Carcassonne

Aufgabe 6.1 - Der Graph

Aufgabe 6.1.4 - Punkteverteilung

Teil (a)

Um ein Kloster zu bewerten wird über die Menge aller Kacheln auf dem Spielfeld iteriert und nur für jede Kachel, welche einen zentralen Knoten besitzt wird weiter auf das Kloster geprüft und ob sich dort ein Meeple befindet. (Es muss eigentlich ein Kloster sein, allerdings gibt es für eine mögliche Erweiterung des Spiels andere Landschaftstypen zentral auf Kacheln) Wenn die Bedingungen erfüllt sind, wird begonnen die Anzahl an Kacheln um das Kloster zu zählen. Dies wird über 2 verschachtelte Schleifen erzielt., welche auf Referenzen prüfen.

Zum Anrechnen der Punkte wird zuerst auf den GameState geachtet. Besitzt dieser den Wert GAME_OVER so wird dem Spieler die Punktzahl direkt angerechnet. Besitzt er diesen nicht, so wird nur im Falle von einer Punktzahl von 9 der Punktestand des Spielers erhöht, ein Meeple zurückerstattet und die Referenz auf den Spieler entfernt.

Teil (b)

Zur Berechnung der Punktzahl einer Stadt, eines Wegs oder einer Wiese wird eine Methode genutzt, welche je nach übergebenem Landschaftstypen anders wertet. Zudem wird die Wiese anders bewertet, wenn der private Wahrheitswert officialFieldCalculation auf wahr gesetzt ist. Dann wird die Bewertung nach Aufgabe 6.3.2 aktiviert.

Innerhalb dieser Methode werden zuerst alle Knoten dieses Landschaftstypen aus dem Graphen ausgewählt. Solange diese Liste an Knoten noch nicht leer ist, müssen noch weitere Zusammenhangskomponenten bewertet werden. Ein Startknoten wird dann aus der Knotenliste entfernt und in einer Queue gespeichert. Anhand dieses Knotens wird der gesamte Subgraph einer zusammengehörigen Landschaft in einer Liste gesammelt. Dies geschieht über eine while-Schleife, solange die Queue noch Elemente besitzt. In jedem Durchlauf werden alle Kanten eines Knotens betrachtet und neu erreichte Knoten in die Queue aufgenommen, sowie aus der Knotenliste entfernt. Anhand dieser neuen Knotenliste des Teilgraphen werden die Spieler auf den Knoten und die Anzahl an Kacheln, über welche sich das Gebiet erstreckt gezählt. Außerdem wird der Graph in einer Subroutine auf Abgeschlossenheit überprüft.

Diese Subroutine heißt isNodeOpen() und testet einen Knoten. Je nach Ausrichtung des Knotens auf der Kachel, wird ein Vektor erstellt. In Richtung dieses Vektors wird dann nach einer weiteren Kachel gesucht. Liegt dort keines, so kann die Landschaft nicht abgeschlossen sein und die Methode liefert wahr zurück, woraufhin in der Bewertungsmethode vorgemerkt wird, dass das Objekt nicht fertiggestellt wurde.

Nach Durchlauf über den Teilgraphen wird der herrschende Spieler/ die Spieler auf der Landschaft ermittelt, indem der Maximalwert an Meeples auf der Landschaft ermittelt wird und am Ende zum Aufrechnen der Punktzahl alle Spieler aus der Liste herausgefiltert werden, welche diese Anzahl an Meeples auf der Landschaft besitzen.

Zuvor wird nach dem Typen der Landschaft verzweigt, um die Wertung der Landschaft zu erhalten. Ist es eine Straße, so ist die Punktzahl nur die Größe der Liste an Kacheln. Handelt es sich um eine

Wiese, so wird je nach Wert von officalFieldCalculation entweder die offizielle Wiesenbewertung in einer eigenen Methode ermittelt oder nur die Anzahl an Kacheln durch vier geteilt. Ist es ein Schloss, so wird je nachdem, ob abgeschlossen oder nicht die Anzahl an Punkten verdoppelt. Dabei wird auch die Anzahl an Wappen berücksichtigt.

