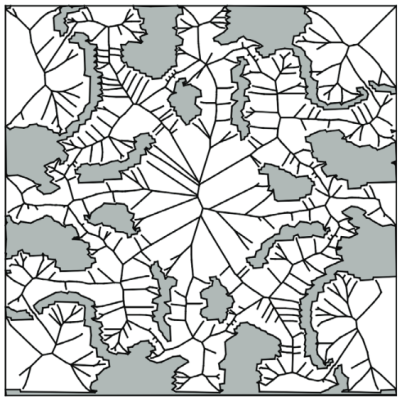


Abbildung 14: Ergebnis der Berechnung des Voronoi Diagramms [23, S. 170]

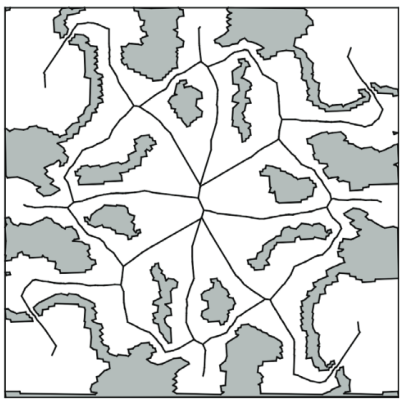


Abbildung 15: Ergebnis der Reduktion des Voronoi Diagramms [23, S. 170]

3. Für jeden Punkt innerhalb des Graphen wird der Abstand zum nächstgelegenen Hindernis berechnet. Dieser wird als Radius bezeichnet. Alle Blätter des Graphen werden anschließend entfernt, wenn ihr Radius kleiner als das ihres Elternteils ist. Wird das Elternteil dadurch ebenfalls zu einem Blatt, wiederholt sich dieser Schritt solange, bis es nur noch Blätter mit einem größeren Radius als den der jeweiligen Elternteile gibt. Abbildung 15 zeigt den dadurch stark reduzierten Graphen.
4. Alle Punkte des Graphen, die mehr oder weniger als zwei Verbindungen haben, könnten interessante potentielle Regionen repräsentieren. An diesen Stellen gibt es nämlich

Kreuzungen, Sackgassen oder komplett isolierte Flächen. Ein Punkt A mit genau zwei Verbindungen wäre laut Perkins nur interessant, wenn er einen bestimmten Radius überschreiten und ein lokales Maximum bilden würde. Dieses wird so definiert, dass alle Punkte innerhalb A‘s Radius einen kleineren Radius haben müssen.

5. Alle verbliebenen Punkte haben nun garantiert zwei Verbindungen und müssen entlang eines Pfades zwischen den Region-Punkten liegen. Der Punkt auf diesem Pfad mit dem kleinsten Radius bildet dann den Engpass. In Abschnitt 2.3.2 wurde bereits gezeigt, dass nicht unbedingt die engste Stelle eines Engpasses entscheidend ist, sondern vielmehr die Zugänge die Vorteile bieten. Dies wird von Perkins in seinem Ausblick auf spätere Forschung ebenfalls angemerkt. Durch die Information über die Radien der Punkte, würden sich aber sicherlich auch diese Zugänge ermitteln lassen. In Abbildung 16 sind alle gefundenen Regionen und Engpässe zu sehen.
6. Um die Anzahl der Regionen zu reduzieren, werden einige angrenzende Regionen miteinander verbunden und der dazwischenliegende Engpass gelöscht. Wann dies der Fall sein soll, wird jeweils durch ein Verhältnis zwischen dem Radius des Engpasses und dem der Regionen bestimmt. Ob eine Region mit genau 2 Engpässen wegfallen kann, wird ebenfalls über ein Verhältnis entschieden. Das Ergebnis ist in Abbildung 17 zu sehen. Laut Perkins können in Zukunft aber noch weitere Kriterien für das Zusammenfügen von Regionen sinnvoll sein und seine Variante biete keine optimale Lösung.
7. An jedem Engpass-Punkt wird nun die entsprechende Kante aus dem zugrundeliegenden Voronoi Diagramm ausgewählt und zwischen die Punkte der Hindernis Polygone eingefügt.
8. Zuletzt werden entlang der verbundenen Punkte die neuen Grenzen der inneren begehbaren Polygone gefunden und der jeweiligen Region zugeordnet.

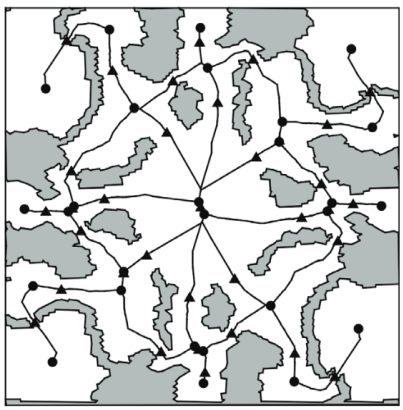


Abbildung 16: Alle gefundenen Regionen sind durch Punkte und alle Engpässe durch Dreiecke gekennzeichnet [23, S. 171]

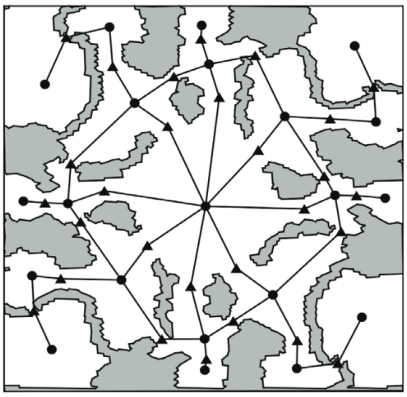


Abbildung 17: Ergebnis nach dem Zusammenfügen der Regionen [23, S. 171]

5.2 Architektur von Kevin Dill

Dieser Abschnitt bezieht sich komplett auf eine Architektur von Kevin Dill aus [25]. Diese wurde ebenfalls für ein Echtzeit-Strategiespiel namens „Kohan II: Kings of War“ entwickelt und soll eine Art räumliches Denken für die KI ermöglichen. Dafür möchte Dill eine Raster-Karte in aussagekräftige Regionen unterteilen. Mit dessen Hilfe soll die KI später Erkundungen durchführen, Positionen für Attacken finden, Wege planen oder besondere räumliche Merkmale, wie beispielsweise Engpässe finden. Dabei ist ein Engpass eine Region, welche die Karte möglichst gut in zwei Gruppen unterteilt. Eine Region besteht wiederum aus ein und manchmal auch mehreren Rechtecken und sollte folgende Eigenschaften aufweisen:

1. Sie repräsentiert einen bestimmten Typ Landschaft wie Wasser, Hügel oder Straßen. Diese Unterteilung würde für Iron Harvest wegfallen, da wir nur zwischen „begehbar“ und „nicht begehbar“ unterscheiden. In späterer Entwicklung des Spiels könnte man allerdings trotzdem darüber nachdenken. Beispielsweise falls es Mechs geben soll, die schwimmen oder durch Wasser laufen können. Oder falls man auf diese Art und Weise, das Durchlaufen einer Region mit bestimmten Kosten versehen möchte. Große Mechs können zudem Hindernisse durchlaufen oder zerstören. Wären diese Hindernisse also als Region in der Wegfindung mit einbezogen, könnten dadurch große Umwege vermieden werden.

2. Die Region sollte nicht „zu groß“ sein. Dill begründet dies damit, dass nicht mehr ausreichend Detail gegeben wäre, um aussagekräftige und genaue Entscheidungen treffen zu können. Er gibt auch an, dass man im Extremfall ja die komplette Karte als eine Region werten könnte, dies aber nicht hilfreich wäre. Ich sehe das etwas anders. Wie bereits in Abschnitt 2.4.2 erklärt, kann ein gezwungenes Unterteilen das Finden von hilfreichen Informationen wie der Anzahl der Zugänge verhindern. Außerdem geht es beim Abstrahieren ja genau darum, zu viel Detail zu vermeiden und möglichst viel Aussagekraft zu erzeugen. Durch die Begrenzung der Größe ist also hauptsächlich sichergestellt, dass alle potentiellen Engpass-Regionen Dills Anforderung „ziemlich eng“ erfüllen.

3. Die Region sollte auch nicht „zu klein“ sein. Auch hier finde ich eine kleine Region sehr angemessen, wenn es sich um eine kleine Fläche wie einen Engpass handelt. Allerdings hat Dill natürlich Recht, dass zu kleine Flächen auch zu viel Detail mit sich bringen können. Sie würden die Umgebung nicht ausreichend abstrahieren und entsprechend wenig Vorteil bieten. Beim Arbeiten mit einer Rasterstruktur können zudem andere Probleme auftreten, die in Abschnitt 1.4 und 7.1.1 genauer erläutert werden.

4. Die Regionen sollen mehr oder weniger konvex und am besten rund/quadratisch/hexagonal sein. Diese Eigenschaft ist natürlich dann besonders wichtig, wenn man die Regionen für das Finden von Wegen nutzen möchte. Denn durch Konvexität ist sichergestellt, dass es innerhalb der Fläche keine zwei Punkte gibt, bei denen Teile einer direkten Verbindungslinie außerhalb der Fläche liegen würden. Dill nimmt es mit dieser letzten Eigenschaft wenig genau und bezeichnet sie sogar als überbewertet. So ist eine seiner Regionen solange ausreichend konvex, wie ihr Mittelpunkt sich noch im Inneren der Region befindet. Die Eigenschaft soll eher verhindern, dass Regionen zu konkav werden, wodurch sie unter Umständen zu viele Informationen zusammenfassen würden. Aus demselben Grund finde auch ich eine Unterteilung prinzipiell nicht verkehrt. Die Stelle dieser Unterteilung sollte meiner Meinung nach jedoch von der Spielflächenstruktur bestimmt und nicht von Eigenschaften festgelegt werden. Am Ende von Abschnitt 2.2.2 wurde bereits darauf hingewiesen, dass besonders verengte Stellen

> zur Unterteilung geeignet sind. Sind diese nicht gegeben, sehe ich keinen Grund, eine Region nur auf Grund ihrer mangelnden Konvexität zu unterteilen. Laut Dill sind lange schmale Rechtecke nützlich, um lange schmale Flächen zu repräsentieren. Genauso finde ich es sinnvoll, dass konkave Flächen wie gebogene Gänge von konkaven Regionen repräsentiert werden sollten.

Im Gegensatz zu Kohan II basieren die Karten von Iron Harvest nicht auf einer Rasterstruktur. Für das Finden von Rechtecken müsste man diese Struktur also extra anlegen. Falls dies zu einem Zeitproblem führen sollte oder die Karte nicht gut repräsentieren würde, führt Dill auch die Option an, eine alternative Grundstruktur wie beispielsweise ein NavMesh zu wählen.

Die anschließend beschriebene Variante zum Finden von Rechtecken, ist laut Dill zeitaufwendig und findet oft auch schmale und längliche oder ungünstig geformte Regionen. Dafür sei sie sehr einfach, gut zu debuggen und diene als geeignete Basis. Sie beginnt in der unteren linken Ecke der Karte und lässt ein Rechteck solange nach rechts und oben wachsen, bis es an ein Hindernis kommt (vgl. Dills Abbildung 18). Zu große Rechtecke werden zusätzlich geteilt und zu kleine Rechtecke schließen sich größeren Regionen an, solange oben genannte Eigenschaften eingehalten werden. In Abbildung 19 aus Dills Ansatz kann man die so gefundenen Regionen gut erkennen.

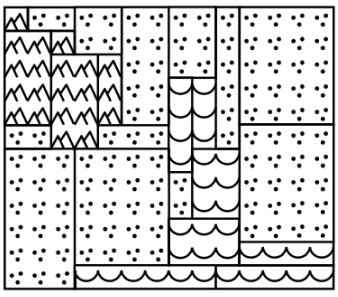


Abbildung 18: Dills Rechtecke der verschiedenen Geländetypen. (Bild aus [25, S. 369])

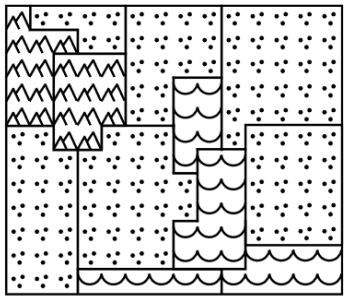


Abbildung 19: Dills aus Rechtecken zusammengefasste Regionen. (Bild aus [25, S. 369])

In Abschnitt 2.2.2 wurde bereits darauf hingewiesen, dass Dill die kürzesten Wege zwischen Regionen in Tabellen speichert und somit zeitliche Vorteile in der Wegfindung erzielt. Für Iron Harvest wäre es sicherlich vom Speicher her nicht machbar, solche Tabellen für alle Regionen anzulegen. Zum einen sind die Karten sehr groß, womit es zu viele Regionen gäbe und zum anderen müsste man diese Tabellen für jede Einheitengröße einzeln berechnen. Da sich die Karte durch Zerstörungen ständig verändert und evtl. durch beispielsweise Influence Maps neben der Länge eines Weges später noch weitere Kriterien wie die Sicherheit hinzukommen könnten, halte ich das Speichern im Fall von Iron Harvest für nicht sinnvoll.

Um eine der gefundenen Regionen nun als Engpass zu identifizieren, führt Dill eine Breitensuche von jeder an der Region R angrenzenden Region durch. Wenn die Suche es nicht schafft, andere angrenzende Regionen zu finden, ohne R selbst in die Suche mit einzubeziehen, sei R ein Engpass. Nimmt man als Beispiel die Region 1 aus Abbildung 20, würde man mit einer Breitensuche von der angrenzenden Region 2 genau zwei Wege zur Region 3 finden. Der eine Weg führt über Region 4 und der andere über Region 1. Über Region R kann nämlich immer ein Weg gefunden werden, da die Start- und alle Ziel-Regionen ja als angrenzend zu R definiert sind. Genau deshalb wird R nicht in die Breitensuche mit einbezogen. Übrig bleibt im Beispiel also nur der Weg über Region 4.

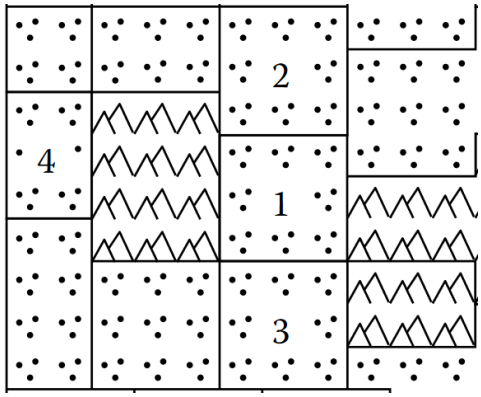


Abbildung 20: Die Region 1 könnte je nach Tiefe eine Engpass-Region zwischen den Regionen 2 und 3 sein. (Bild aus [25, S. 374])

Damit die Suche zum einen nicht so lange dauert und das Ergebnis zum anderen nicht ganz so streng ausfällt, empfiehlt Dill die Tiefe der Breitensuche entsprechend zu begrenzen. Bei einer festgelegten Maximaltiefe von 5 wäre Region 1 somit eine Engpass-Region. Alles darüber hinaus würde die Region 3 erreichen und die Bedingung damit nicht erfüllen.

Wie bereits in Abschnitt 2.3.3 beschrieben, muss ein Engpass allerdings nicht immer eine Region sein und für die meisten Vorteile sind eher die Übergänge der Engpass-Regionen interessant. Wenn man einen ähnlichen Ansatz aber auf die Übergänge anwendet, ist es trotzdem möglich auch diese als Engpass zu finden.

5.3 Auswertung

Perkins Ansatz bringt kaum Nachteile mit sich und würde sich sicherlich hervorragend für das Finden von Engpässen und Regionen eignen. Allerdings bietet die Architektur von Kevin Dill den ganz klaren Vorteil, dass sie die Umgebung nicht nur aussagekräftig abstrahiert, sondern neben dem Finden von Engpässen auch zahlreiche andere Anwendungsmöglichkeiten offen lässt. So könnte man neben den Regionen auch die kleineren Rechtecke für die Wegfindung in einer Hierarchie oder für Influence-Maps nutzen. Der Nachteil ist, dass die gefundenen Rechtecke durch ihre teilweise schmale und längliche Form die Aussagekraft der Flächen stark einschränken können. Würde man diesen Nachteil entfernen oder das Finden der Ausgangsflächen komplett abändern, hat der Ansatz allerdings außerordentlich großes Potential. Die Flexibilität dieser Architektur, die es beispielsweise auch ermöglichen würde, das spätere NavMesh als Basis zu nehmen, hat mich daher überzeugt. Falls es möglich wäre, außerdem Informationen über die Größe der Rechtecke oder Polygone zu erhalten, wäre es sogar denkbar, die Regionen am Ende ähnlich wie in Perkins Ansatz zu bestimmen, was beide Ansätze miteinander verbinden würde. In Kapitel 7 wird genauer darauf eingegangen, welche Abwandlungen erfolgt sind und wie der Algorithmus schrittweise vorgeht.

6 Ansätze zur Erstellung eines NavMeshs

Wie in der Einleitung bereits angedeutet, ist das Verwenden von NavMeshs mittlerweile einer der besten Varianten zum Planen von Wegen für KIs und soll im späteren Verlauf der Entwicklung auch für Iron Harvest implementiert werden. Damit dieses NavMesh auch für das Finden von Engpässen genutzt werden kann, müssen folgende beiden Anforderungen erfüllt werden. Diese haben sich aus der Umsetzung und der Analyse aus Abschnitt 7.1.1 ergeben.

1. Jede Kante muss an beiden Enden an ein Hindernis angrenzen. Wäre diese Anforderung nicht erfüllt, könnte nicht getestet werden, ob eine bestimmte Einheitenbreite von einem Polygon ins nächste gehen kann.

2. Die kürzesten Abstände zwischen Hindernissen müssen als Kante vorhanden sein. Letztendlich können nur Übergänge als Engpass erkannt werden, die im NavMesh auch vorhanden sind.

In diesem Kapitel werden nun verschiedene Ansätze zur Generierung eines NavMeshs im Hinblick auf die beiden genannten Anforderungen analysiert. Dieses muss zusätzlich für alle Polygone das Kriterium der Konvexität erfüllen. Dadurch wird bei der Wegfindung sichergestellt, dass man im Inneren jeden Punkt durch eine gerade Linie erreichen kann, ohne mit dieser ein Hindernis zu schneiden.

