FOP Projekt Wintersemester 2019/20

Yannik Sebastian Hayn, Julian Imhof,
Erik Prescher, Lennart Schmidt

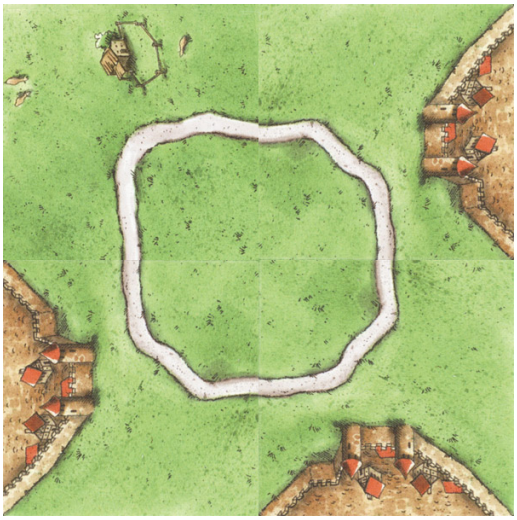
27. März 2020

1 Aufgabe 6.1.5 Theorie

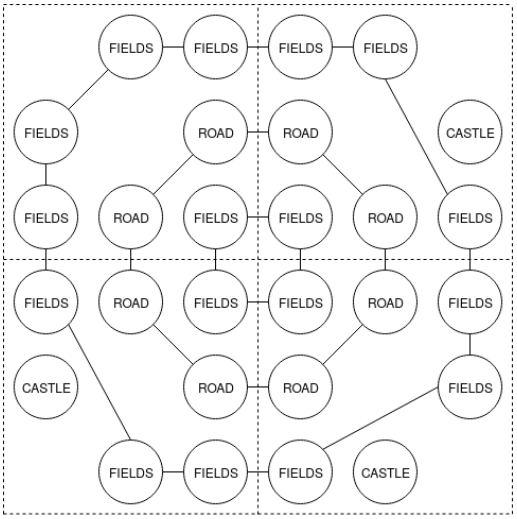
1.1 Azyklischer Graph

Da nur ein gerichteter Graph azyklisch sein kann, der hier gegebene Graph allerdings nicht gerichtet ist, kann er folglich auch nicht azyklisch sein.
Im Folgenden gehen wir allerdings davon aus, dass auch ein nicht gerichteter Graph azyklisch sein könne. Dabei nehmen wir an, dass ein Zyklus aus mindestens drei Nodes bestehen muss und der Zyklus über $n-1$ Kanten durchlaufen werden können muss, ohne dabei zwei mal über eine Kante zu gehen, wobei $n =$ *Anzahl der Nodes* ist.
Im Folgenden zeigen wir, dass der Graph zyklisch sein kann.

Betrachten wir einmal die einfachste Idee, um einen Zyklus zu erzeugen: Ein Kreisverkehr aus mehreren Straßenkarten.



(a) v.l.n.r. Plättchen: V, J, J, J



(b) zugehöriger Graph

Abbildung 1: Eine einfache Möglichkeit mit wenigen Plättchen viele Zyklen zu erzeugen

Wie man sehen kann, sind durch diese 4 Karten bereits 3 Zyklen entstanden.
Der Graph ist somit nicht azyklisch. ■

Wenn man diese Idee ein bisschen weiter führt, kommt man relativ schnell zu dem Schluss, dass der Graph in den allermeisten Fällen spätestens bei Spielende zyklisch sein wird.

1.2 Zusammenhangskomponenten

Für jedes Plättchen p gilt, dass es theoretisch maximal 9 Teilmengen von verschiedenen Zusammenhangskomponenten haben kann.

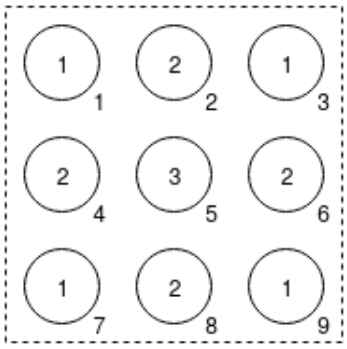


Abbildung 2: zeigt alle maximal möglichen Zusammenhangskomponenten eines einzelnen Plättchens

Für k nicht verbundene Plättchen gilt also:

$$n \leq 9 * k$$

Nur die Knoten eines neu angelegten Plättchens können eine neue Zusammenhangskomponente bilden, wenn sie nicht Teil einer Kante mit einem bereits gelegten Plättchen sind. Da ein neues Plättchen aber nur an mindestens einem alten Plättchen angelegt werden kann, gilt, dass mindestens drei Knoten an einer gemeinsamen Kante liegen, und somit maximal 6 neue Zusammenhangskomponenten existieren können.