Die Spieler erhalten dann nur Punkte, wenn die Landschaft abgeschlossen ist oder sich das Spiel im GAME_OVER-Zustand befindet. Ist das Spiel noch nicht beendet, so werden auch die Meeples auf den Feldern zurückgegeben.

Aufgabe 6.1.5 - Theorie

1) Der Graph ist nicht immer azyklisch.

Es kann sogar schon auf einer einzigen Kachel ein Zyklus sein. Dies ist auf Kachel B, der Kachel mit einem freistehenden Kloster, der Fall. Die Knoten vom Typ Feld: oben, rechts, unten und links sind alle durch Kanten miteinander verbunden und es ergibt sich ein Zyklus wie:

oben -> rechts -> unten -> links -> oben

Es lässt sich auch aus Wegplättchen ein Zyklus legen. Ein Kreis aus 4 bogenförmigen Wegplättchen vom Typ V würde über die linken und unteren Knoten einen Zyklus bilden.

2) Die Anzahl der Zusammenhangskomponenten wird durch die Ungleichung beschränkt.

Maximal besitzt eine Kachel 9 Knoten. Diese könnten im Fall eines Klosterplättchens mit 4 Wegen (siehe unten) dann zu 9 verschiedenen Zusammenhangskomponenten/ unzusammenhängenden Graphenteilen führen. Beim Verbinden mit weiteren Kacheln können maximal 6 weitere Zusammenhangskomponenten hinzukommen, da hier an der Verbindungsstelle die bis zu 3 Zusammenhangskomponenten erweitert werden. Somit steigt der Graph pro Kachel nicht um mehr als 9 Zusammenhangskomponenten und die Ungleichung ist erfüllt.

K gespielte Legekarten L = $\{l_1, \ldots, l_k\}$, dann gilt für die Anzahl der Zusammenhangskomponenten n im dazugehörigen Graph: n <= k * 9

3) Die Laufzeit der Methode zur Berechnung der Punkte ist durch O(|E| * |V|) beschränkt.

Zur Berechnung der Punkte aller Merkmale (Features) auf dem Spielfeld müssen alle Knoten V des Graphen überprüft werden. Schließlich kann potentiell auf jedem Knoten ein Meeple stehen und damit können Punkte für einen Spieler entstehen.

Zu jedem Knoten v muss dann für die Zusammenhangskomponente die Anzahl an Meeples der Spieler und die Anzahl an Kacheln über welche sich eine Landschaft erstreckt ermittelt werden. Dazu iteriert die Methode über alle Knoten zugehörig zu dieser Landschaft durch Betrachtung aller Kanten, welche zu den Knoten führen. Hierbei werden aus der gesamten Menge E aller Kanten diese ausgewählt, welche zum ersten Knoten aus dem Stapel gehören. Mit dieser Teilmenge an Kanten lassen sich die nächsten verbundenen Knoten herausfinden. Diese werden aus der Liste aller Knoten entfernt, wodurch sich an der Anzahl der insgesamt durchlaufenen Knoten nichts ändert. Die Laufzeit hier ist also gleich der Anzahl an Knoten |V|.

Für jeden neuen Knoten zur Landschaft werden in einem weiteren Schleifendurchlauf auch alle Kanten e durchlaufen, um die komplette Zusammenhangskomponente zu erreichen. Dies wird beschränkt durch die Gesamtanzahl aller Kanten |E| typunabhängig. Bei diesem Durchlauf wird jede Kante dieser Zusammenhangskomponente doppelt betrachtet, da dieser Graph ungerichtet ist. Die

Anzahl der Durchläufe ist trotzdem geringer als |E|, da es immer noch andere Landschaften mit Kanten gibt, die nicht in der Berechnung durchlaufen werden.

Die Berechnung des Spielers, welcher die Kontrolle über die Landschaft besitzt, und die Endabrechnung der Punkte sind von anderer Größenordnung und besitzen keinen Einfluss auf die Komplexität der Methode. So besitzt die Endberechnung nur eine Komplexität von einem Aufruf und die Spielerbetrachtung besitzt auch nur die Größenordnung der Anzahl an Spielern.