6.1 Triangulierung

Ein möglicher Ansatz zur Erstellung eines NavMeshs ist die Triangulierung bei der alle Punkte der Hindernisse zu Dreiecken miteinander verbunden werden. Häufig wird dabei die Delaunay-Triangulierung (DT) [42] verwendet. Diese findet weniger schmale oder längliche Dreiecke, da jede Kante, die vier Punkte in zwei Dreiecke unterteilt, gewechselt werden kann. Der Wechsel findet solange statt bis der umrandende Kreis K keinen weiteren Punkt mehr enthält. Aufgrund einer Dualität, entsteht durch ein Verbinden der Mittelpunkte dieser Kreise außerdem ein Voronoi-Diagram (vgl. Abbildung 21). Da dadurch unter Umständen aber keine Kanten an den Grenzen der Hindernisse entstehen, kann in der Constrained-Delaunay-Triangulierung (CDT) [43] das Wechseln von bestimmten Kanten verhindert werden.

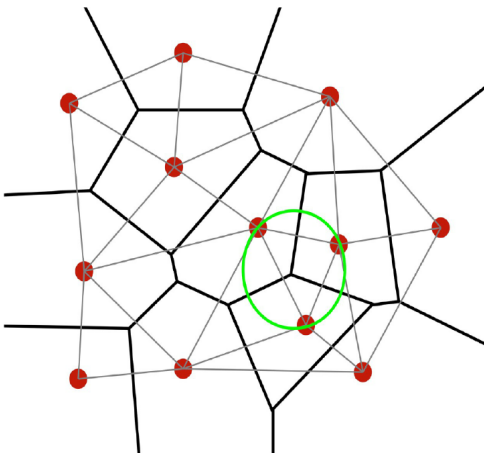


Abbildung 21: Gezeigt wird die Dualität zwischen Voronoi Diagrammen und der Delaunay Trinagulierung. Innerhalb des grünen Kreises befindet sich kein anderer roter Punkt.

Durch die Triangulierung findet eine ziemlich detailreiche Unterteilung der begehbaren Fläche in Dreiecke statt. Da Dreiecke immer konvex sind, müssen in dieser Variante keine inneren Winkel überprüft werden. Durch die hohe Anzahl an Kanten könnte man fälschlicherweise annehmen, dass die für das Finden der Engpässe wichtigen Verbindungen automatisch mit vorhanden wären. Aber genau diese fehlt in Abbildung 23 Die zweite Anforderung wird also nicht erfüllt.

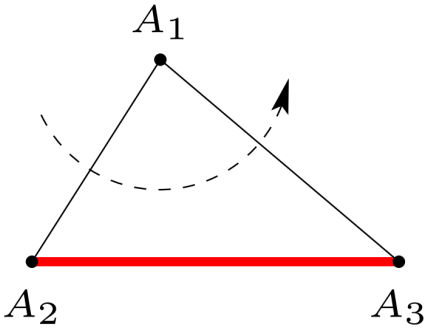


Abbildung 23: Vom Punkt A1 zur Kante A2 bis A3 gäbe es noch einen kürzeren Abstand [44, S. 12].

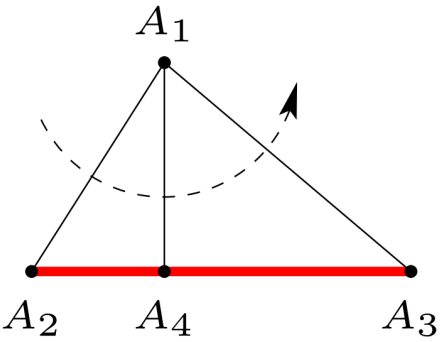


Abbildung 22: Durch das Verfeinerungsverfahren wird die fehlende Kante hinzugefügt [44, S. 12].

Durch ein Verfeinerungsverfahren in [44] haben Lens und Boigelot allerdings eine Variante gefunden, um diese kürzesten Abstände als Kanten nachträglich hinzuzufügen. Dafür werden Steiner-Punkte bzw. Orthogonalprojektionen hinzugefügt (vgl. Abbildung 22). Da der Ansatz inkrementell ist, aktualisiert sich die Triangulierung automatisch und bietet damit einen großen Vorteil für eine mögliche Verwendung zur Laufzeit eines Spiels. Beim Zerstören von Hindernissen könnten Punkte einfach entfernt und neue hinzugefügt werden, ohne dass eine erneute Berechnung des kompletten Spielfeldes notwendig wird. In Abbildung 24 ist das Ergebnis der Verfeinerung mit den zusätzlichen Kanten zu sehen. Damit wäre dies eine Variante, die beide Anforderungen für das spätere Finden der Engpässe erfüllt.

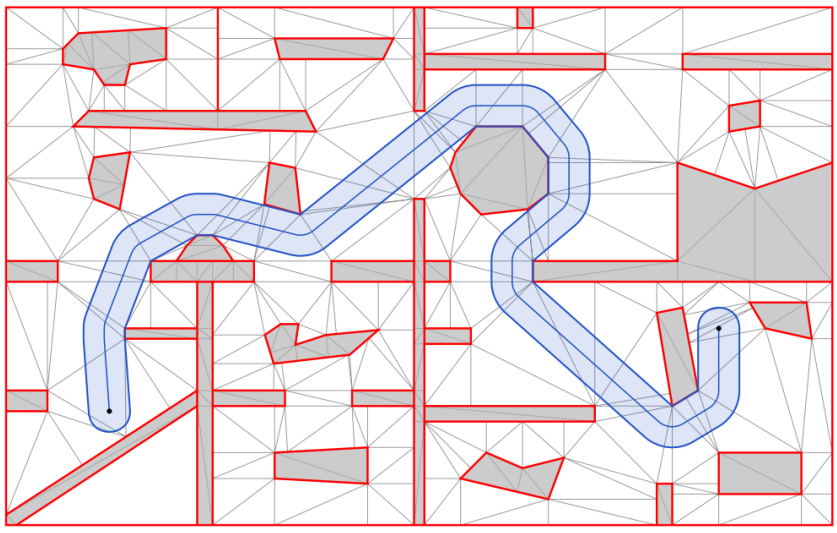


Abbildung 24: Ergebnis nach dem Verfeinerungsschritt [44, S. 12]

Ein weiterer Ansatz ist der Hertel-Mehlhorn-Algorithmus [45], welcher in Abbildung 25 zu sehen ist. Dieser löscht nach der Triangulierung nicht notwendige Diagonalen. Leider werden auch hier nicht die kürzesten Abstände gefunden.

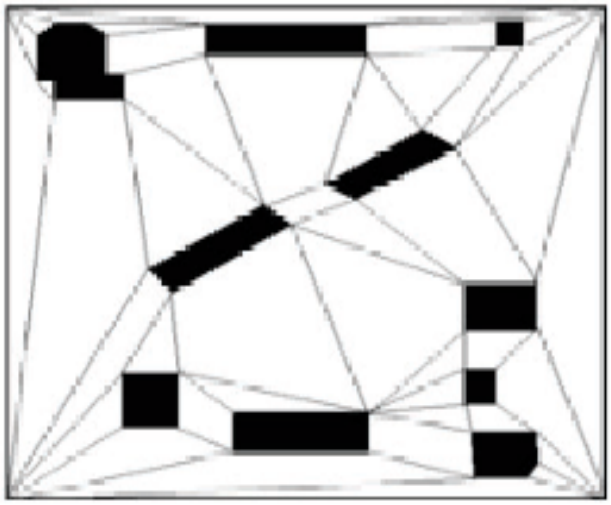


Abbildung 25: Ergebnis Hertel-Mehlhorn
(stark vergrößerter Ausschnitt aus [17, S. 177])

6.2 Space Filling Volumes

Hale et al. beschreibt in [17] einen erweiterten Space Filling Volumes Algorithmus (DEACCON), bei dem die ursprünglichen Rechtecke gegen komplexere schräge Hindernisse wachsen können und somit eine bessere Abdeckung der Fläche ermöglichen. Der Vorteil gegenüber dem Hertel-Mehlhorn NavMesh sei eine bessere algorithmische Kontrolle, das Verhindern der Polygon-Ansammlungen in den Ecken und eine Verbesserung der Navigationsgeschwindigkeit von Agenten. Auch wenn der Ansatz besser mit komplexen Hindernissen umgehen kann, erfüllt das Ergebnis aus Abbildung leider keine der benötigten Anforderungen

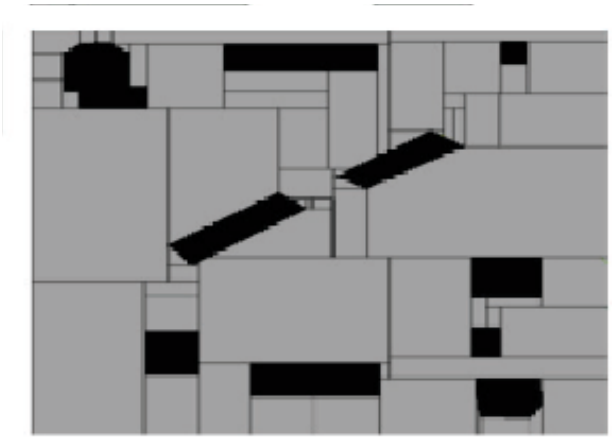


Abbildung 26: Zu sehen ist das Ergebnis des DEACCON Algorithmus (Bildausschnitt stark vergrößert aus [17, S. 177]).

6.3 Automatic Navigation Mesh Generator (ANavMG)

Eine weitere Möglichkeit ein NavMesh zu erstellen wurde in [46] präsentiert. Der gesamte Abschnitt bezieht sich auf diese Quelle. Ziel des Algorithmus ist es so wenig Polygone wie möglich zu erzeugen, die Kanten dazwischen so kurz wie möglich zu halten und zu kleine Winkel in den Polygonen zu vermeiden. Zweiteres ist im Hinblick auf das Finden von Engpässen besonders interessant. Sieht man sich die rot markierten Verbindungen im Ergebnis aus Abbildung 27 aber genauer an, erkennt man bereits, dass einige Kanten eher gewählt wurden, weil sie das Einteilen in zwei konvexe Polygone ermöglichen. Das Finden der kürzesten Abstände funktioniert an vielen Stellen zwar beeindruckend gut, aber Konvexität hat scheinbar noch die höhere Priorität.

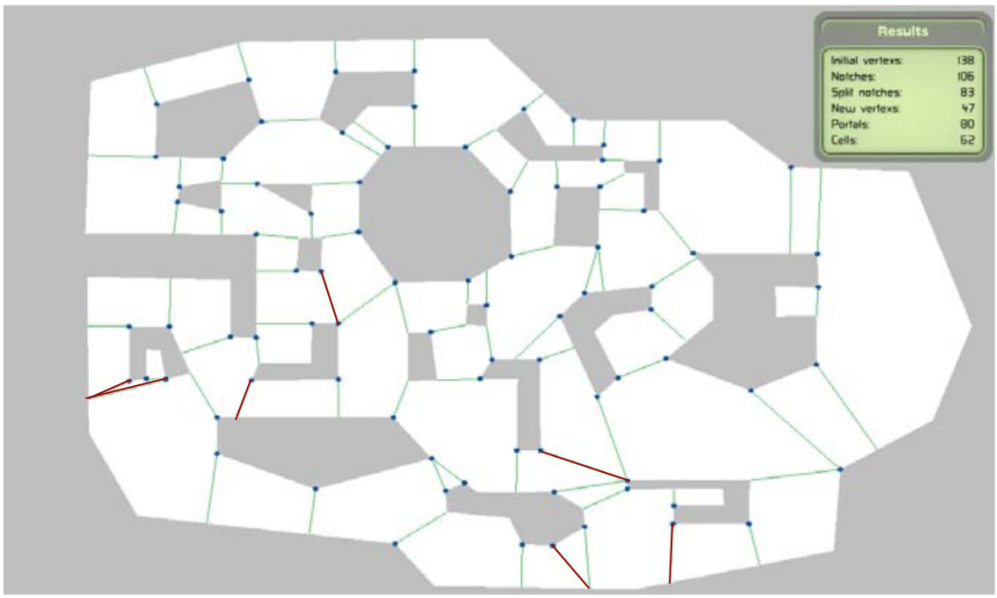


Abbildung 27: Die rot markierten Übergänge entsprechen nicht den kürzesten Abständen zwischen den Hindernissen (Bild ohne die roten Markierungen aus [46, S.11]).

Der ANavMG nimmt als Input ein 2D Polygon mit entgegen dem Uhrzeigersinn angeordneten Punkten mit oder ohne Löcher, welches die begehbare Fläche darstellt. Die Löcher stellen dabei die Hindernisse dar und müssen entsprechend aus im Uhrzeigersinn angeordneten Punkten bestehen. Die Schnittkanten, welche später gleichzeitig die Verbindungen oder Portale zwischen den konvexen Polygonen sein werden, verlaufen sowohl zwischen bereits vorhandenen als auch gegebenenfalls neu hinzuzufügenden Punkten.

Als erstes wird das Polygon nach sogenannten „Notches“ durchsucht. Dies sind Punkte mit einem inneren Winkel größer 180°. Gesucht wird dann nach der kürzesten Schnittkante, die den Innenwinkel gleichzeitig in je zwei Winkel kleiner 180° unterteilt (nachfolgend Konvexitäts-Bedingung genannt).

Damit eine alleinige Kante für das Unterteilen in konvexe Polygone ausreichend ist, wird für jeden „Notch“ wie in Abbildung 28 ein Winkelbereich festgelegt. Jede Kante im Inneren dieses Bereiches würde demnach die Konvexitäts-Bedingung erfüllen.

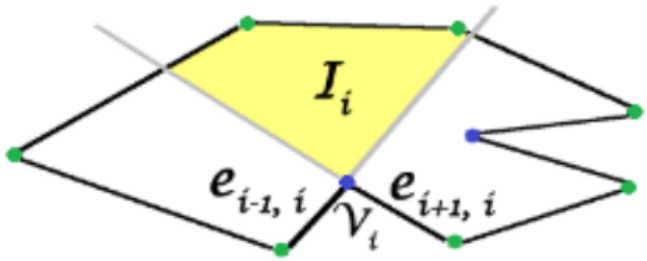


Abbildung 28: Der Grenzbereich I_i wird für den „Notch“ V_i festgelegt [46, S.04].

Dabei wird zwischen drei verschiedenen Fällen unterschieden.

1. Die kürzeste Verbindung ist ein anderer Punkt. In diesem Fall müsste also lediglich eine Kante zwischen diesen beiden Punkten gebildet werden. Falls es sich beim anderen Punkt ebenfalls um einen „Notch“ handelt, könnte es sein, dass auch für ihn durch die neue Kante die Konvexitäts-Bedingung bereits erfüllt ist. Dementsprechend müsste für ihn keine weitere Kante gefunden werden. Für die zweite Anforderung ist dies leider nicht hinreichend, da für den zweiten „Notch“ auch noch kürzere Kanten wichtig sein könnten, die entsprechend noch gesucht werden sollten.

2. Die kürzeste Verbindung ist die Projektion auf eine Kante. Falls sich der Projektionspunkt dabei nicht direkt auf der Kante befindet, wird stattdessen das nähere Ende der Kante genommen. Ist der Projektionspunkt wie in Abbildung 29 außerhalb des Winkelbereiches, wird stattdessen der Schnittpunkt der äußeren Begrenzung des Winkelbereichs und der Kante genommen. Diese äußere Begrenzung bildet sich aus den Verlängerungen der an dem Notch angrenzenden Kanten. Für die zweite Anforderung wäre es jedoch wichtig, die Kante zwischen dem eigentlich kürzesten Abstand zu finden, der außerhalb des Winkelbereichs liegt. Würde man solche Kanten ebenfalls hinzufügen, hätte der Ansatz also großes Potential. Dafür könnte man den Winkelbereich beispielsweise einfach viel weiter ausdehnen.

3. Die kürzeste Verbindung ist ein anderes Portal. Da es nicht sinnvoll wäre, Kanten zu anderen Portalen zu finden, würde ich Portale gar nicht in die Überprüfung der kürzesten Abstände mit einbeziehen. Dementsprechend gehe ich auf diesen dritten Fall

nicht näher ein.

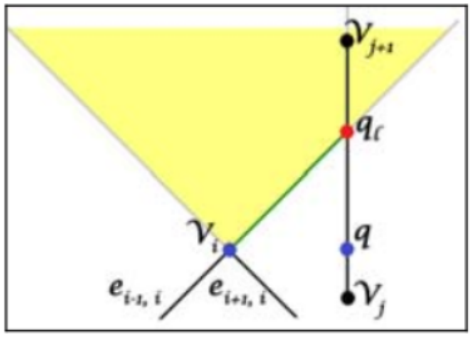


Abbildung 29: Hier ist der Winkelbereich nicht groß genug, um den Projektionspunkt q mit einzubeziehen. Eine Kante zu diesem Punkt würde den kürzesten Abstand allerdings gut repräsentieren [46, S.06].

Zum Schluss könnten überflüssige Kanten wieder entfernt werden. Auch böte es sich an, bei der Konvexitäts-Bedingung einen Toleranzbereich von 5° hinzuzufügen.

6.4 Auswertung

Obwohl der Ansatz von Lens und Boigelot hier als einziger die Anforderungen erfüllt, findet dieser zusätzlich zahlreiche Kanten, die für das Finden von Engpässen nicht wichtig wären. Da dieser Teil des Algorithmus später wahrscheinlich von dem neuen NavMesh für Iron Harvest ersetzt wird, ist für die Auswahl zum Testen vor allem eine schnelle und einfache Implementierbarkeit notwendig. Sind die Tests erst erfolgreich, kann grundlegend jeder Ansatz, der die Anforderungen erfüllt die vereinfachte Testvariante ersetzen. Daher habe ich mich hier für den ANavMG entschieden, der sich bereits mit dem Finden der kürzesten Abstände beschäftigt. In Abschnitt 7.1.2 wird versucht, die roten Kanten in Abbildung 27 für die benötigten Anforderungen zu korrigieren.