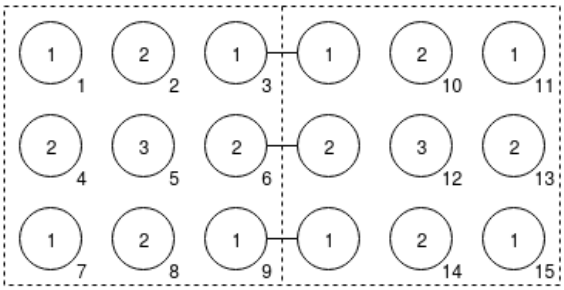


Abbildung 3: zeigt, dass es nur maximal 6 neue Zusammenhangskomponenten geben kann

Somit gilt, wenn man ein neues Plättchen an nur einem alten anlegt:

$$n \leq 9 + 6 * (k - 1)$$

Legt man ein neues Plättchen hingegen an 2 oder mehr Plättchen an, so ergeben sich immer weniger als 6 potentielle neue Zusammenhangskomponenten, wie man Abbildung 4 entnehmen kann.

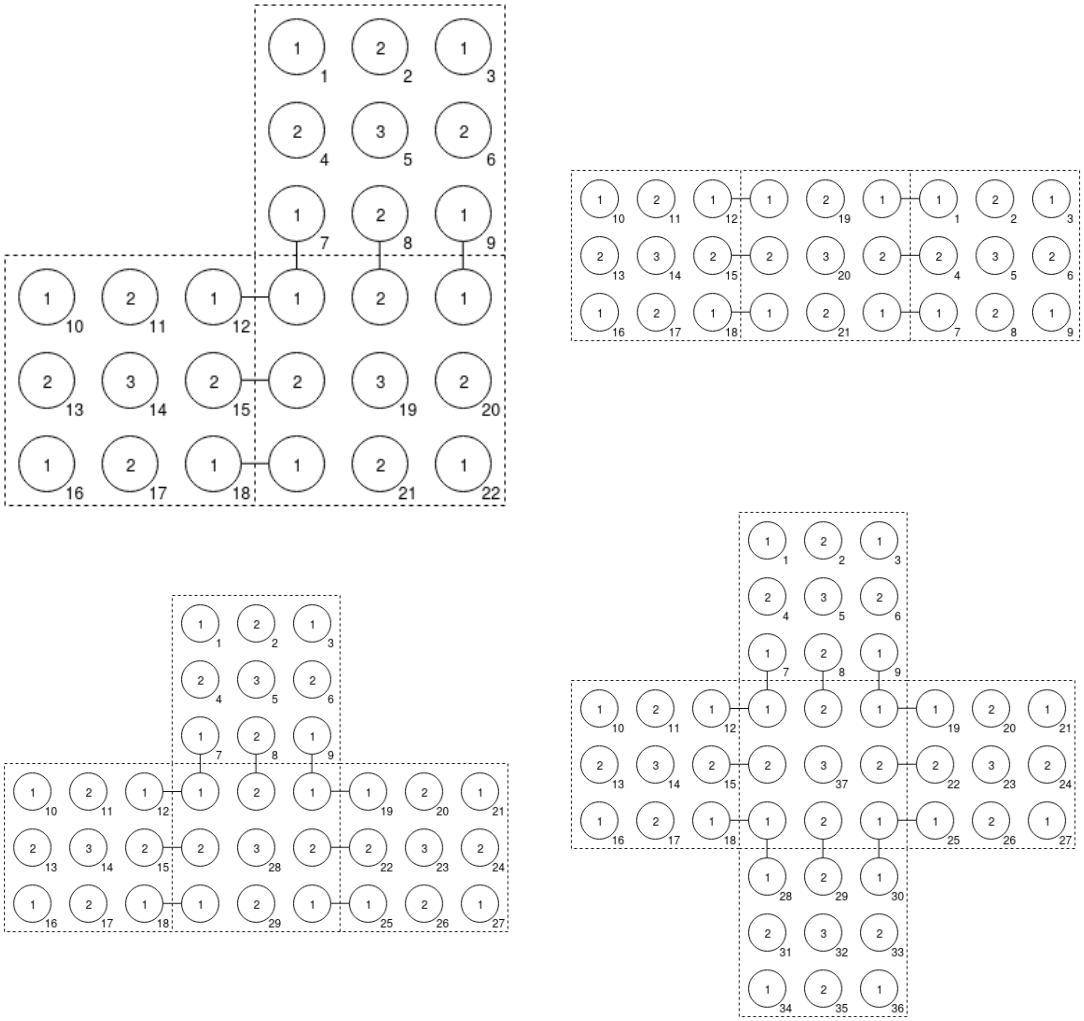


Abbildung 4: Alle Möglichkeiten ein neues Plättchen an mehr als 2 bereits gelegte Plättchen anzulegen

Somit gilt die oben ermittelte Ungleichung also für alle an das Startplättchen angelegte Plättchen.

$$\begin{aligned} & n \leq 9 + 6 * (k-1) \\ \Leftrightarrow\ & n \leq 9 + 9 * (k-1) \\ \Leftrightarrow\ & n \leq 9 * (k-1+1) \\ \Leftrightarrow\ & n \leq 9k \end{aligned}$$

Somit zeigt sich, dass die gegebene Ungleichnug nicht die Präziseste, allerdings korrekt ist. ■

1.3 Laufzeitkomplexität von calculatePoints()

Sei der Graph $G = (V, E)$ gegeben.
In der Methode *calculatePoints(FeatureType type, State state)* werden alle Nodes und Edges im Graph der Reihe nach bearbeitet, um anschließend für jedes zusammenhängende Konstrukt im Spiel die Punkte zu errechnen. Die Reihenfolge ist dabei abhängig von den Konstrukten, was sich jedoch nicht weiter bedeutend auf die Laufzeit dieser Berechnung auswirkt.
Zu jeder Node werden gleichermaßen alle Edges auf Zusammenhängigkeit untersucht. Vereinfacht kann man die Komplexität mit der der folgenden Funktion beschreiben:

```
1 for (node : graph)
2   for (edge : graph)
3     [...]
```

Das entspricht einer Komplexität von $\mathcal{O}(|V| * |E|)$.
Hinzu kommen die Berechnungen zur Bewertung. Die Häufigkeit dieser ist allein abhängig von der Anzahl an Konstrukten.