Deshalb ist die Laufzeit von calculatePoints(type, state) durch O(|E| * |V|) beschränkt!

Aufgabe 6.3 - Weitergestaltung des Spiels

Aufgabe 6.3.1 - Highscore View

Zur Umsetzung der GBO der Highscoreview in unserer Implementierung, wird in dem schon vorgefertigten Frame wird eine JTable erstellt. Beim Erstellen der Tabelle wird aus der Liste von Highscores die von jedem Score Objekt in ein 2D Array gepackt und dann an die Tabelle übergeben. Um dem Benutzer den Suchvorgang zu erleichtern, wird dieser anschließend die Möglichkeit des Sortierens (durch Klick auf den Tabellenkopf) hinzugefügt. Somit ist es möglich die einzelnen Zeilen der Tabelle entweder nach Datum, Spielername oder Punktzahl zu sortieren. Für die Spalten Name und Punktzahl ist dies relativ trivial, jedoch muss für die Datumsortierung ein eigenes Comparatorinterface implementiert werden, da diese chronologisch in eine Reihenfolge gebracht werden sollen und nicht nach Buchstaben bzw. Ziffern sortiert werden sollen. Für den Comparator müssen die einzelnen Objekte der Tabelle in Strings umgewandelt werden: Dafür ist es essentiell, dass dabei ein einheitliches Datumsformat verwendet werden sollte, sodass dieses kann dann wieder in eine Zahl umgewandelt werden kann und dann verglichen werden kann. Ähnliches passiert bei dem Vergleichen der Score Einträge.

Links: https://www.codejava.net/java-se/swing/6-techniques-for-sorting-jtable-you-should-know [Abgerufen 17.03.2020]

Aufgabe 6.3.2 - Offizielle Bewertung einer Wiese

Für die Berechnung der offiziellen Wiesenwertung, wird zunächst vor Aufruf der Methode officialFieldScore eine Menge mit allen abgeschlossenen Städten als private Variable im Gameboard gespeichert. Dies passiert durch den Aufruft von calculatePoints(type, state) mit type = CASTLE und state = GAME_OVER am Spielende. Diese Menge enthält eigens angelegte Objekte namens completedCastle. Dieses completedCastle Objekt enthält sowohl eine Liste aller der Stadt angehörigen Nodes vom Typ Castle, als auch eine ID, welche die Stadt eindeutig identifiziert. Damit soll verhindert werden, dass Städte später mehrfach gezählt werden.

Der Methode officialFieldScore wird nun eine zusammenhängende Wiese übergeben und soll für diese den Score berechnen, welchen ein Spieler bekommen würde, wenn er diese Wiese besetzt hält. Dazu werden für jeden Knoten der Wiese alle (maximal jedoch nur 2) anliegenden Knoten vom Typ Castle ermittelt. Stadt-Knoten können sich auf einem Tile nur auf Position TOP, RIGHT, BOTTOM und LEFT befinden. Befindet sich ein Wiesen-Knoten beispielsweise auf Position BOTTOM, so kommen für die Überprüfung anliegender Städte nur die Knoten auf Position LEFT und RIGHT in Frage. TOP darf für diesen Wiesen-Knoten nicht überprüft werden, da zwischen BOTTOM und TOP eine Straße verlaufen kann. Denkt man dies für alle Positionen durch, ergeben sich immer 2 mögliche Positionen für Stadt-Knoten (durchgeführt von Hilfsmethode getPossibleCastlePositions) bei denen dann geprüft werden muss, ob sich dort wirklich eine Stadt befindet (getAllNearbyCastleNodes). Nun wird von der eigentlichen Methode über alle Wiesen-Knoten

iteriert. Wird ein Stadt-Knoten gefunden, so wird überprüft ob es sich um eine abgeschlossene Stadt handelt, also ob sich ein completedCastle Objekt in der gespeicherten Menge befindet, welches diesen Stadt-Knoten enthält. Ist dem der Fall, muss zuletzt noch überprüft werden, ob diese Stadt für diese Wiese noch nicht gewertet wurde. Dazu dient eine interne Menge, die alle IDs der bereits gewerteten Städte enthält. Für jede abgeschlossene angrenzende Stadt werden 3 Punkte berechnet. Wer die Punkte bekommt ist bereits in calculatePoints realisiert.