7 Umsetzung von Dills Architektur

Die Anforderungen für die Erstellung eines NavMeshs, auf die in Kapitel 6 bereits eingegangen wurde, ergaben sich größtenteils erst aus den Erkenntnissen der Implementierung aus Abschnitt 7.1.1. Direkt im Anschluss wird daher ein weiterer Ansatz präsentiert, der auf diese Anforderungen eingeht.

7.1 Finden von Rechtecken/Polygonen

Wie in Abschnitt 5.2 bereits beschrieben, bringt der Ansatz von Kevin Dill oft längliche und schmale Rechtecke hervor. Dies passiert besonders dann, wenn die äußeren Kanten der Hindernisse nicht entlang der X- oder Y-Achse orientiert sind. Bei komplexen Hindernissen mit Schrägen entsteht durch das Einteilen der Karte in ein Raster ein Treppeneffekt, wie man ihn aus Pixelbildern kennt. Stufe für Stufe stößt das Rechteck wie in Abbildung 30 beim Wachsen also an einer Seite an das Hindernis und kann zur anderen Seite ins Längliche weiterwachsen. Solche Flächen sind leider nicht aussagekräftig, weil ihre Form die eigentlich nicht schmale oder längliche Fläche nicht gut repräsentiert/abstrahiert. Das Unterteilen in kleinere Rechtecke führt außerdem zu unnötig vielen Kontakten zu umliegenden Rechtecken an den Außenkanten. So kann ein Rechteck mehr als 4 Nachbarn haben, obwohl es nur 4 Außenkanten besitzt, weil es allein an einer Kante an mehrere Rechtecke angrenzen kann. Da es in Iron Harvest Einheiten mit sehr unterschiedlichen Breiten gibt, eignet sich eine solche Struktur der Rechtecke nicht für die Wegfindung. Man könnte nicht absehen, ob eine große Mecheinheit durch einen Übergang passen würde. Dafür müsste man die komplette offene Kante kennen.

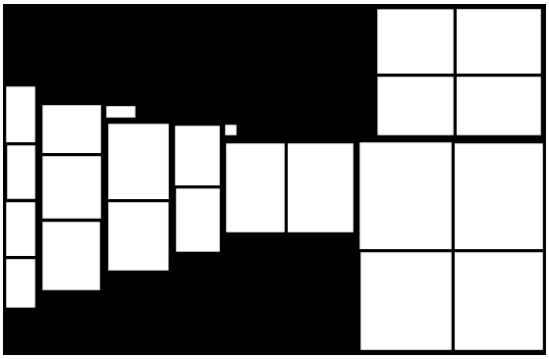


Abbildung 30: Durch den Treppeneffekt stoßen die Rechtecke beim Wachsen rechts Stufe für Stufe an ein Hindernis und können daher nur nach oben ins Längliche wachsen.

Außerdem kann es ebenfalls passieren, dass genau in der Mitte eines Engpasses unterteilt wird und dieser dadurch gar nicht erst gefunden werden kann. In Abbildung 31 wird beispielsweise kein Engpass zwischen den Regionen 2 und 5 gefunden. Selbst wenn man Region 4 als zu klein bewerten würde, könnte sie sich mit Region 3 verbinden, wodurch das Problem weiterhin bestehen würde. Das Zusammensetzen der Regionen wirkt dadurch oft willkürlich und bietet kaum Aussagekraft. Da der Algorithmus immer von der unteren linken Ecke beginnt und sich so schrittweise vorarbeitet, wird außerdem der Platz für potentiell größere Rechtecke verbaut. So könnte die Region 2 in Abbildung 31 auf der linken Seite etwas größer sein, wie es in Abbildung 32 der Fall ist. In der gefundenen Größe ist sie mit 5 Regionen verbunden, obwohl der Bereich eigentlich nur drei Zugänge hätte. Aus diesen Gründen und weil die Spielfelder in Iron Harvest durchaus komplexe Hindernisse mit Schrägen und Rundungen beinhalten, habe ich eine

Abwandlung zum Finden von Rechtecken bzw. Polygonen implementiert. Dazu muss noch gesagt werden, dass Rechtecke generell nicht geeignet sind, um solche komplexen Karten gut abzubilden. Dill beschreibt in [25, S.268] selbst, dass man damit vor allem schnell und einfach Ergebnisse produzieren kann, diese dann aber nicht gerade perfekt wären. Durch die Flexibilität der Architektur ließen sich Regionen aber schon gut testen bevor zum Finden der Polygone auch ein weiterer Ansatz ausprobiert wurde, der in Abschnitt 7.1.2 näher erläutert wird.

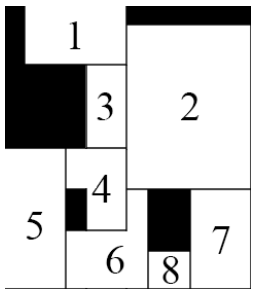


Abbildung 31: Gefundene Regionen aus eigener Implementierung von Dills Ansatz

7.1.1 Eigene Abwandlung von Dills Ansatz

Die in diesem Abschnitt beschriebene Abwandlung soll vor allem die größtmöglichen Rechtecke/Polygone finden. Dafür werden zunächst die Positionen gefunden, an denen die Fläche am größten ist. Dort platzierte Rechtecke dehnen sich komplett aus und dürfen sich an den Ecken auch gegenseitig überlappen. Dies ist in Abbildung 32 gut zu erkennen. Anschließend werden offene, in den Raum stehende Ecken abgeschnitten und aus dem Rechteck wird ein Polygon mit mindesten fünf Punkten. Diese Schnittkante ist gleichzeitig ein guter Übergang zwischen den Polygonen. Die Überlappung wäre damit nicht mehr vorhanden. Statt nun zweimal die jeweiligen Ecken abzuschneiden, kann man auch einfach eine der Ecken nach dem Abschneiden wieder in den Raum hineinwachsen lassen. Da auch dort eine mögliche offene Ecke erneut abgeschnitten werden würde, schmiegen sich die Polygone Stück für Stück den Hindernissen entlang durch die Karte. In Abbildung 32 kann man außerdem noch erkennen, dass neben den großen blauen Rechtecken auch kleine weiße Rechtecke übrig bleiben. Diese scheinen zunächst ein konsequentes Abfallprodukt zu sein, bilden beim genaueren Hinsehen allerdings sehr gute Engpass-Regionen. Sucht man also nach größeren Regionen, bilden sich automatisch auch kleine Regionen, da diese durch das Ausdehnen der anderen weniger Platz übrig haben.

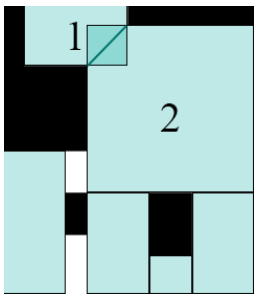


Abbildung 32: Zwischen den Rechtecken 1 und 2 entsteht durch Überlappung der Ecken eine rechteckige Schnittfläche.

In Abbildung 33 kann man erkennen, dass sich diese Einteilung in Polygone nicht für die Wegfindung mit verschiedenen Einheitenbreiten eignen würde. Ginge es nur nach der Kantenlänge, würde der Mech M durch beide Übergänge der Engpass-Region hindurch passen. Da dieser von der Höhe aber zu klein ist, kommt er trotzdem nicht in Region 2. Aus diesem Grund müsste man auch die Größe der Regionen in die Wegfindung mit einbeziehen.

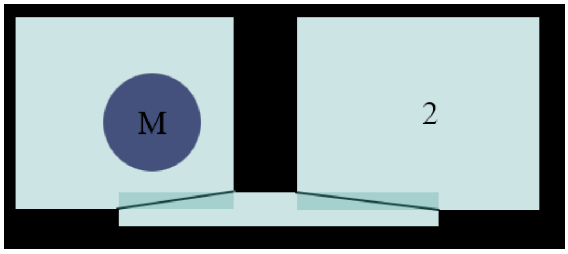


Abbildung 33: Der Mech M könnte die Übergänge der Regionen passieren, um in Region 2 zu kommen. Die Höhe der Engpass-Region wäre aber zu schmal.

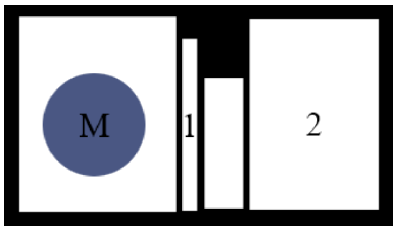


Abbildung 34: Der Mech M würde problemlos durch die Region 1 nach 2 kommen, obwohl die kleinere Seitenlänge von 1 sehr schmal ist.

Dies ist allerdings nicht allein mit der kleinsten Seitenlänge des Rechtecks getan. In Abbildung 34 kann man sehen, dass der Mech M problemlos in Region 2 kommen würde, aber die kleinste Seitenlänge von Region 1 wäre kleiner als die Breite des Mechs. Hier wäre also die größere der beiden Seitenlängen entscheidend. Aus diesem Grund schließe ich die weißen Lücken aus Abbildung 31 durch das Herauswachsen eines neuen Rechtecks aus den offenen Kanten der größeren bereits gefundenen Polygone. Mit „offen“ sind Abschnitte gemeint, die nicht direkt an einem Hindernis liegen, aber trotzdem an andere Polygone angrenzen können. Die Größe der neuen so gefundenen Polygone ist dann das Minimum aus der Größe des alten Polygons und der Länge der Ursprungskante. Kleinere Rechtecke entwachsen also aus den Größeren und letztere müssen folglich zuerst gefunden werden.

Die Implementierung startet aus der oberen linken Ecke, da dort bei mir die (0,0) Position liegt und sucht in der gesamten Karte systematisch nach Quadraten. Sprich ein Quadrat wächst immer nach rechts und unten bis es an ein Hindernis stößt. Bis hierhin gibt es also kaum einen Unterschied zu Dills Variante. Anschließend wird die Kantenlänge des Quadrates in einem zweidimensionalen Array gespeichert. In Abbildung 35 kann man dazu ein Beispiel sehen.

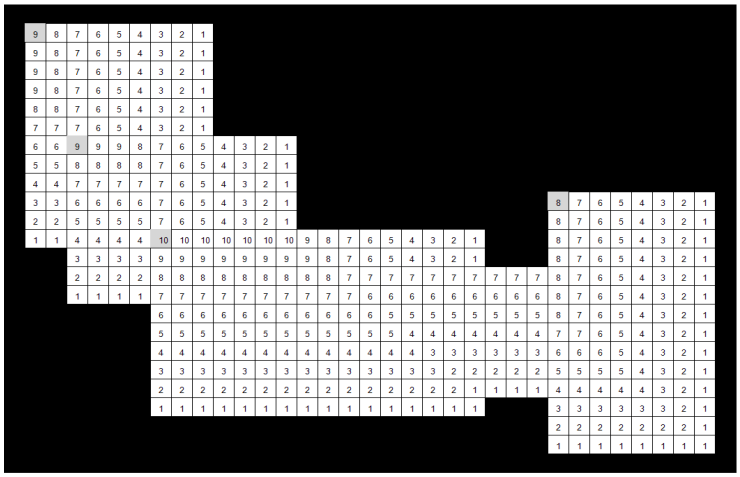


Abbildung 35: Die Seitenlängen der gefundenen Quadrate an den jeweiligen Positionen werden im Array gespeichert.

Dort lässt sich auch gut erkennen, dass der diagonale Wert in der Zeile über und links von der aktuellen Startposition bereits etwas über die Größe des aktuell zu findenden Quadrates aussagt. Dieses muss nämlich mindestens so groß wie dieser Wert weniger eins sein. Fängt man bei dieser Größe erst mit dem Wachsen an, arbeitet der Algorithmus deutlich schneller und muss nicht ständig Flächen überprüfen, die er bereits mehrmals überprüft hat.

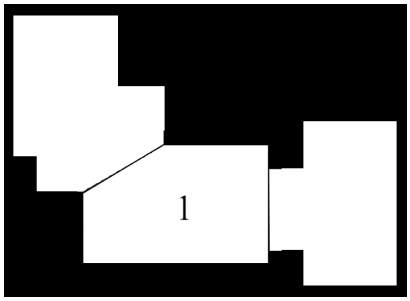


Abbildung 36: Ausgedehntes ursprüngliches Quadrat mit abgeschnittener Ecke

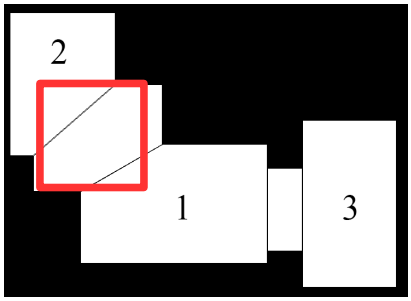


Abbildung 37: Der Platz für das rot umrandete Quadrat ist nicht mehr frei.

Ist das fertige Quadrat nach dem Wachsen größer als die umliegenden Vergleichswerte, wird dessen Startposition zusätzlich in einer Priority-Queue gespeichert, bei der die Priorität von der Kantenlänge des Quadrates bestimmt wird. Da links oben gestartet wird und es entsprechend rechts und unterhalb noch keine Werte gibt, ist mit den umliegenden Werten der Wert links und darüber gemeint. Jede Priorität kommt nur einmal vor und speichert eine Liste aus den Startpositionen aller Quadrate dieser Größe. Die größten gefundenen Quadrate stehen dann an erster Stelle. Im Beispiel der Abbildung 35 werden also vier Quadrate gefunden, deren Startpunkte grau hervorgehoben sind. Die Priority-Queue enthält eine Position mit der Priorität 10, dann zwei Positionen mit 9 und zuletzt eine Position für die Priorität 8.

An der ersten Position der Priority-Queue mit der Priorität 10 wird das anfängliche Quadrat und spätere Polygon mit der Id 1 platziert. In Abbildung 36 kann man sehen, wie es nach dem Ausdehnen und mit abgeschnittener Ecke schließlich aussieht. Da das Abschneiden einer Ecke durch eine Gerade erfolgt, die das Polygon an zwei Stellen schneidet, kann das Polygon nicht konkav werden. Denn da die Kante von der Geraden geschnitten wird, entstehen 4 Winkel, von denen gegenüberliegende gleich groß sind und die insgesamt 360° ergeben. Entsprechend müssen beide Winkel kleiner 180° sein, was der Bedingung für Konvexität aus Abschnitt 2.3 entspricht. In einem Id-Array wird für jede Position innerhalb des Polygons zusätzlich die Id 1 gespeichert. Dies ermöglicht eine O(1) Zugriffszeit von einer Position zum dazugehörigen Polygon wie sie beim BWTA2 aus [21, S.58f] durch den in [47] beschriebenen Algorithmus auch ermöglicht wurde.

Die nächste Priorität ist die 9 und enthält zwei Positionen für mögliche Quadrate. Bevor diese gesetzt werden, wird zunächst für alle bereits vorhandenen Polygone überprüft, ob aus dessen offenen Ecken oder Kanten neue Rechtecke entwachsen könnten. Das bereits gefundene Polygon 1 hat eine offene Kante mit der Länge 7 und eine offene Ecke mit Katheten der Länge 7 und 4. Für die aktuelle Priorität 9 ist also kein Entwachsen möglich. Anschließend wird versucht, die beiden Quadrate aus der Liste in der Priority-Queue zu platzieren. Dies ist beim ersten mit der Position |1||1| kein Problem. Es dehnt sich komplett aus und schneidet die offene, untere, rechte Ecke ab. Das Quadrat mit der roten Umrandung in Abbildung 37 kann anschließend aber nicht platziert werden, da die entsprechenden Positionen im Id-Array bereits belegt sind und das Quadrat sich daher nicht ausbreiten kann.

Die größte offene Kante ist nach wie vor die 7 und somit gibt es auch vor dem Platzieren des

letzten Quadrates der Priorität 8 keine „Entwachsungen“. Das Quadrat 3 wird gesetzt und dehnt sich wie gewohnt aus. Anschließend entwachsen Polygone an den verbliebenen offenen Kanten in Reihenfolge ihrer Größe. Als erstes wächst wie in Abbildung 38 die Ecke von Polygon 1 nach oben links und überlappt dabei Polygon 2.

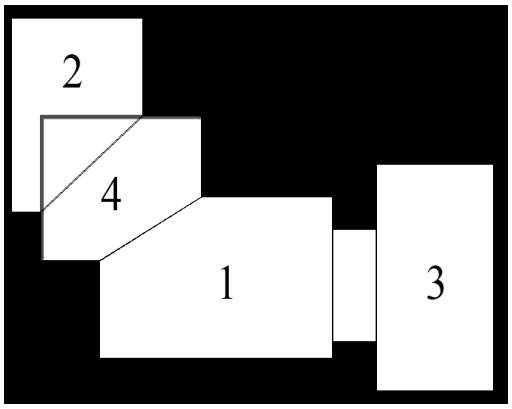


Abbildung 38: Beim Ausdehnen kann es zu Überlappungen kommen, die nach dem Abschneiden der Ecken wieder entfallen.

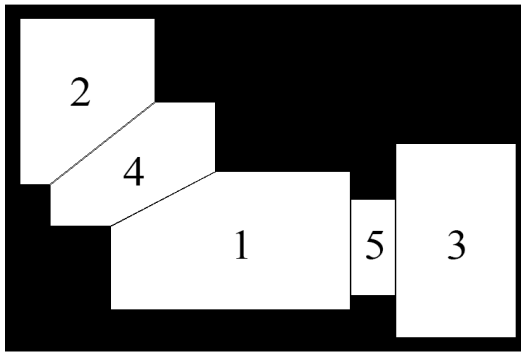


Abbildung 39: Fertige Ausdehnung der Polygone

Damit dieses Überlappen auch klappen kann, ist es Kanten erlaubt auch dann weiter zu wachsen, wenn an entsprechenden Stellen bereits Werte im Id-Array gesetzt worden sind. Einzige Bedingung dabei ist, dass mindestens ein Feld entlang der Kante noch unbelegt sein muss. Nach dem Abschneiden der Ecke ist die Überlappung nicht mehr vorhanden und die letzte Kante an Polygon 1 kann ebenfalls nach rechts wachsen. Damit ist in Abbildung 39 schließlich die komplette Fläche abgedeckt.

