Funktionale und objektorientierte Programmierkonzepte
Wintersemester 18/19

Abgabedatum: 26.03.2019
Gesamtpunktzahl: 21 (runterskaliert von 51)

Projekt - v1.4

Game of Castles

Hauptautor: Roman Hergenreder

Mitwirkende: Philipp Imperatori, Nils Nedderhut, Louis Neumann

Organisation und Aufgaben: Florian Kadner, Lukas Röhrig

Gesamtleitung: Prof. Karsten Weihe

Inhaltsverzeichnis

1 Organisatorische Informationen

1.1 Grundlegende Informationen und Bonus

Das FOP-Projekt ist zwar nicht verpflichtend und auch für die formale Prüfungszulassung (Studienleistung) **nicht** notwendig, aber mit dem FOP-Projekt können zusätzliche Bonuspunkte erreicht werden. Voraussetzung für den Bonus ist natürlich, dass die Studienleistung erreicht ist.

Durch die Hausübungen konnten Sie 113 Punkte erreichen, wobei Sie 57 Punkte für die Studienleistung brauchen. Die anderen der 56 erreichten Punkte gehen als Bonuspunkte auf Ihr Konto ein. Im Projekt können Sie zusätzlich 51 Punkte erreichen, welche am Ende durch 2.5 geteilt und auf die nächstgrößere, ganze Zahl aufgerundet werden. Sie bekommen im Projekt also bis zu 21 Punkte für Ihr Bonuspunktkonto.

Ihre finale Anzahl an Bonuspunkten ergibt sich also aus

(Hausübungspunkte - 57) + aufrunden(Projektpunkte/2.5).

Sie werden das Abschlussprojekt in Teams von genau vier Personen bearbeiten.

1.2 Zeitplan im Überblick

- 14.02.19 - 18 Uhr: Deadline zur Anmeldung
- 18.02.19 - 01.03.19: verbindliche Teamtrainings der HDA
- 27.02.19 ab 12:00 Uhr in S105/122: Git-Workshop der Fachschaft
- 18.02.19 - 26.03.19 Poolsprechstunden
- 26.03.19 - 23:55 Uhr: Abgabe des Projekts

1.3 Poolsprechstunden

Für eventuell anfallende Fragen, bieten die Projekttutoren vom 14.02. - 26.03. Sprechstunden an. Diese Sprechstunden finden im C-Pool des Piloty Gebäudes statt. Eine Übersicht über die Sprechstundentermine finden Sie im Projektabschnitt des moodle-Kurses.

1.4 Anmeldung (bis 14.02. zu erledigen!)

Um sich anzumelden, bilden Sie Gruppen aus genau 4 Studierenden. Sollten Sie noch keine Gruppe haben, steht Ihnen im Projektabschnitt des moodle-Kurses ein Forum zur Gruppenfindung zur Verfügung.

Bei der Eintragung in die Gruppen wählen Sie gleichzeitig Ihren Termin für das Teamtraining (siehe nächster Abschnitt). Alle 4 Gruppenmitglieder müssen sich manuell in die Gruppen eintragen.

1.5 HDA-Teamtrainings

Die Teamtrainings liegen im Zeitrahmen vom 18. Februar bis zum 01. März und werden rund zwei Stunden dauern. Dort nehmen Sie mit Ihrer gesamten Gruppe und noch 3 anderen Gruppen teil. Bei Abwesenheit ist unverzüglich ein Attest bei uns einzureichen.

Achtung: Die Teilnahme am Teamtraining ist obligatorisch für den Bonus.

Fehlen Sie unentschuldigt bei Ihrem Teamtraining, erhalten Sie automatisch **keine** Punkte für das Projekt.

1.6 Abgabe des Projekts

Das Projekt ist bis zum 26.03. um 23:55 Uhr Serverzeit auf moodle abzugeben. Das Projekt exportieren Sie genauso wie die Hausübungsabgaben als ZIP-Archiv, hier gelten die gleichen Konventionen. Die Benennung erfolgt folgendermaßen: `Projektgruppe_xxx`, wobei Sie die `xxx` durch Ihre Gruppennummer ersetzen. Dabei gibt eine Person aus Ihrem Team das komplette Projekt ab.