Sei $|S|$ also die Anzahl an Konstrukten. Dann ist die Komplexität beschreibbar durch

$$\begin{aligned} & \mathcal{O}(|V| * |E| + |S|) \\ = \ & \mathcal{O}(|E| * |V|) \end{aligned}$$

■

2 Dokumentation

2.1 calculatePoints

Die Methode operiert als eine Queue, in die zunächst eine zufällige Node aus der Liste aller Nodes zum entsprechenden Typ geladen wird. Über diese können anschließend mit dem Graph alle verbundenen Nodes, sowie die relevanten Informationen, wie Tiles, Meaples, sowie für Burgen Pennants ermittelt werden. Die verbundenen Nodes werden in die Queue geladen, die ursprüngliche Node daraus entfernt und die weiteren Informationen vorerst in Listen gespeichert. Weiterhin wird zunächst davon ausgegangen, das aktuelle Konstrukt abgeschlossen ist. Sobald eine Node gefunden wird, die sich am Rand befindet, aber keine Tiles verbindet, wird der status auf unabgeschlossen gesetzt. Sind zu einem Konstrukt alle Nodes gefunden worden, d.h. die Queue ist leer, so werden abhängig vom Spielstatus und anhand der ermittelten Informationen die Punkte entsprechend der Spielregeln berechnet und gutgeschrieben.
Bei Wiesen gibt es noch einen Sonderfall. Hier wird jede Node auf anliegende Burgen-Nodes überprüft. Gehört diese zu einer abgeschlossenen Burg, so werden 3 Punkte an den Besitzer der Wiese vergeben.

2.2 Highscore View

In der Highscore View wurde die Schriftart der Highscores auf Celtic geändert, damit das Spiel einen einheitliches Gesamtaussehen und einen durchgängiges Design hat. Außerdem wurden die ersten 3 Highscores in Gold, Silber, Bronze eingefärbt und anschließend mit einem Gradient versehen, um die Anzeige dynamischer zu gestalten. Ab der 3. Zeile wechseln sich Weiß und ein helles Grau ab um die Zeilen besser verfolgen zu können. Die Horizontalen Linien wurden dafür ausgeschaltet.

2.3 Mission