In den Abbildungen 40 und 41 können die Rechtecke von Dills Ansatz abschließend nochmal mit den gefundenen Polygonen der Abwandlung verglichen werden.

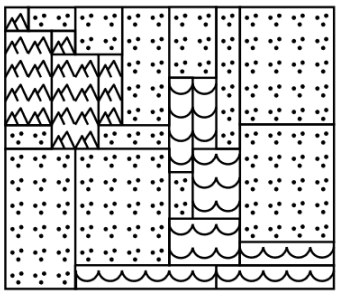


Abbildung 40: Dills Rechtecke der verschiedenen Geländetypen (Bild aus [25, S. 369]).

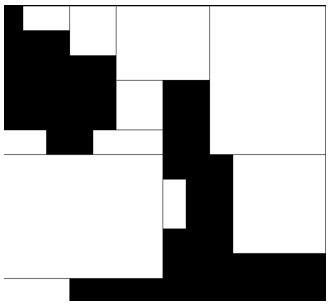


Abbildung 41: Gefundene Polygone aus der Abwandlung von Dills Ansatz (Spielfeld aus [25, S. 369] nachgebaut).

Durch den bereits erwähnten Treppeneffekt, kann es leider auch bei meiner Abwandlung passieren, dass sich Kante für Kante wie in Abbildung 42 immer nur sehr schmale Polygone ergeben. Diese liegen teilweise so dicht beieinander, dass sie sich kaum noch voneinander abgrenzen und nahezu eine Fläche bilden.

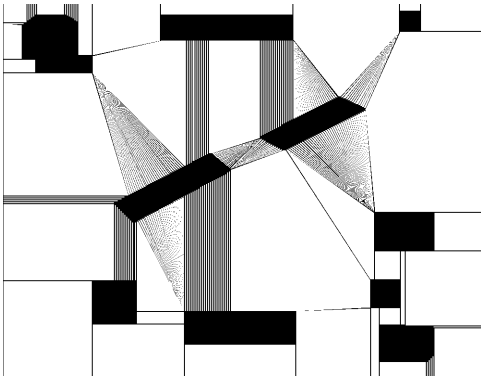


Abbildung 42: Durch den Treppeneffekt bilden sich beim Ausdehnen viele sehr schmale Polygone. (Spielfeld aus [17, S. 177] nachgebaut)

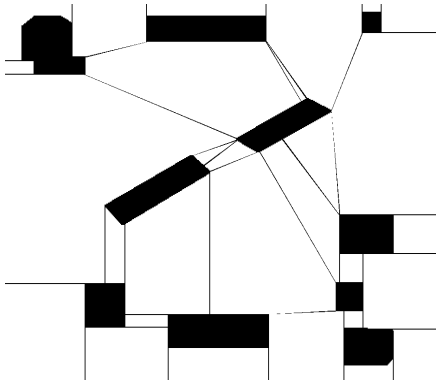


Abbildung 43: Durch den Toleranzwert werden die Polygone wieder größer, verlieren aber leider ihr Merkmal der Konvexität. (Spielfeld aus [17, S. 177] nachgebaut)

Aus diesem Grund kann dem Algorithmus ein Toleranzwert übergeben werden. Dadurch kann der offene Teil der Kante beim Ausdehnen auch dann weiter wachsen, wenn er größer oder gleich der eigentlichen Kantenlänge weniger der Toleranz ist. Damit die wichtigen Engpässe später aber trotzdem gefunden werden können, muss Folgendes noch beachtet werden. Hat die Kante beim Ausdehnen mit Toleranz die maximale Engpassbreite erreicht oder unterschritten, muss sie an dieser Stelle abbrechen. Diese hat also eine höhere Priorität als die Toleranz.

Die mit Toleranzwert gefundenen Polygone in Abbildung 43 sind leider nicht länger konvex. Falls die Polygone gleichzeitig als NavMesh dienen sollen, muss die Konvexität jedoch gewährleistet werden und somit darf der Toleranzwert in diesem Fall nicht genutzt werden.

Verbindungen zwischen Polygonen können ermittelt werden, indem man im entsprechenden Array lediglich überprüft, welche Ids sich in benachbarten Positionen der offenen Kanten befinden. Das ist einer der Vorteile an einer Raster-basierten Variante. Ein weiterer Vorteil ist, dass die Kollision mit anderen Objekten einfacher überprüft werden kann. Statt beispielsweise eine Linie auf Kollision mit jedem in der Nähe liegenden Objekt zu überprüfen, kann man lediglich die Kästchen im Raster entlang der Linie nach Hindernissen durchsuchen. Ein Nachteil der Rasterstruktur ist jedoch der bereits angesprochene Treppeneffekt, wodurch sehr kleine bzw. im Bezug auf die Größe der Rasterung auch zu kleine Polygone entstehen können. Eine schräge Linie, die weniger oder stärker als 45° geneigt ist, lässt sich im Raster nicht richtig darstellen und dementsprechend nur annähern. Liegen also mehrere davon sehr eng beieinander, wird dies vor allem beim späteren Erkennen der Verbindungen und wichtigen Engpässe zu Fehlern führen.

Aus diesem Grund habe ich mit dem Gedanken gespielt, den selben Ansatz ohne das Raster zu implementieren. Beispielsweise könnte eine Verbindung mit dem Ansatz aus [17], bei dem die Rechtecke gegen schräge Hindernisse wachsen, vielversprechend sein. Der Treppeneffekt wäre dann kein Problem mehr und der Toleranzwert könnte weggelassen werden. Dadurch wären die Polygone wieder konvex.

Leider weist der komplette Ansatz jedoch einige Schwächen auf, die ich anfangs übersehen hatte. Das Quadrat als Grundlage ist zwar ein großer Indikator dafür, welche Einheitengröße in das Polygon hinein passt, in Abbildung 44 kann man aber trotzdem erkennen, dass diese anfängliche Annahme nicht immer richtig ist. Der weiße Kreis innerhalb des größten grünen Quadrates könnte noch leicht weiter nach außen auf die Größe des blauen Kreises wachsen, bevor er die Hindernisse berühren würde. Dies ist immer dann der Fall, wenn ein zu großer Teil des ursprünglichen Quadrates nicht direkt an Hindernisse angrenzt und es somit offene Ecken gibt.

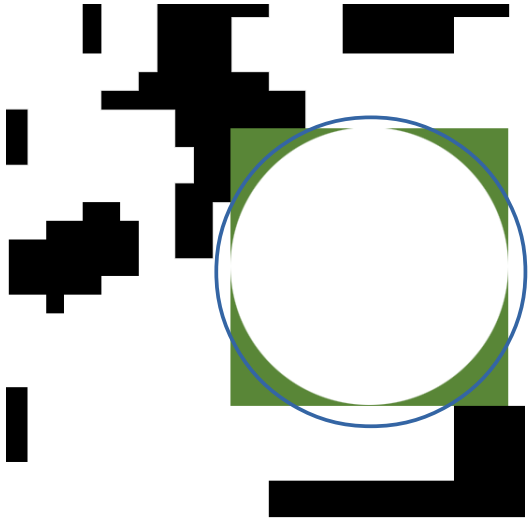


Abbildung 44: Der weiße Kreis im Inneren des grünen, größten Quadrates könnte eigentlich auf die Größe des blauen Kreises anwachsen, bevor er an die schwarzen Hindernisse stoßen würde.

Ein großes Problem zeigt sich in Abbildung 45. Dort würden die rot eingezeichneten Engpässe leider fehlen, obwohl sich dort die Zugänge der Engpass-Region befinden. Wären diese kürzesten Abstände zwischen den Hindernissen aber vorhanden, müsste man für die Wegfindung auch die Größe der Polygone nicht länger mit einbeziehen und das Problem aus Abbildung 32 auf Seite 27 wäre somit gelöst. Lediglich die Breite der Übergänge wäre entscheidend.

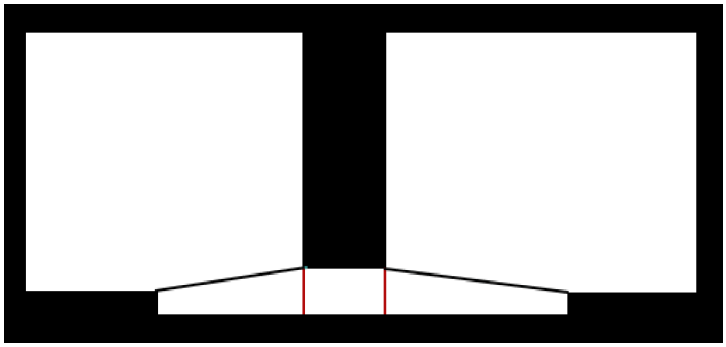


Abbildung 45: Die roten Übergänge werden durch den Ansatz nicht gefunden und können somit später auch nicht als Engpass erkannt werden.

Aus diesem Zusammenhang ergibt sich eine interessante Erkenntnis. Nicht nur eignet sich ein NavMesh für das Finden von Engpässen, sondern andersherum bilden durch Engpässe (bzw. kürzeste Abstände zwischen Hindernissen) getrennte, konvexe Polygone auch ein geeignetes NavMesh. Dieses wäre sogar in der Lage, die Breiten der Einheiten mit einzubeziehen. Daher beschäftigt sich die nächste Abwandlung mit dem Finden der kürzesten Abstände zwischen den Hindernissen. Aus dieser Erkenntnis ergaben sich die Anforderungen ein ein NavMesh für Kapitel 6.

7.1.2 Eigene Abwandlung des ANavMG

Der Algorithmus in [46], auf den sich dieser gesamte Abschnitt bezieht, hatte sich bereits zum Ziel gesetzt, möglichst kurze Verbindungen zu finden, dies aber auf einen für die Konvexität relevanten Winkelbereich begrenzt. Die Priorität lag stärker auf dem Einteilen in möglichst wenige konvexe Flächen. Weitet man diesen Winkelbereich jedoch weiter aus (siehe Abbildung 46), können auch noch kleinere Verbindungskanten gefunden werden. Diese garantieren dann allerdings keine konvexe Unterteilung mehr und somit werden mehrere Verbindungen pro „Notch“ notwendig, die insgesamt natürlich deutlich mehr Polygone erzeugen.

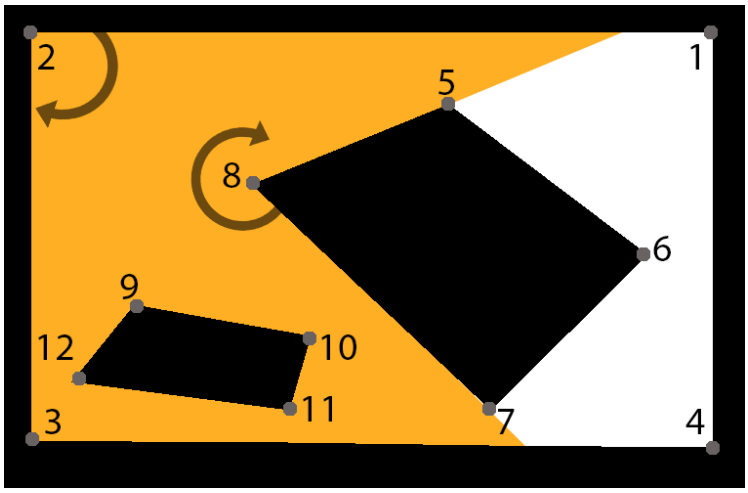


Abbildung 46: Der gelbe Winkelbereich ist deutlich größer gefasst und hat somit eine Chance, die kürzesten Abstände zu finden.

Diese Abwandlung des ANavMG arbeitet ebenfalls mit einem äußeren Polygon, welches die Gesamtfläche der Karte repräsentiert, und gegebenenfalls kleineren, inneren Polygonen. Diese entsprechen den Außengrenzen von Hindernissen auf dem Spielfeld und sind damit Löcher in der begehbaren Polygonfläche. Nur die Punkte des äußeren Polygons sind gegen den Uhrzeigersinn angeordnet. Alle anderen verlaufen mit dem Uhrzeigersinn. Dadurch wird sichergestellt, dass es sich beim Messen des Winkels zwischen der Kante vor und der Kante nach dem jeweiligen Punkt immer um einen Winkel innerhalb der begehbaren Fläche handelt (vgl. Abbildung 46). Denn dieser kann wie an den Pfeilen verdeutlicht, dann immer den ersten Winkel entlang des Uhrzeigersinns nehmen. Ist dieser Winkel größer 180°, handelt es sich um einen „Notch“. Innerhalb des nun größeren Winkelbereichs wird für diesen nach den kürzesten Verbindungskanten gesucht. Diese beginnen alle im „Notch“ und können dann entweder an einem Punkt des Polygons oder an der Orthogonalprojektion des „Notches“ auf die jeweils nachfolgende Außenkante enden. In Abbildung 47 sind alle möglichen Verbindungskanten vom „Notch“ 8 aus in Grün eingezeichnet. Die roten Kanten außerhalb des gelben Winkelbereiches würden durch das eigene Hindernis hindurchführen und sollen daher nicht in Betracht gezogen werden. Ziel ist es nun, aus dieser Sammlung an grünen Kanten nur die Wichtigsten für die Bestimmung der kürzesten Abstände auszuwählen. Dafür werden alle Kanten ihrer Länge nach sortiert und beginnend bei der Kürzesten solange welche hinzugefügt, bis die Konvexitäts-Bedingung für den „Notch“ erfüllt ist. In Abbildung 48 kann man erkennen, dass dies im Beispiel vom „Notch“ 8 bereits nach zwei Kanten der Fall ist. Die beiden kürzesten Kanten wurden dort ausgewählt und da es nun zwischen keiner der Kanten an „Notch“ 8 mehr einen Winkel größer 180° gibt, ist die Konvexitäts-Bedingung erfüllt. Die restlichen grünen Kanten aus Abbildung 47 würden demnach verfallen. Dadurch werden gleichzeitig alle Kanten, die durch Hindernisse hindurchführen würden, wie bei der Kante zu Punkt 3 aussortiert. Denn dieses Hindernis hat ebenfalls Punkte und Kanten, die deutlich näher am „Notch“ dran sind. Der rote Bereich in Abbildung 48 kenn-

zeichnet außerdem den ursprünglichen Winkelbereich des ANavMG. Beide gefundenen kürzesten Kanten befinden sich außerhalb dieses Bereichs. Dadurch wird nochmal sehr deutlich gezeigt, warum es zum Finden der kürzesten Kanten den größeren, gelben Bereich geben muss.

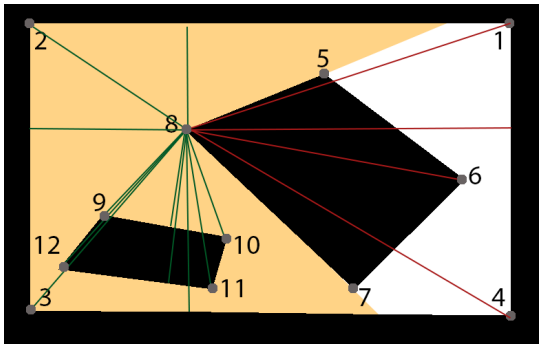


Abbildung 47: Alle möglichen Verbindungskanten für den „Notch“ am Punkt 8. Rote Kanten liegen außerhalb des gelben Winkelbereichs und werden nicht in Betracht gezogen.

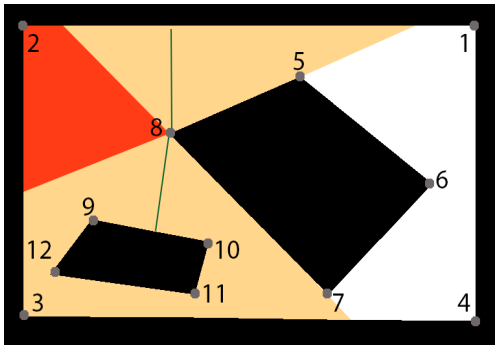


Abbildung 48: Bereits zwei Kanten reichen für die Unterteilung des „Notches" 8 aus. Beide wären im rot eingezeichneten, ursprünglichen Winkelbereich des ANavMG nicht abgedeckt gewesen.

Um die Liste aus allen möglichen Punkten bereits im Vorhinein etwas zu verkleinern und damit das Sortieren zeitlich zu optimieren, kann eine Maximallänge der Kanten eingeführt werden. Diese würde durch die erste Kante vorgegeben werden, die allein die Konvexitäts-Bedingung erfüllen würde. In Abbildung 49 wird das äußere rechteckige Polygon also gegen den Uhrzeigersinn als erstes überprüft und speichert die Verbindungskante zu Punkt 1 in der Liste. Die nächste Kante zu Punkt 2 erfüllt die Konvexitäts-Bedingung bereits und legt damit die Maximallänge fest. Bereits direkt im Anschluss wird diese vom Abstand zu Punkt 3 weiter eingeschränkt, womit der Punkt in der unteren linken Ecke nicht mehr in Betracht gezogen wird. Dass diese beiden Kanten die einzigen sind, die die Konvexitäts-Bedingung erfüllen, kann man auch an dem ursprünglichen Winkelbereich des AnavMG erkennen.

In diesem Beispiel sind zwar immer noch viele Kanten in der Liste vorhanden, aber gerade bei größeren Spielfeldern, kann sich diese Eingrenzung lohnen. Weitere Optimierungen wurden bisher nicht vorgenommen, aber sicherlich wäre es auch sinnvoll, die Zahl der zu überprüfenden Hindernisse generell einzuschränken und bei Hindernissen in der näheren Umgebung mit dem Überprüfen zu beginnen. Damit wäre die Maximallänge nämlich von Beginn an sehr klein festgelegt und später müssten weniger Kanten sortiert werden.

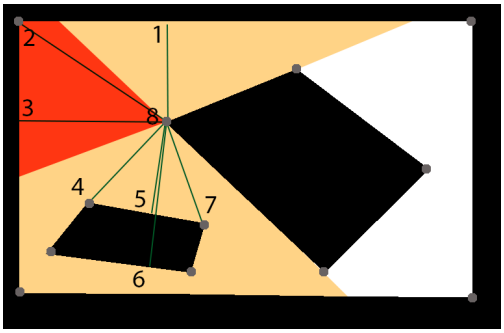


Abbildung 49: Die Kanten zu den Punkten 2 und 3 würden jeweils alleine ausreichen, um den Winkel an Punkt 8 so zu unterteilen, dass dort zwei konvexe Polygone entstehen.