Neben dem Code geben Sie zusätzlich eine PDF-Datei ab, diese soll sich ebenfalls in dem abzugebenden ZIP-Archiv befinden. Sie geben also am Ende nur ein ZIP-Archiv ab, das das Projekt sowie die obengenannte PDF enthält. In der PDF finden sich zum einen die Dokumentationen der weiterführenden Aufgaben und die Lösung der Theorieaufgaben.

Achtung:
Nachdem eines Ihrer Teammitglieder das Projekt in moodle hochgeladen hat, müssen alle anderen Teammitglieder diese Abgabe im entsprechenden Modul bestätigen. Andernfalls wird die Aufgabe nur im Entwurfsmodus gespeichert und nicht als Abgabe gespeichert. **Es werden keine Abgaben im Entwurfsmodus akzeptiert. Diese werden nicht bewertet und egal, was als Entwurf hochgeladen wurde, es mit 0 Punkten bewertet.** Geben Sie also nicht kurz vor Deadline ab und denken Sie daran, dass alle Teammitglieder die Abgabe bestätigen müssen!

1.7 Plagiarismus

Selbstverständlich gelten die gleichen Regelungen zum Plagiarismus wie auch bei den Hausübungsabgaben!

Insbesondere hier im speziellen nochmals der Hinweis, dass wenn Sie Repositorys zur Gruppenarbeit verwenden, diese privat sein müssen!

Beachten Sie: Sollte Ihre Dokumentation der Lösungswege unvollständig sein oder uns Anlass für den Verdacht geben, dass Sie nicht nur innerhalb der eigenen Gruppe gearbeitet haben, müssen Sie damit rechnen, dass Sie Ihre theoretischen Ergebnisse bei einem privaten Testat bei Prof. Weihe persönlich vorstellen und erläutern müssen.

2 Übersicht über das Projekt

Das Thema des Projektes ist die Strategiesimulation *Game of Castles*, welche an den Brettspielklassiker Risiko[1] angelehnt ist. Das Spiel ist für 2-4 Spieler und hat zum Ziel, alle Burgen zu erobern.

Die Simulation spielt auf einer zweidimensionalen Karte, welche neben Gras und Wäldern auch Gebirge und Wasser enthält. Auf dieser Karte sind Burgen verschiedener Königreiche verteilt und mit Pfaden untereinander verbunden.

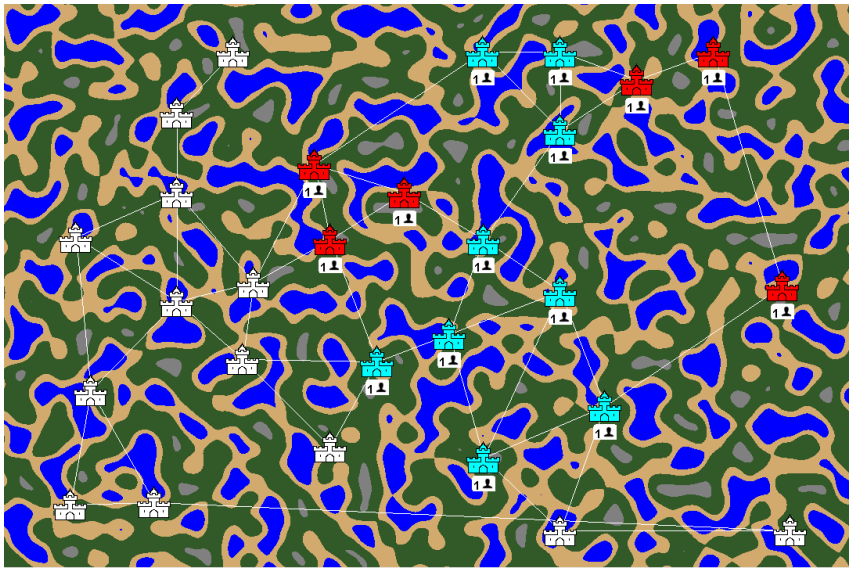
2.1 Regeln für die Spieler

Vorbereitung

Bevor Sie ein Spiel starten, müssen Sie diverse Einstellungen vornehmen. Im Spielmenü stellen Sie die Anzahl der Spieler sowie deren Namen und Farben ein. Sie haben außerdem die Möglichkeit, einem computergesteuerten Gegner (z.B. das vorimplementierte `BasicAI`) am Spiel teilhaben zu lassen. Weiterhin können Sie die Größe der Spielfeldkarte einstellen und eine Spielmission festlegen. Bei der Standardspielmission Eroberung gewinnt der Spieler, der zuerst alle Burgen erobert hat. Pro Spieler werden maximal 7 Burgen für Spielfeldgröße Klein, 14 Burgen für Mittel und 21 Burgen für Groß generiert. Haben Sie alle Einstellungen vorgenommen, kann das Spiel beginnen.