Aufgabe 6.3.3 - Mission

Der oder die Spieler werden auf dem Spielstarten-Bildschirm darüber aufgeklärt, wie das Spiel mit Missionen funktioniert. Damit soll vermieden werden, dass sich Spieler im Spiel ärgern, weil sie nicht wissen oder vergessen haben, dass die Möglichkeit besteht durch das Erzielen einer Mission frühzeitig zu gewinnen.

Allgemein für Missionen wird beim Verarbeiten des Game-Over-States geprüft, ob eine der beiden Missionen erfüllt wurde (und diese dann zu dem Wechsel in diesen State geführt hat). Ist dies der Fall, so wird ein angepasster Text im Statusbar-Panel und auch in der Popupmessage dargestellt.

Mission 1:

Speziell für die erste Mission wird in der Methode nextRound, nachdem das TempMeepleOverlay entfernt wurde (sofern keine AI zuvor gelegt hat), auf Erreichen dieser Mission geprüft.

Die Methode zum Prüfen der Mission 1 nutzt eine Hilfmethode, um an die Anzahl an Städten pro Spieler zu gelangen. Diese Methode ist playersCastleAmounts() und unten beschrieben. Diese nutzt als Hilfmethode playersRulingCastles(), welche eine Liste zurückgibt, in welcher jeder Spieler so häufig vorkommt, wie er Städte besetzt hält. In der Methode whichPlayersRuleCastle wird dann ermittelt, welcher Spieler die Vorherrschaft in einer Stadt besitzt.

whichPlayersRuleCastle: Ist ein Teilgraph zu einer Stadt gegeben, so wird in dieser Methode errechnet, welcher oder welche Spieler die größte Anzahl an Meeples auf dieser Stadt besitzt (besitzen) und damit die Kontrolle besitzt. Dies geschieht durch Iterieren über alle Knoten der Stadt und im Falle eines gesetzten Meeples wird der Spielername mit der Stärke von einem Meeple in eine HashMap aufgenommen. Wenn ein weiterer Meeple eines Spielers hinzukommt, so wird seine Anzahl an Meeples erhöht. Zuletzt werden durch lineare Suche die Spieler mit der größten Anzahl an Meeple in eine ArrayListe geschrieben und zurückgegeben.

PlayersRulingCastles: Um Herauszufinden, wie wiele Städte einem Spieler gehören, werden alle Knoten des Graphen, welche eine Stadt darstellen nach zusammenhängenden Städten aufgeteilt. Dies geschieht ähnlich zur Punkteberechnung. In einer ArrayDeque werden die Knoten des Teilgraphen durchlaufen und die Kanten durchlaufen, um alle Knoten des gesamten Teilgraphen in einer Liste (castleList) zu sammeln. Diese Liste wird weitergegeben an die

In der Methode playersCastleAmounts(), wird dann eine HashMap erstellt, die am Ende die Player Objekte als Schlüssel und deren Anzahl an Castles als Wert zurückgegeben wird. Dazu wird zunächst über die Spieler mithilfe von Players.getPLayers() iteriert und wiederum ein Loop über die in playersRulingCastle() erstellte Liste ausgeführt. Falls nun der Name des aktuellen Spielers mit dem Namen in der Liste übereinstimmt, so wird ein Zähler jedes Mal um 1 erhöht. Dann werden das Player Objekt als Schlüssel und der Zähler als Wert der HashMap hinzugefügt. Dieser Vorgang wiederholt sich mit allen Spielern und zum Schluss der Methode wird die HashMap dann zurückgegeben.