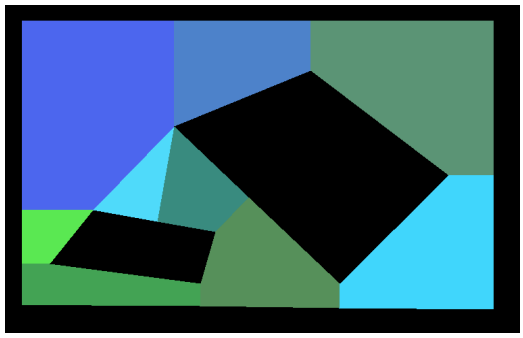


Abbildung 50: Ergebnis der Unterteilung durch die jeweils kürzesten Abstände an den "Notches"

Falls ein „Notch“ bereits durch die Kante eines anderen „Notches“ unterteilt wurde und dadurch die Konvexitäts-Bedingung von Anfang an erfüllt, müssen alle kürzeren Kanten trotzdem noch hinzugefügt werden.

Hat der Algorithmus auf diese Weise das komplette Polygon inklusive aller Löcher abgearbeitet, werden überflüssige Kanten wieder gelöscht. Dafür müssen sie mehrere Bedingungen erfüllen:

1. Es muss sichergestellt sein, dass durch ein Fehlen der Kante die Bedingung der Konvexität nicht verletzt wird.
2. Die Kante darf nicht kleiner sein als benachbarte Kanten. Mit benachbart sind Kanten gemeint, die beim Durchlaufen der Polygone im Sinne eines NavMeshs vor oder nach der jeweiligen Kante besucht werden würden. Dies ist genau dann der Fall, wenn die Endpunkte der Kanten benachbart bzw. miteinander verbunden sind.
3. Für den Punkt am anderen Ende der Kante müssen die vorherigen Bedingungen ebenfalls erfüllt sein.

Falls Kanten sich überkreuzen sollten, bleibt nur die kürzere Kante bestehen.

Sind auf diese Weise alle Kanten reduziert, wird das begehbare Polygon entlang der Kanten zerschnitten und aneinander liegende Polygone werden miteinander verbunden (vgl. Abbildung 50).

Da in diesem Ansatz nur Kanten an den bestehenden Punkten beginnen können, ist es nicht möglich die Stellen zwischen zwei aufeinander zulaufenden Kanten (wie die Kante 2 in Abbildung 51) als potentielle spätere Engpässe zu finden. Würde die Kannte 3 also nicht gefunden werden, weil sie beispielsweise zu groß wäre, würde man zu dieser Engpass-Region lediglich den Eingang 1 finden. Dabei bestünden an der Kante 2 trotzdem alle Vorteile außer den Nahkampf-Vorteil. Solche Kannten müssten bei Bedarf also noch zusätzlich im Anschluss ermittelt werden, was ein Nachteil für die meisten NavMesh-Ansätze wäre.

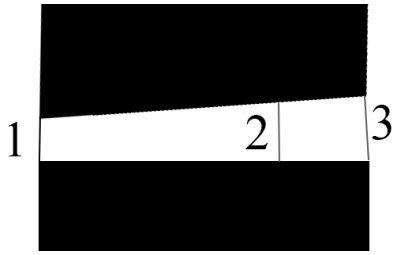


Abbildung 51: Wenn Kante 1 eng genug für einen Engpass wäre, Kante 3 jedoch nicht, würde mit diesem Ansatz Kante 2 nicht gefunden werden.

7.2 Finden von Regionen und Engpässen

Beide oben beschriebene Ansätze zum Finden von Polygonen haben Vor- und Nachteile. Durch die flexible Architektur spielt es keine Rolle, für welchen man sich letztendlich entscheidet. Daher ist es auch möglich, später einen ganz neuen Ansatz oder auch das neue NavMesh zu wählen, das für Iron Harvest bereits geplant wird. Dieses sollte natürlich die Anforderungen aus Kapitel 6 erfüllen und außerdem Informationen über die Größe der Polygone liefern können. Mit Hilfe dieser Größe ist es möglich, Regionen ähnlich wie in Perkins Ansatz aus [23] zu bestimmen. Wie genau diese Größe bestimmt werden kann, wird ab der nachfolgenden Seite erläutert.

Genau wie in Dills Ansatz, besteht eine Region in dieser Abwandlung aus mehreren zusammengefassten Polygonen. Während seine Regionen allerdings keine weiteren Rechtecke hinzufügen, wenn der Mittelpunkt der Region dann nicht mehr im Inneren liegen würde, habe ich mich eher an Perkins Regel aus [23, S. 168] orientiert. Demzufolge sind Regionen immer über Engpässe miteinander verbunden und werden daher ausschließlich aufgrund der Spielflächenstruktur voneinander getrennt. Wie in Abschnitt 2.2 festgehalten, sollen diese sowohl durch ein Verhältnis als auch über eine Maximalbreite der Engpässe gefunden werden können. Auf das Verhältnis wird ab Seite 39 genauer eingegangen.

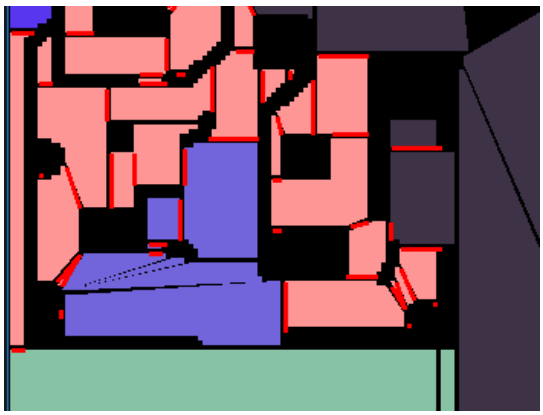


Abbildung 52: Nur die Länge der Kanten allein ist nicht entscheidend, um Engpässe zu finden. Gezeigt ist ein Ausschnitt aus dem Nachbau eines Spielfeldes aus Iron Harvest. Engpass-Regionen sind hier und nachfolgend in hellem Rot gekennzeichnet.

Einfach alle Übergänge zwischen Polygonen durchzugehen und jeden Übergang mit einer Breite kleiner oder gleich der Maximalbreite als Engpass festzulegen, ist für das Finden von Engpässen nicht ausreichend. In Abbildung 52 kann man erkennen, dass diese Variante zahlreiche Engpässe

auch innerhalb von schmalen Gängen findet, die man eigentlich als Engpass-Region bezeichnen könnte. Diese sind für die meisten Vorteile aus Abschnitt 2.3.2 nicht wichtig und selbst für den Fallen-Vorteil wurde festgehalten, dass *jede* Stelle innerhalb einer Engpass-Region den Vorteil bieten würde. Sprich es wäre sinnvoller die Region entsprechend als Engpass-Region zu kennzeichnen statt jede mögliche Stelle als potentiellen Engpass zu speichern. Für den Verschluss-Vorteil kann zusätzlich natürlich, falls vorhanden, noch ein weiterer schmalster Übergang als Engpass gespeichert werden. Dieser würde die Engpass-Region aber nicht unterteilen und stattdessen als Referenz zur Verfügung stehen.

Um das Finden der unwichtigen inneren Übergänge also zu verhindern, teilt die Maximalbreite der Engpässe stattdessen zwei Grenzbereiche voneinander. Zum einen der Bereich mit allen Werten größer der Maximalbreite und zum anderen der Bereich mit allen Werten kleiner oder gleich der Maximalbreite. Falls es weitere Maximalbreiten für die jeweiligen Vorteile geben soll, kann es dazwischen auch weitere Bereiche geben. Beim Zusammenfassen der Polygone zu einer Region werden dann alle verbundenen Polygone des gleichen Grenzbereichs zusammengefasst. Dabei muss auch die Breite der Verbindungs-Kante diesem Bereich angehören. Denn wie in Abbildung 53 zu sehen, kann es Regionen gleicher Größe geben, die aber nur einen sehr kleinen Übergang zueinander haben und dementsprechend nicht Teil der gleichen Region sein sollten.

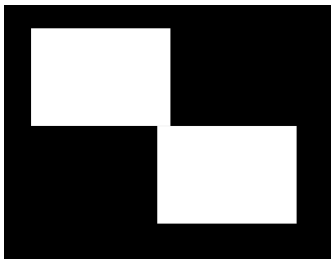


Abbildung 53: Beide Polygone haben die gleiche Größe, haben aber nur einen sehr schmalen Zugang zueinander.

Um ermitteln zu können, welchem Bereich ein Polygon angehört, muss die Größe festgestellt werden. Dabei geht es allerdings nicht um den eigentlichen Flächeninhalt, sondern vielmehr um den Platz, den ein Polygon beispielsweise für den Nahkampf-Vorteil bieten würde. Ein langer Gang könnte einen großen Flächeninhalt haben, bietet dabei aber nicht automatisch mehr Platz am Übergang. Daher ist eher der Durchmesser des größtmöglichen Kreises (nachfolgend Maximaldurchmesser genannt) von Bedeutung. Dieser kann auf sehr unterschiedliche Weisen ermittelt werden, die auch stark vom jeweiligen Ansatz zur Erstellung des zugrundelegenden Nav-Meshs abhängen. Da der Maximaldurchmesser lediglich als Orientierungswert für die Größeneinordnung der Polygone dienen soll, ist keine exakte Genauigkeit erforderlich.

- Im Raster Ansatz aus Abschnitt 7.1.1 hat es sich angeboten, die Seitenlänge des anfangs gefundenen Quadrates zu nehmen. Im selben Abschnitt wird auch beschrieben, wie die Größe bzw. nun der Durchmesser der kleineren Rechtecke, die aus den Kanten der Größeren entwachsen, berechnet werden kann. Diese Werte sind leider als Durchmesser nicht unbedingt korrekt, wie am Ende des Abschnitts erklärt wurde. Aber sie geben trotzdem eine nützliche Auskunft über die Größe eines Polygons und wären damit hierfür ausreichend.
- Alternativ bietet sich die Länge der längsten Kante an, die zu einem verbundenen Rechteck führt. In Abbildung 54 wären die beiden Übergänge sehr schmal und die längste Kante wäre b. Entsprechend würde das Polygon 2 dadurch genauso groß wie der

an b angrenzende Engpass bewertet werden. Ins Innere des Polygons würden aber unter Umständen trotzdem größere Einheiten passen.

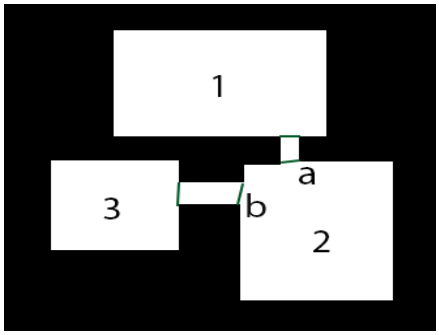


Abbildung 54: Da die Region 2 nur sehr schmale Zugänge besitzt, würde ihre Größe entsprechend falsch eingeschätzt werden.

Auf der einen Seite kann eine größere Einheit nur dann in das Polygon hineinlaufen, wenn eine Verbindungskante mit entsprechender Länge vorhanden wäre. Man könnte also argumentieren, dass an den Übergängen der Region 2 aus Abbildung 54 keine Engpässe notwendig wären, weil beispielsweise für den Flucht-Vorteil aus Polygon 2 gar keine Gefahr durch größere Mechs drohen könnte. Handelt es sich um ein Polygon, in welches Einheiten hinein spawnen/teleportieren können, wäre ein Polygon ohne diese Kanten wahrscheinlich eher ein Fehler im Leveldesign. In diesem Fall wäre die Einheit nämlich im entsprechenden Polygon eingesperrt.

Auf der anderen Seite ist es natürlich wichtig zu wissen, wie groß eine Fläche wirklich ist. Für den Nahkampf-Vorteil hat der Ausgang eines wichtigen Engpasses taktische Vorteile, wenn die Fläche dahinter groß ist. Selbst wenn große Mechs dort nicht hineingelangen könnten, würde eine große Ansammlung aus kleinen Einheiten die feindliche Armee dort überwältigen können. Auch Bunker könnten dort unter Umständen gebaut werden.

Je nachdem, wie die Basisflächen gefunden wurden, bietet sich diese Methode trotzdem an, um auch die Engpässe für den Nahkampf-Vorteil zu finden. Bei einer Triangulierung ist die Region beispielsweise im Inneren von so vielen Kanten durchzogen, dass das Problem höchstwahrscheinlich gar nicht auftreten wird. Durch die langen Diagonalen kann aber auch hier der Wert wieder sehr verfälscht werden, was einen anderen großen Nachteil bedeuten würde.

- Falls andere Daten zur Verfügung stehen, kann man natürlich auch über weitere Alternativen nachdenken. Bei Polygonen aus einer Delaunay-Triangulierung könnte man beispielsweise auch den Durchmesser des Kreises nehmen, der durch die drei Punkte verläuft. Genaugenommen wäre dieser allerdings je nach Dichte der Punkte etwas bis viel zu groß, wie man in Abbildung 55 besonders an den oberen mittigen Kreisen erkennen kann.

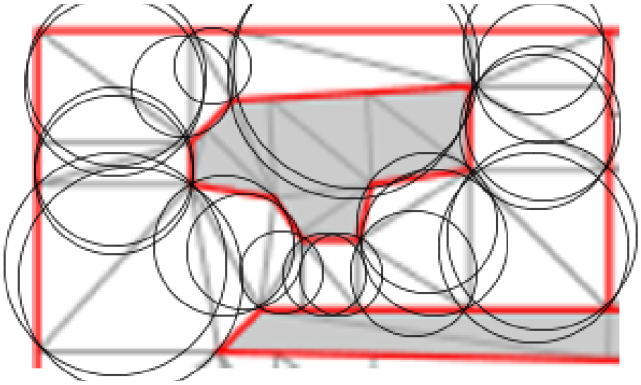


Abbildung 55: Ein Kreis geht immer durch die drei Punkte des jeweiligen Dreiecks. Die Größe des Kreises ist je nach Dichte der Punkte unterschiedlich groß und hilfreich (Bildausschnitt stark vergrößert aus [44, S.12], Kreise hinzugefügt).

- Wenn man es ganz genau wissen möchte, kann man natürlich auch alle Kanten der Hindernisse durchgehen und den kürzesten Abstand suchen. Das wäre aber eine sehr aufwendige Variante.

Der Maximaldurchmesser einer Region entspricht dem des größten enthaltenen Polygons und wird mit jedem hinzugefügten Polygon aktualisiert. Da sich der Maximaldurchmesser dadurch ständig verändert, ist das Ergebnis stark davon abhängig, mit welchem Polygon die Region begonnen wurde und welche Nachbarn als erstes betrachtet werden. Dieses Problem lässt sich vermeiden, wenn beim Bestimmen von Regionen immer mit dem größten noch nicht zugeteilten Polygon begonnen wird. In meinem ersten Ansatz zur Bestimmung von Polygonen ist dies automatisch der Fall, da das erste Polygon jenes mit dem größten Quadrat entspricht.

Beim Überprüfen eines neuen Polygons, das in eine Region aufgenommen werden soll, ist das durch Parameter bestimmte Verhältnis zwischen dem größeren und kleineren der jeweiligen Durchmesser entscheidend. Hinzugefügt wird das Polygon nur, wenn das Verhältnis groß genug ist. Somit könnte ein Polygon mit dem Maximaldurchmesser 60 einer Region mit dem Maximaldurchmesser 100 bei einem Verhältnis von 50% noch hinzugefügt werden. Alle Polygone mit einem Maximaldurchmesser unter 50 könnten dies jedoch nicht. In klar abgegrenzten Spielfelder liegt ein günstiges Verhältnis zwischen 30% und 50%. Bei 0% gäbe es entsprechend gar keine Unterteilung durch das Verhältnis und bei Werten zwischen 50% und 100% gäbe es zu viele.

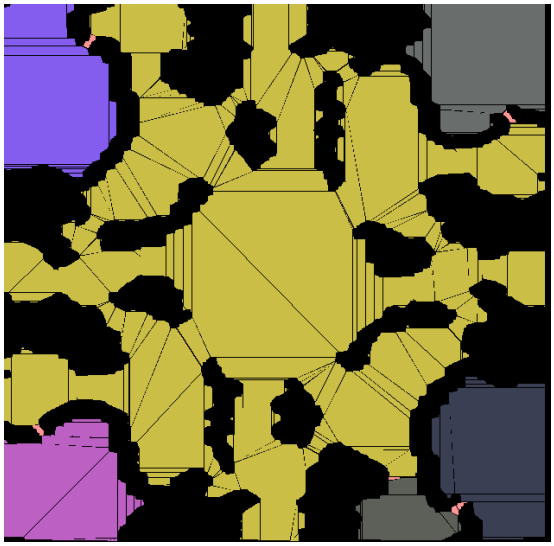


Abbildung 56: Bei kleinen Engpassbreiten ohne weitere Unterteilung durch das Verhältnis, ergibt sich in der Mitte der Karte eine zu große Region mit zu wenig Details (Spielfeld nachgebaut aus [23, S. 169]).

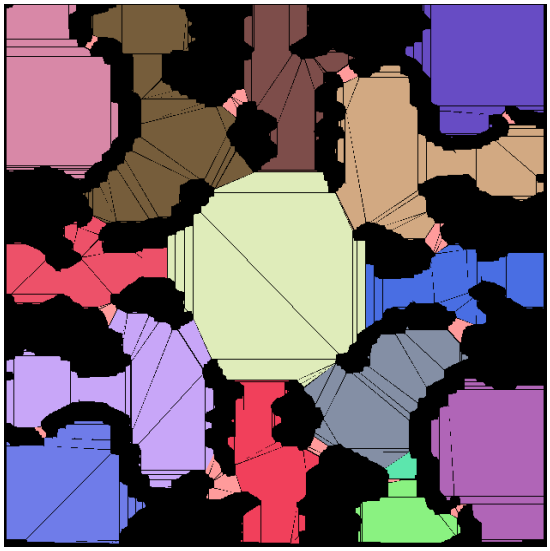


Abbildung 57: Durch zusätzliche Unterteilungen im größten Grenzbereich werden sehr aussagekräftige Regionen gefunden. Die Region in der Mitte ist zwar sehr groß, repräsentiert dabei aber die große freie Fläche sehr gut (Spielfeld nachgebaut aus [23, S. 169]).