Erste Runde

Zu Beginn des Spiels werden alle verfügbaren Burgen an die Spieler verteilt. Dies geschieht reihum, wobei am Anfang zufällig bestimmt wird, welcher Spieler anfangen darf, und jeder drei Burgen auswählen kann, bevor der nächste an der Reihe ist.



Nachdem auf diese Weise alle Burgen verteilt wurden, geht das Spiel in den regulären Spielmodus über.

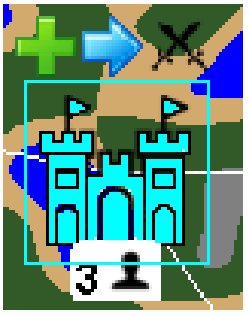
[1] `https://de.wikipedia.org/wiki/Risiko_(Spiel)`

Weitere Spielrunden

Nachdem alle Burgen aufgeteilt wurden, erhält der Spieler, der am Zug ist, Truppen abhängig von den Anzahl an Burgen, die in seinem Besitz sind. Auf 3 Burgen kommt eine Truppe und für jedes vollständig besetzte Königreich von ihm erhält der Spieler eine Truppe extra. Sollte der Spieler weniger als 9 Burgen haben, erhält er dennoch 3 Truppen.

Anschließend stehen dem Spieler folgende Aktionen zur Auswahl:

- **Truppen verteilen:** Um die zusätzlichen Truppen am Rundenstart zu verteilen, wählt man eine seiner Burgen aus. Dann erscheint ein grünes Plus über der Burg, mit dem man die Anzahl der Truppen auf dieser Burg erhöhen kann
- **Truppen verschieben:** Mit dem blauen Pfeil über der Burg kann man Truppen von einer auf eine andere Burg verschieben. Die Burgen müssen dabei über einen Pfad erreichbar sein, auf dem nur eigene besetzte Burgen liegen
- **Angreifen:** Mit dem Schwertsymbol kann man feindliche Burgen angreifen, welche direkt an die eigenen Burgen angrenzen.



Der Angriff

Entscheidet sich der aktive Spieler dafür, eine benachbarte gegenerische Burg anzugreifen, so muss er zunächst festlegen, mit wie vielen Truppen er den Angriff ausführen will. Dies geht mit mindestens 1 Truppe und mit maximal 3 Truppen. Für jede Truppe wird dabei ein Würfel genommen. Der Verteidiger kann sich nun ebenfalls entscheiden, allerdings nur ob er mit einer oder zwei Truppen verteidigt. Auch für jede verteidigende Einheit wird jeweils ein Würfel gewählt. Dann wird gewürfelt und die Augenzahlen des jeweils höchsten und zweithöchsten Wurfergebnis von Angreifer und Verteidiger gegenübergestellt. Hat der Angreifer bei diesen paarweisen Vergleichen eine höhere Zahl gewürfelt, so verliert der Verteidiger jeweils eine Einheit (bei gleicher Augenzahl gewinnt der Verteidiger). Der Angreifer selbst entscheidet, wie oft er angreifen möchte und wie viele Einheiten er dafür benutzen möchte.

Ein Beispiel: Anna hat 5 Einheiten auf ihrer Burg und greift die benachbarte Burg von Berta an, welche dort 3 Einheiten platziert hat. Anna kann sich jetzt entscheiden, mit wie vielen Würfeln sie werfen möchte. Sie entscheidet sich für drei Würfel und Berta möchte mit zwei Würfeln verteidigen. Anna würfelt nun eine 6,3 und 1. Berta würfelt 5 und 4. Die höchsten Ergebnisse werden miteinander verglichen, also die 6 von Anna und die 5 von Berta. Damit verliert Berta eine Einheit. Danach werden noch die zweithöchsten Ergebnisse, also die 3 und die 4 verglichen, somit verliert auch Anna eine Einheit. Nach diesem Angriff hat Anna also noch 4 Einheiten und Berta noch 2 Einheiten, jetzt steht es Anna zu zu entscheiden, ob sie noch einmal angreifen möchte.