Nachdem ermittelt wurde, welcher Spiele wie viele Städte beherrscht, kann mithilfe linearer Suche der Spieler mit den meisten Städten ermittelt werden. Besitzt dieser mehr als drei Städte und haben alle anderen Spieler mindestens drei Städte weniger als dieser Spieler unter ihrer Kontrolle, so wird der Name des Spielers zurückgegeben.

Mission 2:

Besitzt der führende Spieler einen Vorsprung von 20 oder mehr Punkten zum Nächstplatzierten, so soll dieser Spieler direkt gewinnen. Somit stellen große Bauvorhaben eines Spielers eine Gefahr für alle weiteren Spieler dar. Spieler werden versuchen große Bauvorhaben zu unterbrechen oder sich geschickt zu beteiligen.

Die Überprüfung, ob die zweite Mission abgeschlossen wurde, erfolgt nach der Berechnung der einzelnen Punktstände der einzelnen Spieler (calculatePoints()). Nach dieser Berechnung wird mit Hilfe der Methode checkMission2() ermittelt, ob und welcher Spieler die Mission vervollständigt hat: Mit einer linearen Suche wird zuerst der Spieler mit den meisten Punkten ermittelt, anschließend wird über die Liste von allen Spieler, ausgenommen dem zuvor ermittelten Spieler mit der höchsten Punktzahl, iteriert. Haben alle Spieler mehr als 20 Punkte weniger als der erste Spieler so muss dieser die Mission erfüllt haben und kann zurückgegeben werden. Andernfalls kann

Aufgabe 6.3.3 - Computergegner

Zur Umsetzung des AI Gegners, wird zunächst über das komplette Spielfeld iteriert, wobei an jeder Position eine Schleife durch die Liste der schon platzierten Tiles iteriert, um zu überprüfen ob die Koordinate überhaupt frei ist. Ist dies nicht der Fall, so wird eine Boolean Wert der zum Beginn der Methode true ist, auf false gesetzt und der Loop über die schon platzierten Tiles wird abgebrochen. Danach wird in einer if-Abfrage eben dieser Boolean Wert abgefragt. Falls die Abfrage erfolgreich ist, wird in einer weiteren if-Abfrage die zuvor implementierte Methode isTileAllowed() auf das der draw-Methode übergebene Tile angewand. Ist diese true, wird das Tile platziert und der Loop über das Spielfeld wird abgebrochen, was dann auch zum Ende der Methode führt. Andernfalls wird das Tile mit der Methode rotateRight() gedreht und es kommt zur gleichen if-Abfrage wie zuvor. Durch das wiederholen dieses Chemas wird das Tile solange gedreht bis isTileAllowed() true zurück gibt, das Tile platziert wird und der Loop mit break beendet wird. Danach endet die Methode.

Um nun einen Meeple zu setzen wird zunächst überprüft ob dies überhaupt möglich ist. Dazu wird überprüft, ob die Methode getMeepleSpots() den Wert null zurückgibt. In diesem Fall wird nextRound() aufgerufen und die Methode endet. Gibt getMeepleSpots() jedoch nicht null zurück, wird zunächst eine ArrayList des Typs Integer erstellt, welcher dann durch einen Loop und einer if-Abfrage die Indexe speichert, die im getMeeplespots() Array true zurückgeben. Somit hat man zum einen durch die Länge der ArrayList die Anzahl möglicher Spots, als auch die möglichen Positionen, da die Methode getMeepleSpots() in ihrem Array bei TOPLEFT startet und dann von links nach rechts die Positionen durchgeht und dem nach zum Beispiel der Index 4 die Position CENTER darstellt. Danach wird mithilfe der Klasse Random eine zufällige Zahl zwischen 0 und der Länge der ArrayList generiert. Diese Zahl wird dann als Index für die ArrayList genutzt welche dann wiederum einen Zahlenwert zwischen 0 bis 9 ausgibt. Danach wird dieser Wert mithilfe von if-Abfragen der dazugehörigen Position zugeordnet und mit der Methode placeMeeple() wird dann letztendlich ein Meeple gesetzt. Danach endet die Methode.