Diese Unterteilung nur nach Verhältnis ist ausschließlich im größten Grenzbereich sinnvoll. Durch ihn wird nämlich verhindert, dass Regionen zu groß werden und somit zu viele Informationen zusammenfassen, wie am Unterschied zwischen den Abbildungen 56 und 57 gut zu erkennen ist. Wichtig ist natürlich weiterhin, dass die Region nicht unterteilt wird, *weil* sie zu groß wird, sondern weil es an dieser Stelle eine Verengung gibt! Da diese durch das Verhältnis gefundenen Übergänge für den Verschluss-Vorteil wichtig sein könnten, können sie ebenfalls als Engpässe bezeichnet werden.

In kleineren Grenzbereichen ist eine zusätzliche Unterteilung und damit das Finden von weiteren Übergängen als Engpässe nur sinnvoll, wenn es an diesen Stellen weitere Vorteile geben würde. Da dies bereits durch das Hinzufügen weiterer Maximalbreiten ermöglicht wird, gibt es dafür keine Notwendigkeit. Falls es in Zukunft aber trotzdem gewünscht wird, stellt der Algorithmus für die Unterteilung der Regionen folgende verschiedene Varianten als Flag bereit.

1. Es wird komplett ohne Verhältnis und nur an den Maximalbreiten der Grenzbereiche unterteilt (vgl. Abbildung 56).
2. Es wird nur nur durch das Verhältnis unterteilt.
3. Es wird jeweils nach beidem, Verhältnis und Maximalbreiten unterteilt.
4. Es wird ausschließlich im größten Grenzbereich mit Verhältnis und sonst mit den Maximalbreiten unterteilt (vgl. Abbildung 57).

Jetzt könnte man sich die Frage stellen, ob es nicht generell sinnvoller wäre, wie bei 2. nur mit einem Verhältnis und gar nicht mit maximalen Engpassbreiten zu arbeiten. Die Entscheidung, ob ein Übergang dann als Engpass gewertet werden kann, könnte man ja der KI überlassen. Das Problem bei dieser Überlegung kann man in Abbildung 58 gut erkennen. Hier würde man den Engpass a für unwichtig halten, weil dieser größer der Maximalbreite ist, obwohl die Fläche da-

hinter sich noch weiter verengt. Daher hat die maximale Engpassbreite immer eine höhere Priorität. Alternativ zu dem Verhältnis, würde es sich natürlich auch anbieten, stattdessen weitere größere Grenzwerte zu den maximalen Engpassbreiten hinzuzufügen. Dazu zeigt Abbildung 59, dass es dadurch auch wieder Unterteilungen geben würde, die nicht wirklich eine Verengung darstellen, aber eben den Grenzwert knapp überschritten haben.

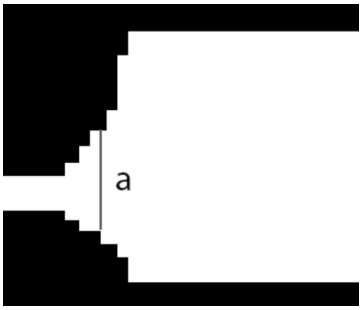


Abbildung 58: Der Übergang a wäre hier zu breit für einen Engpass und der eigentliche Engpass dahinter kann nicht mehr erkannt werden.

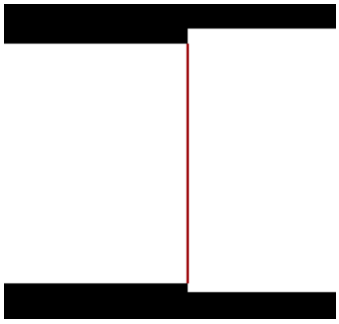


Abbildung 59: Diese Kante wird allein durch ihre Breite als Engpass erkannt, obwohl hier keine wirklich starke Verengung stattfindet.

Obwohl die Ergebnisse für die StarCraft-Karten bereits sehr vielversprechend aussehen, ist für diesen Ansatz noch ein weiteres Kriterium für die Unterteilung in Regionen notwendig. Dies kommt daher, weil die Spielfelder in Iron Harvest viel offener und durch die Hindernisse nicht so klar abgegrenzt sind. In Abbildung 60 ist ein typischer Aufbau dieser Karten zu sehen. Die gefundenen Polygone sind bereits farbig eingezeichnet. Würde man diese wie in der 4. Variante zu Regionen zusammenfassen, müsste das Verhältnis sehr groß gewählt werden, um überhaupt Unterteilungen im größten Grenzbereich zu bewirken (vgl. Abbildung 62). Andernfalls hätte man wieder das Problem, dass die Regionen zu wenig Abstraktion bieten würden.

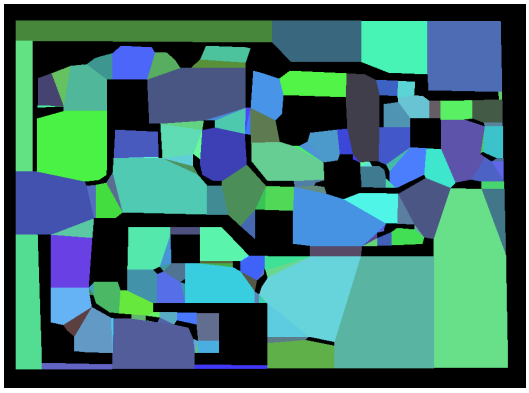


Abbildung 60: Polygone nach eigener Abwandlung des ANavMG im Ausschnitt aus einer Iron Harvest Karte

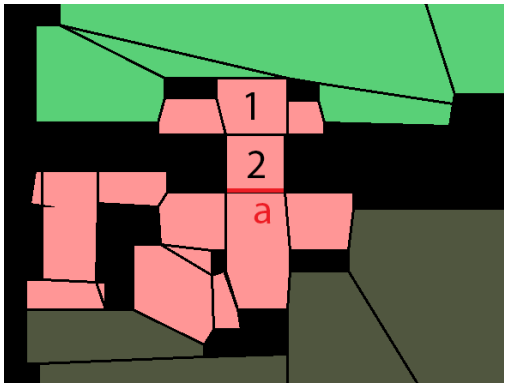


Abbildung 61: Die Kante a wird als eigentlich wichtiger Engpass nicht gefunden, weil alle umliegenden Polygone ungefähr gleichgroß sind.

Aber auch sehr wichtige Engpässe würden unter Umständen nicht gefunden werden, weil sie womöglich im Inneren einer Engpass-Region liegen. Ein solcher Engpass wäre die rote Kante a in Abbildung 61. Da alle umliegenden Polygone kleiner der Maximalbreite für Engpässe sind, würden sie zu einer Engpass-Region zusammengefasst werden. Normalerweise ist dies auch richtig so, weil dann später nur die Zugänge bzw. generell die Übergänge zwischen Regionen

auf ihre Wichtigkeit als Engpass überprüft werden. An diesen bestehen normalerweise auch die meisten Vorteile. Aber selbst, wenn man mehrere Maximalbreiten für Engpässe definieren würde, wären Polygon 1 und 2 immer Teil der selben Region, da sie fast gleichgroß sind. Auch Perkins Regeln für das Finden von Regionen aus Abschnitt 5.1 würden Kante a wahrscheinlich nicht finden, da dort Engpässe entfernt werden, wenn sie einen ähnlichen Radius wie die Regionen haben.

Abbildung 62: Zusammengefasste Regionen durch die Maximalbreite 16 und ein Verhältnis von 90% (Berechnung des Durchmessers durch die Kantenlängen)

Abbildung 63: Zusammengefasste Regionen durch die Maximalbreite 16, ein Verhältnis von 30% und die Unterteilung durch die Umweg-Bedingung

Um diese Probleme zu lösen, wird ein weiteres Kriterium für die Unterteilung bestimmt, die den Algorithmus allerdings deutlich langsamer macht. Ursprünglich sollte dies lediglich ein Kriterium für die Wichtigkeit eines Engpasses sein und dementsprechend an deutlich weniger Kanten überprüft werden müssen. Kante a ist genau deshalb von so einer hohen Bedeutung, weil sie die Umweg-Bedingung erfüllt. Genau dies muss also auch bereits beim Bilden von Regionen mit überprüft werden. Dabei kann die Umweg-Bedingung auch von einem Set aus mehreren Engpässen erfüllt werden. Die Kante a wäre ebenfalls ein wichtiger Engpass, wenn sie durch ein weiteres Hindernis in der Mitte getrennt wäre. In Abbildung 63 sieht man das Ergebnis der Unterteilung zusammen mit diesem neuen Kriterium im Vergleich zu Abbildung 62.

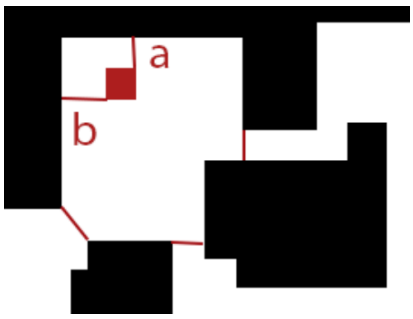


Abbildung 64: Die gefundenen Engpässe am roten POI sind für die Verteidigung der Region unbrauchbar.

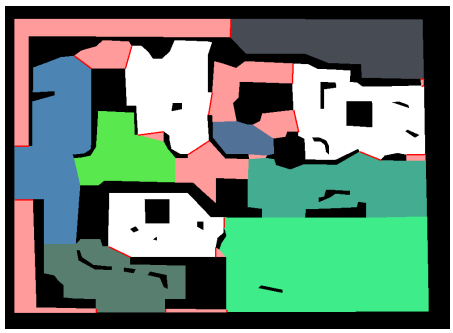


Abbildung 65: Drei weiße POI-Regionen haben sich bis zu den Engpässen ausgedehnt.

Neben normalen Regionen gibt es in diesem Ansatz auch POI-Regionen. Diese enthalten die POI's wie beispielsweise Flaggen oder Ressourcengebäude und werden als Erstes erstellt. Dafür werden zunächst alle an dem POI angrenzenden Polygone zusammengefasst. Dies soll verhin-

dern, dass Engpässe gefunden werden, die wie in Abbildung 64 zwischen einem POI und einem anderem Hindernis liegen. Die Engpässe an den POI-Regionen sollen vor allem für die Verteidigung gefunden werden. Ist der Gegner bereits direkt am POI, wäre ein angrenzender Engpass nicht mehr hilfreich. Anschließend dehnt sich jede Region weiter aus und soll natürlich an den Engpässen gestoppt werden. In Abbildung 65 sieht man die gefundenen POI-Regionen in Weiß.

Ist ein Polygon bereits Teil einer anderen POI-Region, können diese sich außerdem zu einer gemeinsamen großen Region zusammenschließen. Entsprechend sollte innerhalb der Region gespeichert werden, wie viele POI's in ihr enthalten sind, damit daraus später von der KI berechnet werden kann, wie wichtig diese Region für die Strategie ist.

Bei manchen POI-Regionen sind die Zugänge besonders breit. Somit würde eine vielleicht trotzdem sehr günstige Kante nicht gefunden werden, wenn sie die maximale Engpassbreite überschreitet. In anderen Fällen bricht eine Region beim Hinzufügen weiterer Polygone zu früh ab, obwohl man einen viel kleineren Zugang hätte finden können, der aber noch weiter außen liegt. Aus diesem Grund ist es den Leveldesignern möglich, für entsprechende Regionen individuelle Engpassbreiten einzustellen.

Nachdem jedes Polygon einer Region zugeteilt wurde, können einige Regionen mit anderen verbunden werden, um bereits unnötige Übergänge auszusortieren. Dadurch entfallen also gleich Engpässe, die ohnehin nicht wichtig gewesen wären. Engpässe an Sackgassen sind beispielsweise nur wichtig, wenn es sich dabei um eine POI-Region handelt. Eine Sackgasse ist es auch, wenn es zwar mehrere Übergänge gibt, aber alle zur selben Region führen.

Die zusammengefügte Region muss anschließend ebenfalls nochmal überprüft werden. Durch das Wegfallen aller Übergänge zur ersteren Region, könnte auch sie wieder eine Sackgasse sein. Dill beschreibt dies in [25, S. 375] damit, dass eine Engpass-Region mit genau zwei Übergängen, bei der einer davon zu einer Sackgasse führt, ebenfalls Teil dieser Sackgasse sei.

Falls Sackgassen erhalten bleiben sollen, dürfen sich die Regionen nur miteinander verbinden, wenn die Sackgasse einen kleineren oder gleich großen Maximaldurchmesser hat.

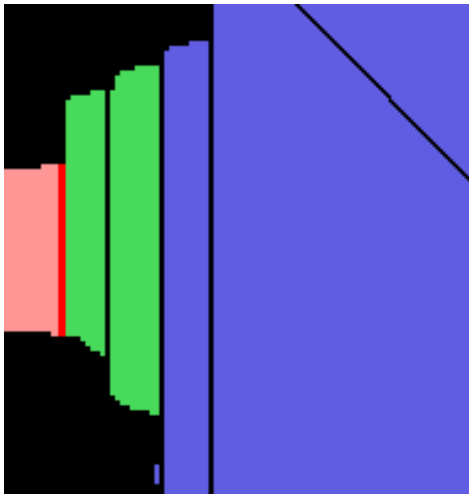


Abbildung 66: Die grüne Region bietet keine zusätzliche Information über die Fläche.

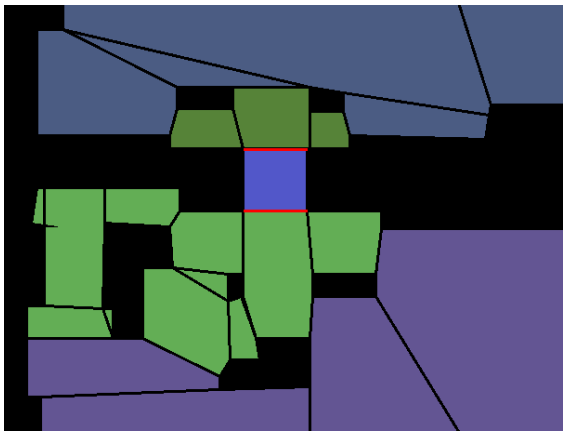


Abbildung 67: Obwohl es bei diesen grünen Regionen mehrere Zugänge gibt, liegen auch diese lediglich zwischen zwei Regionen und bieten keine weiteren Informationen.

Bei insgesamt zwei Nachbarn handelt es sich genau dann um eine unnötige Region, wenn einer davon den Maximaldurchmesser des selben Bereichs hat. Dies ist nur möglich, wenn zusätzlich durch das Verhältnis oder die Umweg-Bedingung unterteilt wurde. In diesem Fall bietet die zu-

sätzliche Region keinen Vorteil, da es sich nicht um eine Kreuzung handelt und die Anzahl der Übergänge sich ohne diese Region nicht verändern würde. Es würde sich wie in Abbildung 66 einfach um eine zusätzliche Region handeln, die zum Erreichen der jeweils angrenzenden Regionen durchlaufen werden müsste. Es können wie in Abbildung 67 sogar unnötige Übergänge wegfallen, wenn mehrere von ihnen zur selben Region führen und die Umweg-Bedingung nicht erfüllen. So würden die grünen Regionen sich den Äußeren anschließen und diese wären anschließend nur noch durch die blaue mittlere Engpass-Region miteinander verbunden. Diese kann sich nicht weiter verbinden, da beide Kanten die Umweg-Bedingung erfüllen.

Am Ende müssen alle in Frage kommenden Übergänge zwischen den Regionen auf ihre Wichtigkeit überprüft werden. In Frage kommen sie genau dann, wenn sie kleiner der größten Maximalbreite der Engpässe sind. Falls allerdings für den Verschluss-Vorteil auch durch das Verhältnis gefundene Übergänge wichtig werden sollen, kommen alle Übergänge zwischen Regionen als Engpass in Frage.

7.3 Wichtigkeit von Engpässen

Wie in den Kriterien aus Abschnitt 2.3.4 bereits beschrieben, ist allein die Tatsache, dass eine Verengung stattfindet, noch kein Hinweis auf die Wichtigkeit eines Engpasses. Zunächst muss bestimmt werden, welche Wege im Spielfeld überhaupt von Bedeutung sind. Ein Engpass, den am Ende niemand durchquert, weil er an einer unwichtigen Stelle wie dem Rand des Feldes platziert ist, wäre nicht von Bedeutung. Dafür wird von jeder POI-Region zu jeder anderen POI-Region der kürzeste Weg für jede Einheitenbreite bestimmt. Für jeden Engpass auf diesen Wegen, wird ein dafür vorgesehener Wert jeweils um eins erhöht. Anschließend hat man also bereits eine gute Basis für die spätere Priorisierung durch eine KI.

Für jeden Engpass, bei dem dieser Wert mindestens bei eins liegt, wird zusätzlich die Umweg-Bedingung überprüft. In Abschnitt 5.2 wurde bereits erklärt, wie dieser Umweg am besten zu finden wäre. Gesucht wird der kürzeste Weg von einem der am Engpass anliegenden Polygone ins andere, ohne den Engpass selbst in die Suche mit einzubeziehen. Dabei kann man, wie Dill in [25, S. 374] vorschlägt, eine Tiefe für die Suche festlegen. Erfolgt die Suche allerdings ohne Tiefe, liefert das Ergebnis genauere Informationen. So könnte ein Engpass umso wichtiger sein, je länger der Umweg ist. In manchen Fällen könnte es sogar passieren, dass gar kein Weg auf die andere Seite gefunden wird. Dann ist der Engpass sogar besonders wichtig, weil er den einzigen Weg in die angrenzende Region darstellt!

Es gibt unterschiedliche Varianten, um die Länge eines Weges bzw. den Weg an sich in einem NavMesh zu finden. Da die Polygone innerhalb der Regionen sehr unterschiedliche Größen haben können, eignet sich ein Weg entlang der Mittelpunkte nicht. Durch diesen wäre es nämlich möglich, dass wie in Abbildung 68 ein zu großer Umweg für die spätere Wertbestimmung gewählt wird. Man könnte sich allerdings auf den Kompromiss einigen, lediglich den Weg entlang der Kanten zu wählen.