In jedem Zug darf der aktive Spieler so häufig Truppen verschieben und so oft angreifen, wie er will, die Reihenfolge bleibt dabei auch ihm überlassen. Möchte er keine weitere Aktion mehr durchführen, beendet er seine Runde über den entsprechenden Button, und der nächste Spieler ist dran.

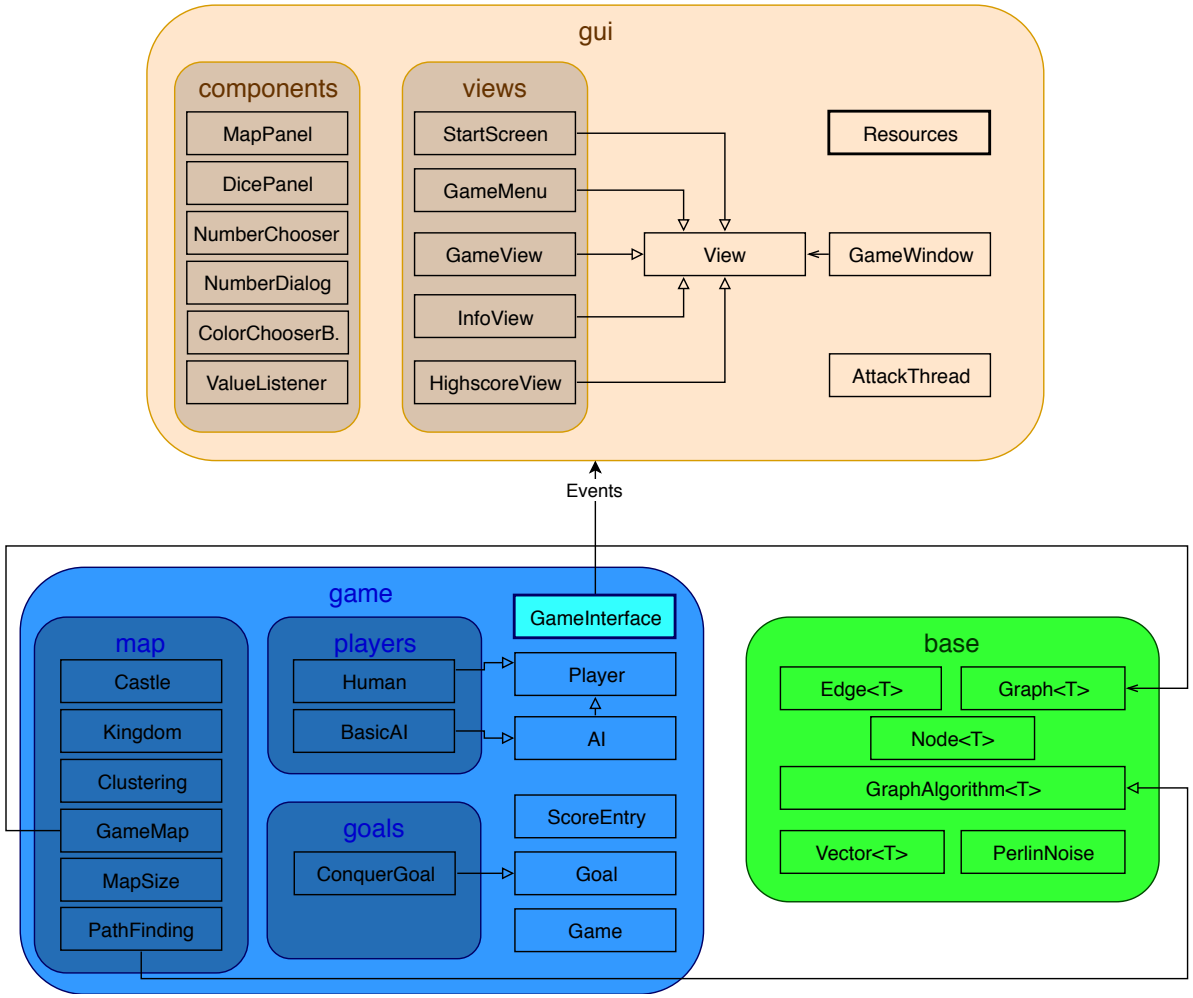
Punkte

Um den Spielfortschritt zu bewerten, werden innerhalb des Spiels Punkte verteilt. Punkte erhält man für folgende Aktionen:

- 5 Punkte für die Auswahl einer Burg in der ersten Runde
- 20 Punkte für den Sieg über eine Truppe des Verteidigers
- 30 Punkte für den Sieg über eine Truppe des Angreifers
- 50 Punkte für die Kontrollübernahme einer feindlichen Burg
- 150 Punkte für die Erfüllung des Missionsziels
- Anzahl der erhaltenden Truppen * 5 für jede neue Runde
- 10 Extrapunkte für jedes vollständig besetzte Königreich je Runde

2.2 Aufbau der Codevorlage

Machen Sie sich zunächst mit der Vorlage und den bereits angelegten Klassen und Packages vertraut. Dabei soll Ihnen das folgende Diagramm helfen.



2.3 Hinweise zur aktuellen Umsetzung der Spielmechanik

Ergänzend zu den bisher genannten Regeln hier einige Hinweise, wie das Ganze in der Vorlage umgesetzt ist:

- Man kann nur angreifen, wenn man mindestens 2 Truppen hat, und wenn, dann muss auch mindestens eine Truppe zurückbleiben, da es sonst passieren könnte, dass man bspw. beide Truppen bei einem Angriff verliert und so unbestimmt ist, wem die Angreiferburg dann gehört.

  Dies bedeutet, dass wenn man z.B. 5 Truppen beim Angriff auswählt, man maximal solange angreift, bis nur noch eine Truppe übrig ist, wenn man alle Würfelduelle verliert. Danach könnte man nicht mehr angreifen, weil ja mindestens eine Truppe zurück bleiben muss. Pro Angriff wird dann aber nur mit maximal 3 Truppen angegriffen (entspricht den 3 Würfeln). In der Vorlage wird also bei Zahlen größer als 3 automatisch immer mit 3 und am Ende einmal mit 1 oder 2 Truppe angegriffen.

- Der Verteidiger hat zwar eigentlich die freie Wahl mit wie vielen Würfeln er verteidigen möchte, allerdings wird in der Vorlage automatisch immer eine Verteidigung mit 2 Würfel gewählt (da alles andere taktisch unsinnig wäre).

- In der Vorlage werden immer maximal 3 Königreiche generiert. Das kann beliebig abgeändert werden, wie viele maximal generiert werden sollen.

Allgemeiner Hinweis:

Das ist IHR Spiel.

Solange die Anforderungen der Aufgabenteile eingehalten werden UND das Grundgerüst des Risiko Spiels nicht verändert wird, *feel free to modify*. Sie können das Spiel beliebig erweitern, verändern und umschreiben. Überraschen Sie uns. Am sinnvollsten halten Sie größere Änderungen aber immer in Ihrer PDF fest.

3 Aufgaben

Das Projekt besteht aus drei verschiedenen Aufgabentypen - Basisaufgaben, weiterführende Aufgaben und Theorieaufgaben. In den Basisaufgaben geben wir Ihnen genau vor, welche Funktionalität Sie zu implementieren haben. In den weiterführenden Aufgaben erweitern Sie das Projekt nach Ihren Vorstellungen.

3.1 Der Graph (29 Punkte)

Das Grundgerüst der Karte ist ein generischer Graph als Listen von Knoten und Kanten, wie Sie ihn auch schon in den Übungen 10 und 12 kennengelernt haben.

3.1.1 Grundfunktionen und Streams — 3 Punkte - Basis

Damit die Karte richtig generiert und angezeigt wird, müssen Sie einige Funktionen ergänzen. Ergänzen Sie die mit `TODO` gekennzeichneten Methoden in der Klasse `base.Graph`

- `getAllValues()`: gibt alle Werte der Knoten in einer Liste zurück
- `getEdges(Node<T> node)`: gibt alle Kanten eines Knotens als Liste zurück
- `getNode(T value)`: gibt den ersten Knoten mit dem angegebenen Wert zurück oder null, falls dieser nicht gefunden wurde

Verbindliche Anforderung: Die Bodies der drei oben genannten Methoden dürfen jeweils nur aus einer Anweisung bestehen. Sie müssen deshalb Java Streams verwenden.