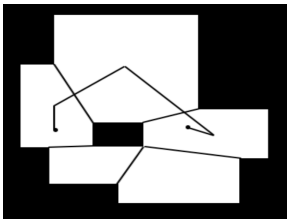


Abbildung 68: Verbindet man die Mittelpunkte der Polygone, ist der Umweg oft viel größer als benötigt.

Ansonsten sind natürlich auch alle Engpässe wichtig, die Zugänge zu POI-Regionen bilden. Hier gab es von KING Art die Vorgabe, dass man sowohl einen Maximalwert für die Summe aus allen Übergangsbreiten als Parameter übergeben können soll als auch als Alternative eine Maximalanzahl an Zugängen. Wären diese Werte also überschritten oder bei Letzterem einer der Zugangsbreiten größer als die größte Maximalbreite der Engpässe, soll für die jeweilige POI-Region dann gar kein Engpass gefunden werden. Jeder POI soll diese Sets ansonsten als Liste aus Referenzen zu den Engpässen enthalten. Alle anderen Engpässe, dessen Umwege größer als ein ebenfalls übergebener Parameter für den Minimalumweg sind oder bei denen es keinen Alternativweg gibt, soll hingegen als eigener POI für die KI dienen. Natürlich kann es so auch Engpässe geben, auf die beides zutrifft.

8 Ergebnis und Ausblick

Die Ergebnisse, die durch verschiedene Ansätze und Abwandlungen erzielt werden konnten, sind ziemlich vielversprechend. Allerdings hat jeder Ansatz auch seine Vor- und Nachteile. Bei der Findung von Polygonen hat sich die zweite Abwandlung des AnavMG als am Vielversprechendsten herausgestellt, da dieser im Gegensatz zur ersten Abwandlung alle kürzesten Abstände findet. Diese sind für den nächsten Schritt zwingend notwendig, da nur Kanten als Engpässe gefunden werden können, die in der Grundstruktur auch existieren.

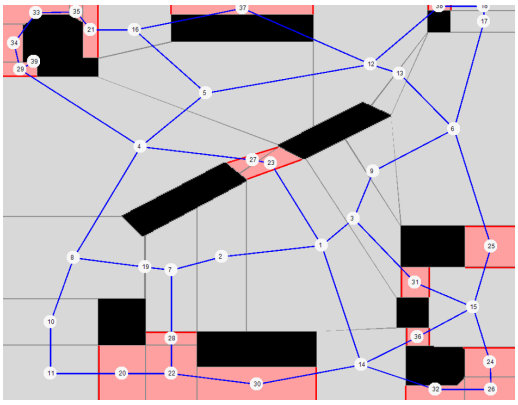


Abbildung 70: Ergebnis des Prototypen durch den Ansatz aus 7.1.1 (Spielfeld aus [17, S. 177])

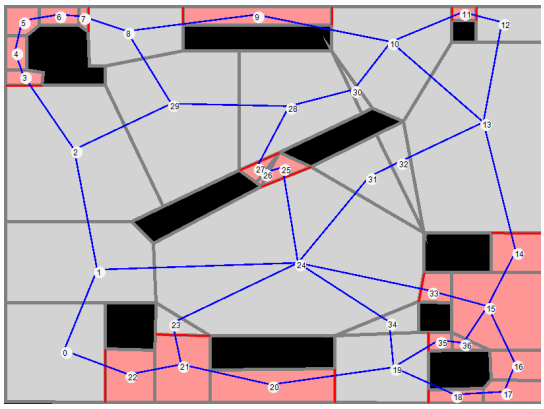


Abbildung 69: Ergebnis des Prototypen durch die Abwandlung des ANavMG aus 7.1.2 (Spielfeld aus [17, S. 177])

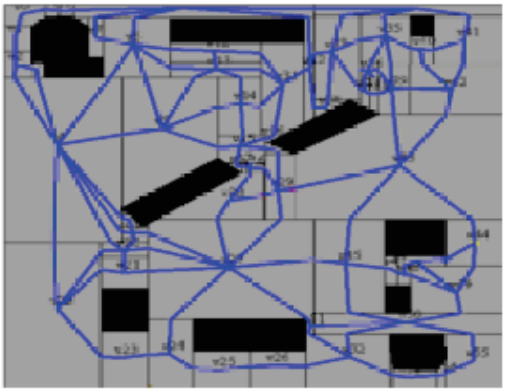


Abbildung 71: Ergebnis DEACCON (stark vergrößerter Ausschnitt aus [17, S. 177])

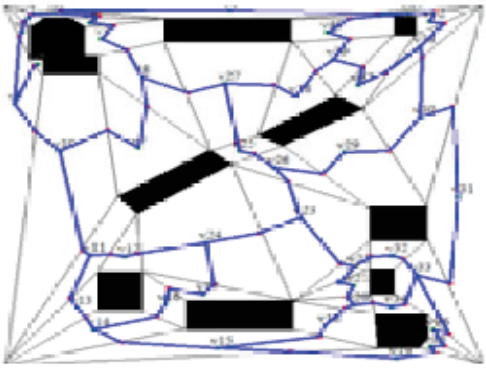


Abbildung 72: Ergebnis Hertel-Mehlhorn (stark vergrößerter Ausschnitt aus [17, S. 177])

In den Abbildungen 70-72 kann man beide Ergebnisse miteinander aber auch nochmal mit dem Hertel-Mehlhorn-Algorithmus und dem DEACCON vergleichen. Die blaue Linie zeigt die Verbingungskanten des so gefundenen Graphen. Hier zeigt sich, dass die Anforderungen aus Kapitel 6 für ein NavMesh zum Finden von Engpässen (die nur von der Abwandlung des ANavMG in Abbildung 69 erfüllt werden) ebenfalls für die Erstellung eines NavMeshs gut geeignet sind. Sind die kürzesten Kanten richtig und verlässlich gesetzt, kann man sehr einfach an der Kantenlänge ablesen, ob die gewählte Einheit die Verbindung passieren kann oder nicht. Es muss also nicht wie beim Corner-Graphen für jede Einheitengröße ein eigenes NavMesh

erstellt werden. In den Abbildungen 73 und 74 sieht man außerdem nochmal die ursprüngliche Karte des ANavMG mit meiner Abwandlung im Vergleich. Durch das Ausweiten des Winkelbereiches, ist es möglich, weitere kürzeste Abstände zwischen den Hindernissen zu finden.

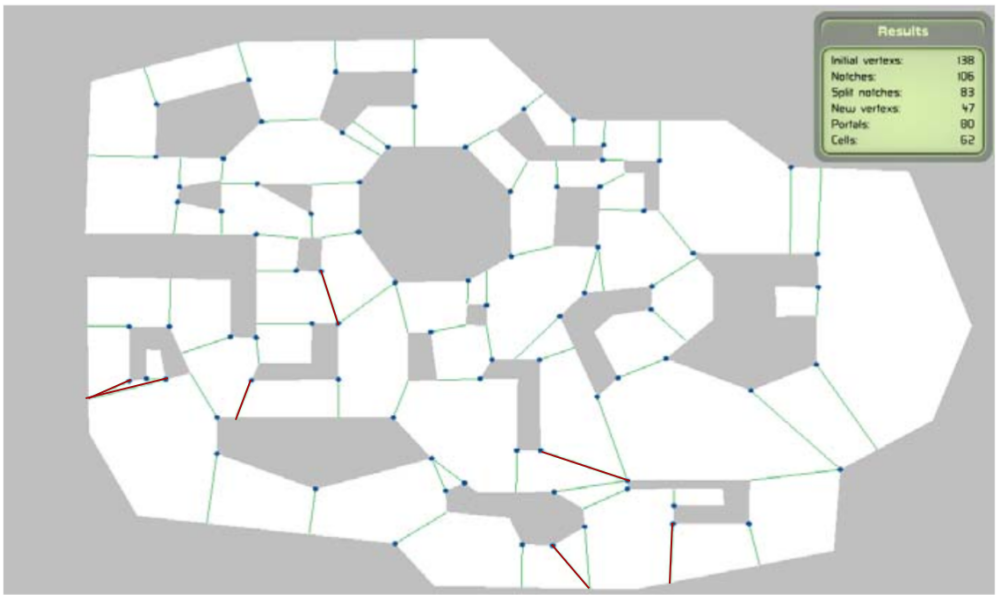


Abbildung 73: Die rot markierten Übergänge entsprechen nicht den kürzesten Abständen zwischen den Hindernissen (Bild ohne die roten Markierungen aus [46, S.11]).

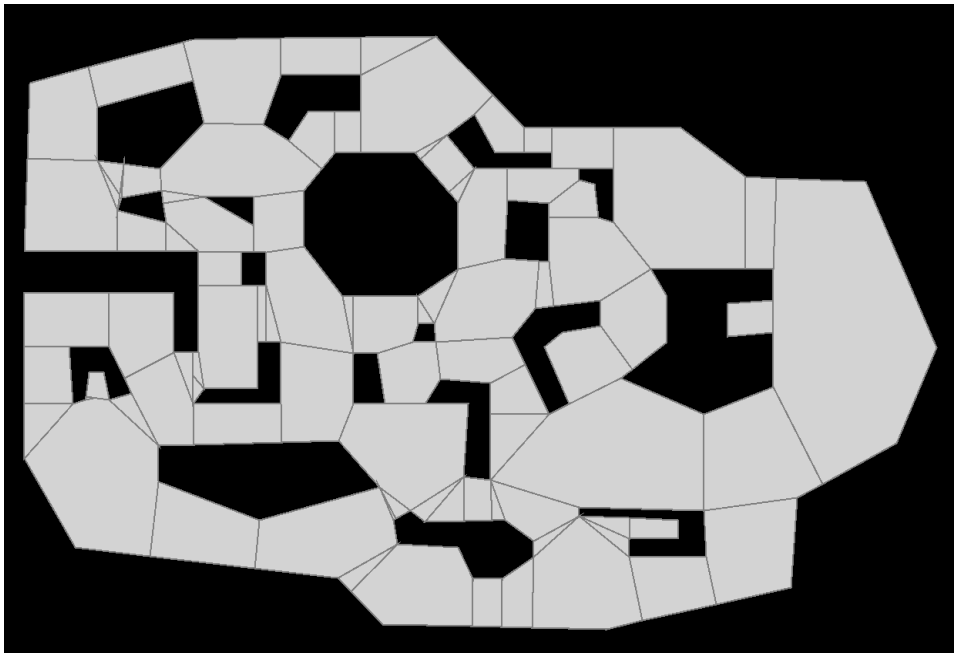


Abbildung 74: Ergebnis der Erweiterung des ANavMG

Für das Unterteilen der begehbaren Fläche in Regionen wurde die Erkenntnis erlangt, dass die Umweg-Bedingung für offene Spielfelder als Kriterium entscheidender ist als das Verhältnis der Größen. Da dies bereits beim Bilden der Regionen an allen Kanten und nicht wie ursprünglich geplant beim Ermitteln der Wichtigkeit von Engpässen überprüft werden muss, wird der

Algorithmus etwas teurer. Grundlegend kann man für das Kriterium der Umweg-Bedingung auch über andere Varianten nachdenken. In Zukunft könnte beispielsweise getestet werden, ob eine Art Heat-Map durch das Finden aller möglichen Wege zwischen POI's durch eine Breitensuche bereits besonders wichtige Kanten kennzeichnen könnte. Diese Information könnte dann beim Bilden der Regionen und der Bestimmung der Wichtigkeit von Engpässen statt dem Berechnen der Umweg-Bedingen für jede einzelne Kante verwendet werden.

In der Abbildung 75 - 77 kann man die gefundenen Regionen oder Engpässe für Spielfelder aus Iron Harvest im Prototyp sehen.

Abbildung 75: Regionen begrenzt durch Umweg-Bedingung

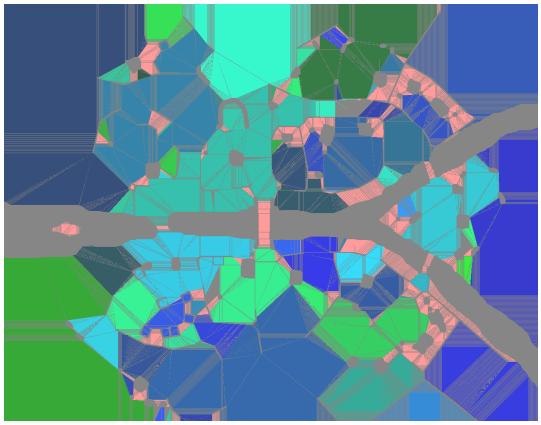


Abbildung 76: Regionen begrenzt durch das Verhältnis

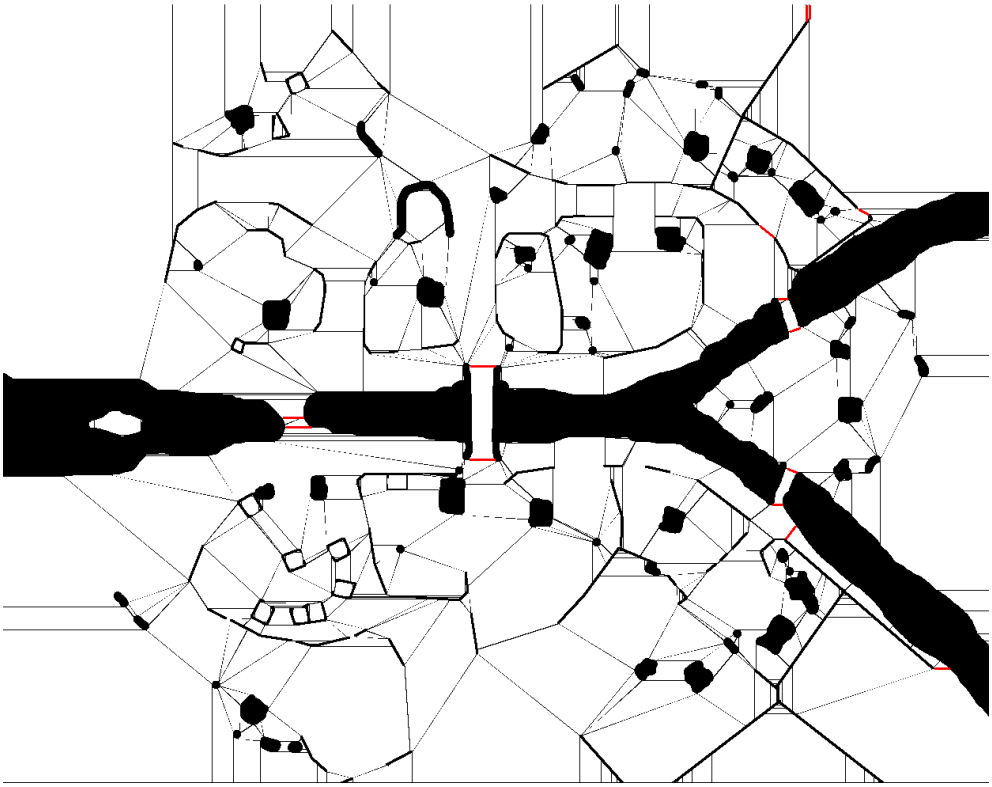


Abbildung 77: In Rot alle potentiell wichtigen Engpässe ohne Berücksichtigung der Wege zwischen POI's

Zuletzt ist in den Abbildungen 78 und 80 auch das Ergebnis des von mir erstellten Tools für den Unity-Editor zu sehen. Dort werden in Blau alle Engpässe für die Verteidigung der POI's angezeigt, in Rot alle sonst wichtigen Engpässe und in Lila alle Engpässe, auf die beides zutrifft. Leider sind die bedeckten Mauern in der Schneelandschaft nur schwer zu erkennen. Bei diesem Spielfeld sind allerdings hauptsächlich die Brücken von Bedeutung. Das andere Spielfeld hat eine sehr offene Struktur und findet keine wichtigen Engpässe, die nicht gleichzeitig auch zur Verteidigung der POI-Regionen dienen. In Abbildung Fehler: Referenz nicht gefunden sind zum Vergleich auch alle Engpässe eingetragen, die nur nach der Umweg-Bedingung wichtig sind, aber nicht auf den Wegen zwischen POI-Regionen liegen und daher aussortiert wurden.

Abbildung 78: Es werden acht POI-Regionen mit blauen Engpässen für die Verteidigung gefunden.

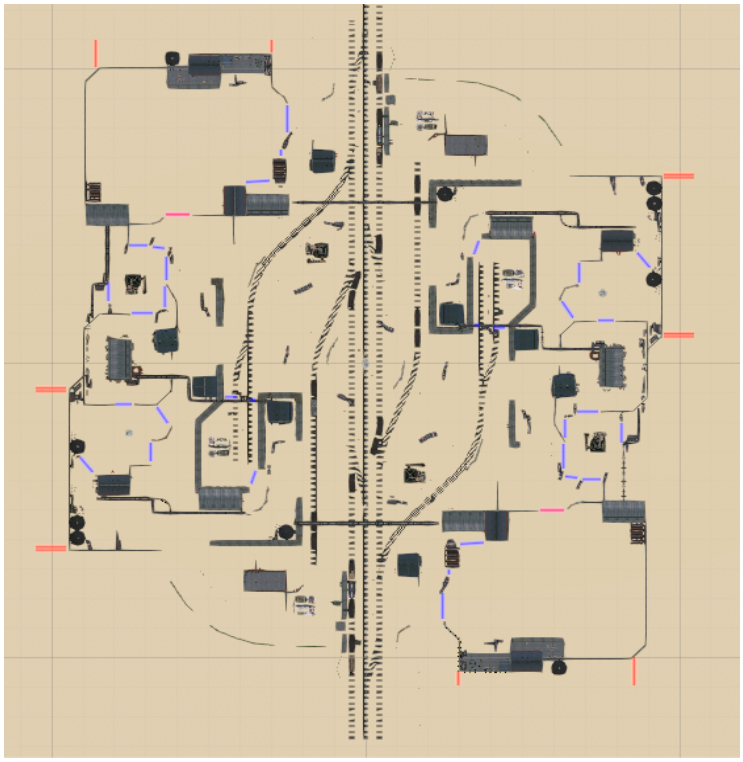


Abbildung 79: Die hier roten Engpässe am Rand der Karte werden aussortiert, da sie nicht auf Wegen zwischen des POI's liegen.