3.1.2 Kanten bilden und überprüfen — 5 Punkte - Weiterführend

Für einen fehlerfreien Spielablauf muss garantiert sein, dass nach der Kartengenerierung auch alle Burgen erreichbar sind. Dies bedeutet konkret, dass von jedem beliebigen Startknoten ein Weg zu jedem anderen Knoten besteht.

Ergänzen Sie die Methode `generateEdges()` in der Klasse `GameMap`, die die Knoten miteinander verbindet. Die Klasse enthält ein Attribut `castleGraph` vom Typ `Graph<Castle>`, über das Sie mittels `getNodes()` alle Knoten abfragen können. Mit der Methode `addEdge(Node<Castle> n1, Node<Castle> n2)` können Sie eine neue Kanten zwischen den beiden Knoten `n1` und `n2` im Graph hinzufügen. Verbinden Sie die Knoten so untereinander, dass alle Burgen erreichbar sind, aber sich gleichzeitig auch ein „schönes“ Bild ergibt (vergleichen Sie dazu auch die Abbildung im Abschnitt 2.1). Das bedeutet: Verbinden Sie die Knoten nicht einfach blind und wild miteinander, sondern überlegen Sie sich, wie Sie möglichst wenig Kanten verwenden können, um die Anforderung einzuhalten und sich möglichst wenig Kanten überschneiden.

Ergänzen Sie die Methode `allNodesConnected()` in der Klasse `base.Graph`, die überprüft, ob alle Knoten erreichbar sind. Erstellen Sie außerdem zwei JUnit-Tests in der Klasse `GraphConnectionTest`, die das Verhalten mit einem korrekten und inkorrekten Graphenbeispiel testet.

Dokumentieren Sie in Ihrer PDF, welches Vorgehen Sie für die Kantengenerierung und die Überprüfung gewählt, und wieso Sie sich dafür entschieden haben.

3.1.3 Iteratoren und Wege finden I — 5 Punkte - Basis

Um Wege zwischen Knoten zu finden, zum Beispiel um Truppen zwischen Burgen zu verschieben, verwenden wir einen speziellen Algorithmus, der ausgehend von einem Startknoten die kürzesten Wege zu allen anderen Knoten bestimmt.

Betrachten Sie die Klasse `base.GraphAlgorithm`. Diese Klasse enthält eine innere Klasse `AlgorithmNode` um jedem Knoten einen Wert (`value`) und einen Vorgängerknoten (`previous`) zuzuordnen. Die Zuordnung von `Node` und `AlgorithmNode` werden in der HashMap `algorithmNodes` gespeichert.

Ergänzen Sie die Methoden

- `getSmallestNode()`: gibt den Knoten mit dem geringsten Wert aus der Liste `availableNodes` zurück und entfernt ihn. Dabei sollen nur Knoten mit nichtnegativen Werten berücksichtigt werden. Besitzen alle Knoten den gleichen Wert, wird der erste aus der Liste gewählt und sollte die Liste leer sein, wird `null` zurückgegeben.

  Verbindliche Anforderung: Zum Durchlauf durch die Liste verwenden Sie das Interface `java.util.Iterator`.

- `run()`: startet den Algorithmus. Dieser funktioniert wie folgt:

  1. Suche den Knoten mit dem geringsten Wert v

  2. Für jeden benachbarten Knoten n von v, der über eine direkte Kante e passierbar ist: Berechne den neuen Wert a von n, indem Du das Kantengewicht von e auf den Knotenwert von v addierst. Sollte der momentane Wert von n nicht gesetzt (d.h. Wert -1) oder a kleiner sein, dann wird der Wert von n auf a und der Vorgängerknoten von n auf v gesetzt.

  3. Wiederhole solange, bis alle Knoten abgearbeitet wurden

- `getPath(Node<T> destination)`: gibt eine Liste von Kanten zurück, die einen Pfad zu dem angegebenen Zielknoten repräsentiert. Dabei werden zuerst, beginnend mit dem Zielknoten, alle Kanten mithilfe des Vorgängerattributs zu der Liste hinzugefügt. Zum Schluss muss die Liste nur noch umgedreht werden. Sollte kein Pfad existieren, geben Sie `null` zurück.