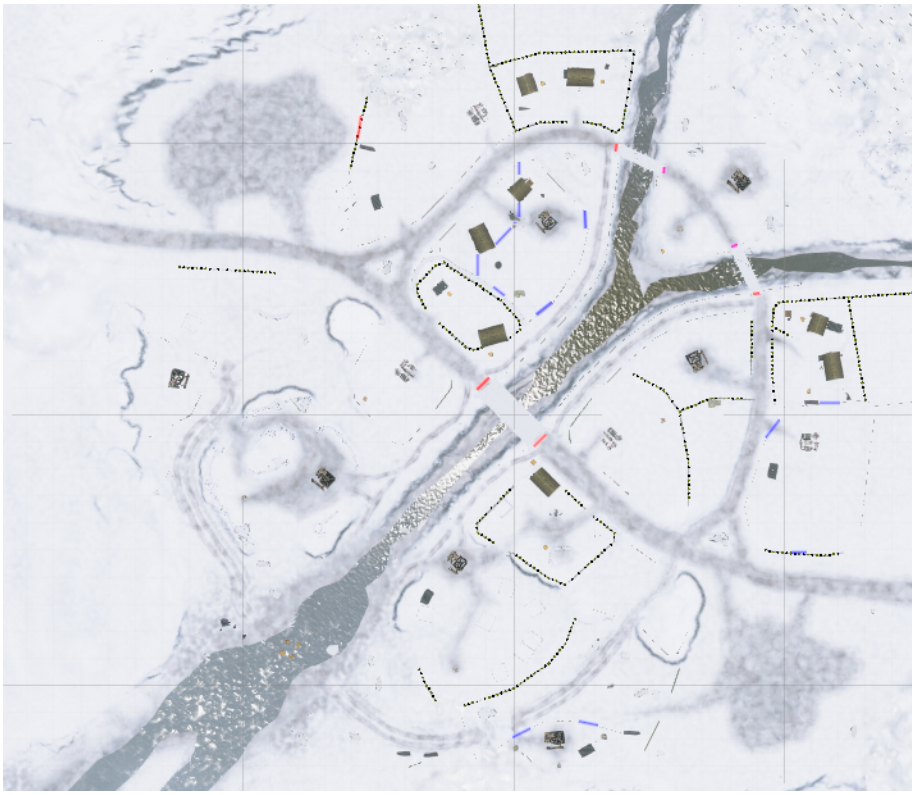


Abbildung 80: Besonders die roten Engpässe an den Brücken sind hier entscheidend.

Langfristig wäre es für den Algorithmus zum Finden von Engpässen durchaus wünschenswert, diesen auch zur Laufzeit aktualisierbar zu machen. Immerhin ist die Umgebung in Iron Harvest zerstörbar und ein Engpass zwischen zwei Gebäuden, die bereits nicht mehr existieren, ist wenig sinnvoll. Ein Updaten wird möglich, sobald die Wegfindung für Iron Harvest vom Corner-Graphen in das NavMesh umgestellt wird. Eine dynamische Berechnung *allein* für das Finden von Engpässen wäre wahrscheinlich zu aufwendig. Da die Anforderungen für die Wegfindung mit unterschiedlichen Einheitenbreiten mit den Anforderungen für das Finden von Regionen und Engpässen übereinstimmen, ist es sehr wahrscheinlich, dass das neue NavMesh von Iron Havest eine geeignete Grundlage für diesen Algorithmus bieten wird. Durch die hohe Modularität von Dills Architektur, ist es kein Problem die Grundstruktur einfach auszutauschen. So wäre es beispielsweise auch denkbar, nach kleineren Anpassungen die Regionen auf Basis des Voronoi Diagramms aus dem BWTA zu erstellen.

Alle gefundenen Ansätze und implementierten Algorithmen müssten natürlich überarbeitet und optimiert werden, wenn diese zur Laufzeit des Spiels eingesetzt werden sollen. Für zukünftige Forschung wäre es auch interessant, diese Ansätze zum Erstellen eines Datensets zu nutzen, welches zum Training einer ML KI verwendet werden könnte.

Als Fazit lässt sich festhalten, dass die Architektur von Kevin Dill eine starke Grundlage bietet, um durch das NavMesh die normale Wegfindung mit taktisch bedeutsamen Informationen wie Engpässen und gegebenenfalls Influence Maps sowie einer höheren Hierarchieebene durch Regionen zu erweitern. Wie die Verwendung von Influence Maps auf diese Struktur übertragen werden kann, wird von Dill in [25, S. 375-387] genauer erläutert. Die Wegfindung bietet in dieser neuen Zusammenstellung nicht nur die Möglichkeit, den kürzesten Weg zum Ziel zu finden, sondern ermöglicht es darüber hinaus auch, möglichst sichere Wege zu wählen. So könnten Gegner gezielt umgangen, taktische Manöver geplant und Fallen für den Gegner gestellt werden. Bereits wenige Informationen, die in den Regionen oder auch direkt an Engpässen gespeichert werden, können die KI deutlich intelligenter wirken lassen und das Erlebnis für den Spieler daher interessanter gestalten. Durch das Speichern der Anzahl von Einheiten, die an einem Engpass oder in einer Region gestorben sind, könnte die KI beispielsweise bestimmte Wege vermeiden und würde somit nicht ständig in dieselbe Falle des Spielers laufen. Außerdem ließe sich durch weitere Werte ebenfalls bestimmen, welche Wege von gegnerischen Einheiten besonders häufig durchlaufen werden. An diesen Stellen bietet sich dann das Planen von taktischen Manövern, Platzieren von Bunkern oder Aufstellen von eigenen weiteren Fallen an.

Quellverzeichnis

[1] „Das Spiel" (2019, Mai 20), Abgerufen von https://kingart-games.com/games/7-iron-harvest/game
[2] Paul Tozour, „Search Space Representations", Steve Rabin (Ed.), 2004
[3] Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill und Mike Preuss, „A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft ", in TCIAIG, 2013.
[4] Paul Tozour, „Influence Mapping" und „Strategic Assessment Techniques", Mark DeLour (Ed.), Game Programming Gems 2, 2001
[5] Paul Tozour, „The evolution of game AI", Steve Rabin (Ed.), 2002
[6] Greg Snook, „Simplified 3D Movement and Pathfinding Using Navigation Meshes", 2000
[7] Raph Koster, „ A theory of fun for game design", O'Reilly Media, Inc., 2013
[8] Stuart Russell und Peter Norvig, „Artificial Intelligence: A Modern Approach", Prentice Hall, 1995
[9] Lawrence Freedman, „Strategy: A History", Oxford University Press, 2013
[10] Gabriel Synnaeve und Pierre Bessière, „A Bayesian model for RTS units controlapplied to StarCraft", Proc. IEEE Comput. Intell. Games, 2011
[11] Khan Adil, Feng Jiang und Shaohui Liu, „State-of-the-Art and Open Challenges in RTS Game-AI and Starcraft", IJACSA 8, 2017
[12] The AlphaStar team, „AlphaStar: Mastering the Real-Time Strategy Game StarCraft II" (2019, Mai 17), Abgerufen von https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/
[13] Pamela McCorduck, „Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence", A K Peters, Ltd., 2004
[14] Choe Sang-Hun, „Google's Computer Program Beats Lee Se-dol in Go Tournament" (2019, Mai 18), Abgerufen von https://www.nytimes.com/2016/03/16/world/asia/korea-alphago-vs-lee-sedol-go.html?_r=0
[15] Michael Buro, „Real-Time Strategy Games: A New AI Research Challenge", 2003
[16] Steve Rabin, „A* Speed Optimizations" und A* Aesthetic Optimizations", 2000.
[17] D. Hunter Hale, G. Michael Youngblood und Priyesh N. Dixit, „Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds", Association for the Advancement of ArtificialIntelligence (AIIDE), 2008
[18] Louis Victor Allis, „Searching for Solutions in Games and Artificial Intelligence", 1994.
[19] Shirish Chinchalka, „An upper bound for the number of reachable positions", International Computer Chess Association, 1996
[20] John Tromp und Gunnar Farnebäck, „ Combinatorics of go", International Conference on Computers and Games, 2006
[21] Alberto Uriarte, „Adversarial Search and Spatial Reasoningin Real Time Strategy Games", 2017
[22] „Region, die" (2019, Mai 09), Abgerufen von https://www.duden.de/rechtschreibung/Region
[23] Luke Perkins, „Terrain analysis in real-time strategy games: An integrated approach to chokepoint detection and region decomposition", in , Artificial Intelligence and Interactive DigitalEntertainment. AAAI Press, 2010
[24] Adi Botea, Martin Müller und Jonathan Schaeffer, „Near Optimal Hierarchical Path-Finding", 2004

[25] Kevin Dill, „Spatial Reasoning for Strategic Decision Making", in Steve Rabin (Ed.), Game AI Pro 2, Boca Raton, FL: CRC Press, 365-388, 2015
[26] „Engpass, der " (2019, Mai 08), Abgerufen von https://www.duden.de/rechtschreibung/Engpass
[27] „Enge, die" (2019, Mai 08), Abgerufen von https://www.duden.de/rechtschreibung/Enge
[28] „Engpass Erläuterung" (2019, Mai 08), Abgerufen von https://deacademic.com/dic.nsf/dewiki/394885
[29] Adam Heinermann, „ Broodwar API" , Abgerufen von https://github.com/bwapi/bwapi
[30] Gabriel Synnaeve und Pierre Bessière, „Special Tactics: a Bayesian Approach to Tactical Decision-making", in IEEE, Computational Intelligence and Games, 409–416, 2012
[31] Alberto Uriarte und Santiago Ontañón, „Improving Terrain Analysis and Applications to RTS Game AI", in Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, 2016
[32] Igor Dimitrijevic, „BWEM library" (2019, Mai 20), Abgerufen von http://bwem.sourceforge.net/
[33] Kenneth D. Forbus, James V. Mahoney und Kevin Dill, „How qualitative spatial reasoning can improve strategy game AIs", EEE Intelligent Systems, 2002
[34] W. Bidakaew, P. Sompagdee, S. Ratanotayanon, and P. Vichitvejpaisal, „Rts terrain analysis: An axial-based approach for improving chokepoint detection method", Knowledge and Smart Technology, 2016
[35] Daniel Higgins, „Terrain analysis in an rts - the hidden giant", , 2002
[36] Julio Obelleiro, Raúl Sampedro und David H. Cerpa, „RTS Terrain Analysis: An Image-Processing Approach", 2008
[37] K´ári Halldórsson und Yngvi Björnsson, „Automated decomposition of game maps", ArtificialIntelligence and Interactive Digital Entertainment, 2015
[38] Chris Jurney und Shelby Hubick, „Dealing with Destruction: AI from the trenches of COMPANY OF HEROES", Game Developer's Conference, San Francisco, CA., 2007
[39] Dave C. Pottinger, „Terrain analysis for real-time strategy games", Game Developers Conference, 2000
[40] William van der Sterren, „Terrain Reasoning for 3D Action Games", Game Developers Conference, 2001
[41] Menelaos Karavelas, „A robust and efficient implementation for the segment Voronoi diagram", Proceedings of the International Symposium on Voronoi diagrams in Scienceand Engineering (VD2004), 2004
[42] Olivier Devillers, „Improved incremental randomized Delaunay triangulation", 1998
[43] Stéphane Lens and Bernard Boigelot, „From Constrained Delaunay Triangulations to Roadmap Graphs with Arbitrary Clearance", 2016.
[44] Stéphane Lens and Bernard Boigelot, „From Constrained Delaunay Triangulations to Roadmap Graphs with Arbitrary Clearance", 2016
[45] Stefan Hertel und Kurt Mehlhorn, „Fast Triangulation of the Plane with Respect to Simple Polygons", International Conference on Foundations of Computation Theory, 1983
[46] Ramon Oliva und Nuria Pelechano, „Automatic Generation of Suboptimal NavMeshes", MIG 2011: Motion in Games, 2011
[47] Fu Chang, Chun-Jen Chen, und Chi-Jen L, „ A linear-time component-labeling algorithm usingcontour tracing technique", in Nikos Paragios (Ed.), Computer Vision and Image Understanding, Amsterdam: Elsevier Inc., 206 - 220, 2004

Anhang

Engpass, der

Wortart INFO — **Substantiv, maskulin**

Häufigkeit INFO

Anzeige

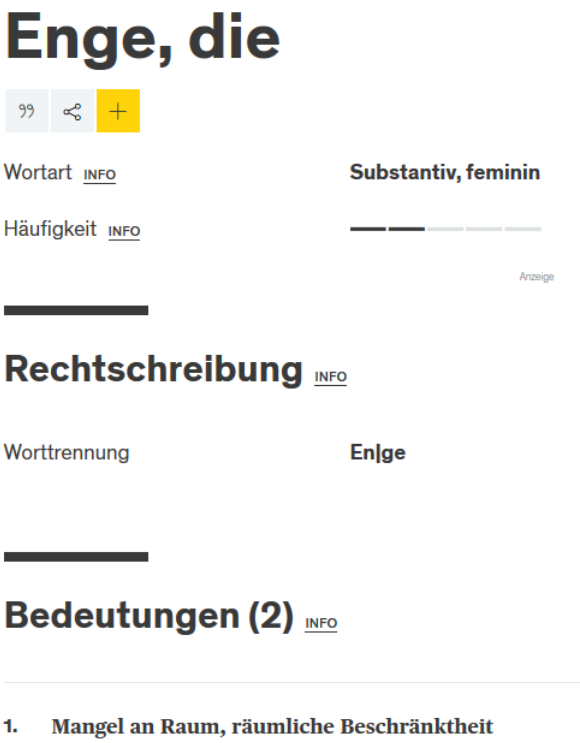
Rechtschreibung INFO

Worttrennung — **Eng|pass**

Bedeutungen (2) INFO

1. **schmale, verengte Stelle auf einem Weg, einer Straße, einem Durchgang o. Ä.**
2. **wirtschaftliche Notlage, schwierige Situation [in der etwas knapp geworden ist]**

Abbildung 81: Screenshot der Quelle [26].



Enge, die

Wortart INFO — **Substantiv, feminin**

Häufigkeit INFO

Anzeige

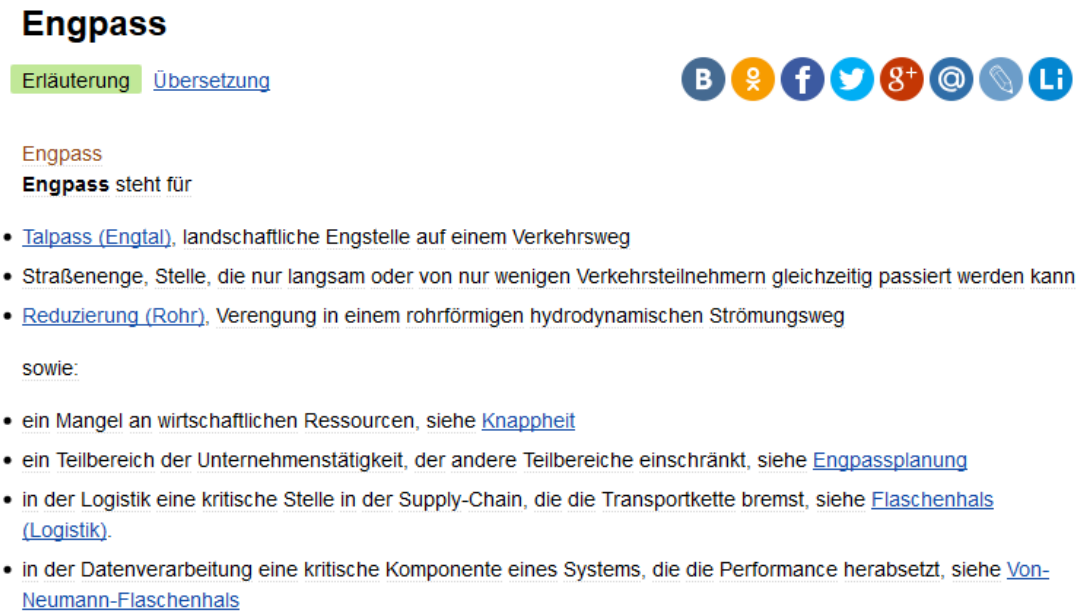
Rechtschreibung INFO

Worttrennung — **En|ge**

Bedeutungen (2) INFO

1. **Mangel an Raum, räumliche Beschränktheit**

Abbildung 82: Screenshot der Quelle [27]



Engpass

Erläuterung Übersetzung

Engpass

Engpass steht für

- Talpass (Engtal), landschaftliche Engstelle auf einem Verkehrsweg
- Straßenenge, Stelle, die nur langsam oder von nur wenigen Verkehrsteilnehmern gleichzeitig passiert werden kann
- Reduzierung (Rohr), Verengung in einem rohrförmigen hydrodynamischen Strömungsweg

sowie:

- ein Mangel an wirtschaftlichen Ressourcen, siehe Knappheit
- ein Teilbereich der Unternehmenstätigkeit, der andere Teilbereiche einschränkt, siehe Engpassplanung
- in der Logistik eine kritische Stelle in der Supply-Chain, die die Transportkette bremst, siehe Flaschenhals (Logistik).
- in der Datenverarbeitung eine kritische Komponente eines Systems, die die Performance herabsetzt, siehe Von-Neumann-Flaschenhals

Abbildung 83: Screenshot der Quelle [28]

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit bisher bei keiner anderen Prüfungsbehörde eingereicht, sie selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Wernigerode,

_____

(Tanja Witke)

Bei der räumlichen Orientierung und dem taktischen Einbeziehen der Umgebung sind Menschen den herkömmlichen Künstlichen Intelligenzen aus Echtzeit-Strategiespielen wie z. B. StarCraft oder Command & Conquer weit überlegen. Um diesen Nachteil auszugleichen und die Herausforderung für den Spieler langfristig aufrecht zu halten, ist die Akquisition von Wissen aus komplexen und dynamischen Spielwelten von zentraler Bedeutung.