3.1.4 Wege finden II — 7 Punkte - Theorie

In der vorherigen Aufgabe haben Sie den Algorithmus in der Methode `run` umgesetzt, der die Distanzen von einem Startknoten zu allen anderen Knoten berechnet. Dafür haben Sie alle Knoten aus einem Attribut vom Typ `List<Node<T>>` abgearbeitet und die Werte der Knoten entsprechend angepasst. Bearbeiten Sie die beiden nachfolgenden Aufgaben in Ihrer Dokumentations-PDF.

Teil (a) — 2 Punkte

Geben Sie die Worst-Case Komplexität des Algorithmus in Groß-O Notation an und begründen Sie Ihre Antwort.

Teil (b) 2 Punkte

Welchen Typ könnten Sie anstelle von `List<Node<T>>` verwenden, um die Sammlung der Knoten zu organisieren? Wählen Sie einen Typ, der die Laufzeit des Algorithmus effizienter gestaltet und geben Sie eine allgemeine Form der Komplexität aus (a) an, welcher allgemein auf alle Typen übertragbar ist.

Teil (c) 3 Punkte

Geben Sie für den Algorithmus die Invariante in der Ihnen bekannten Form „Nach $h \geq 0$ Durchläufen gilt:“ an. Ihre Formulierung der Invariante beschreibt dabei

1. den Zustand der Liste der noch nicht abgearbeiteten Knoten
2. den Zustand der Knoten, welche noch nicht abgearbeitet wurden, sowie
3. den Zustand der Knoten, welche abgearbeitet wurden.

3.1.5 Kürzester Pfad zu allen Knoten 5 Punkte - Theorie

In dieser Aufgabe wollen wir nun abstrakt einen Algorithmus betrachten, welcher auf einem **gerichteten** Graphen den kürzesten Pfad von **jedem** Knoten zu **jedem** anderen Knoten bestimmt. Für einen Graphen mit Knotenmenge V und Kantenmenge E richtet der Algorithmus dafür eine $(n \times n)$-Matrix ein, wobei $n = |V|$ gilt. Der Eintrag an der i-ten Zeile und der j-ten Spalte liefert am Ende die kürzeste Distanz zwischen dem i-ten und dem j-ten Knoten zurück. Mit M^h bezeichnen wir die Distanzmatrix nach h-Durchläufen.

Gegeben seien nun die folgenden Informationen:

Schleifeninvariante: Nach $h \geq 0$ Durchläufen enthält $M^h(v, w)$ die Länge des kürzesten Pfades von v nach w mit maximal $h + 1$-Kanten, wobei $v, w \in V$.

Schleifenvariante: h erhöht sich um 1.

Schleifenabbruch: $h = n - 1$.

Teil (a) 2 Punkte

Als negativen Zykel bezeichnen wir einen Zykel, bei dem die Summe aller Kantengewichte negativ ist. Zeigen Sie basierend auf der Schleifeninvariante, dass der Algorithmus nicht korrekt arbeitet, wenn ein negativer Zykel im Graph vorkommt.

Teil (b) 3 Punkte

Geben Sie die asymptotische Komplexität des Algorithmus im Best- und Worst-Case als Θ-Ausdruck an und begründen Sie Ihre Antwort!

Hinweis:

Sie haben hier keinerlei Informationen über die interne Repräsentation des Graphens, diese brauchen Sie auch nicht, Sie haben die abstrakte Beschreibung des Algorithmus. Dieser funktioniert **nur über das Updaten der Matrix**. Überlegen Sie sich, wie Sie den Algorithmus nur mithilfe der abstrakten Informationen betrachten können ohne Annahmen über den Graphen treffen zu müssen.

3.1.6 Königreiche generieren 4 Punkte - Basis

Um die verschiedenen Königreiche zu bilden, müssen wir Gruppen der Knoten bilden. Dafür ergänzen Sie die Methode `List<Kingdom> getPointsClusters()` in der Klasse `Clustering`. Die Methode soll eine Liste von Königreichen zurückliefern, welche wir auf eine spezielle Art und Weise bilden wollen. Jedes Königreich wird dabei durch ein eigenes Zentrum, eine Liste von dazugehörigen Burgen und eine ID zwischen 0 und 5 definiert. Entsprechende Attribute finden Sie in der Klasse `Kingdom`. Wir wollen die Burgen auf der Karte nun zu den Königreichen zuordnen, indem wir die Summe der quadrierten Abweichungen von den Zentren der Königreiche minimieren. Mathematisch entspricht dies dem folgenden Optimierungsproblem, wobei wir k Königreiche K haben mit den Zentren μ und den Burgenpositionen x.

$$\arg\min \sum_{i=1}^{k} \sum_{x_j \in K_i} \|x_j - \mu_i\|^2$$

Gehen Sie nun in folgenden Schritten vor:

1. Wählen Sie zuerst die k Zentren der Königreiche auf der Karte zufällig.
2. Ordnen Sie jeder Burg jeweils ein Königreich zu. Dafür wählen Sie dasjenige Königreich, dessen Zentrum die geringste euklidische Distanz zur Burg aufweist.
3. Setzen Sie nun die Zentren der Königreiche neu. Dafür bilden Sie den Mittelwert aus allen x- und y-Koordinaten der Burgpositionen aus dem jeweiligen Königreich.
4. Wiederholen Sie die Schritte 2. und 3. nun solange, bis sich die Zuordnungen nicht mehr ändern.

Sie dürfen neue Klassen, Attribute und Methoden anlegen, um diesen Algorithmus umzusetzen. Am Ende liefert die Methode die fertig partitionierte Liste von Königreichen, wobei jedes Königreich sein eigenes Zentrum und die enthaltenen Burgen enthält.

3.2 Highscore und Dateien (5 Punkte)

In dieser Aufgabe behandeln Sie den Umgang mit Dateien. Betrachten sie dazu die Klassen `gui.Resources` und `game.ScoreEntry`. Die Klasse `Resources` verwaltet dabei unter anderem Bilder, Icons und Schriftarten. Wir wollen nun, dass unsere letzten Spielergebnisse in einer Datei gespeichert werden und wieder ausgelesen werden können. Diese sollen dann im GUI als Highscore-Seite sortiert angezeigt werden.

3.2.1 Strings und PrintWriter 2 Punkte

Ergänzen Sie in der Klasse `ScoreEntry` die Methoden

- `read(String line)`: Liest eine gegebene Zeile ein und wandelt dies in ein `ScoreEntry`-Objekt um. Ist das Format der Zeile ungültig oder enthält es ungültige Daten, wird `null` zurückgegeben. Eine gültige Zeile enthält in der Reihenfolge durch Semikolon getrennt: den Namen, das Datum als Unix-Timestamp (in Millisekunden), die erreichte Punktzahl und den Spieltypen. Gültig wäre beispielsweise: `"Florian;1546947397000;100;Eroberung"`

- `write(PrintWriter printWriter)`: Schreibt den Eintrag als neue Zeile mit dem gegebenen `PrintWriter`. Der Eintrag sollte im richtigen Format (siehe Aufgabe davor) gespeichert werden.

3.2.2 File Streams **2 Punkte**

Ergänzen Sie in der Klasse `Resources` die Methoden

- `loadScoreEntries()`: lädt den Highscore-Table aus der Datei `"highscores.txt"`. Dabei wird die Liste `scoreEntries` neu erzeugt und befüllt. Beachten Sie dabei, dass die Liste nach dem Einlesen absteigend nach den Punktzahlen sortiert sein muss. Sollte eine Exception auftreten, kann diese ausgegeben, aber nicht weitergegeben werden, da sonst das Laden der restlichen Resourcen abgebrochen wird.
- `saveScoreEntries()`: speichert alle Objekte des Typs `ScoreEntry` in der Textdatei `"highscores.txt"`. Jede Zeile stellt dabei einen `ScoreEntry` dar. Sollten Probleme auftreten, muss eine `IOException` geworfen werden. Die Einträge sind in der Liste `scoreEntries` zu finden.

3.2.3 Sortiertes Einfügen **1 Punkt**

Ergänzen Sie in der Klasse `Resources` die Methode

- `addScoreEntry(ScoreEntry scoreEntry)`: fügt ein `ScoreEntry`-Objekt der Liste von Einträgen hinzu. Beachten Sie, dass nach dem Einfügen die Liste nach den Punktzahlen absteigend sortiert bleiben muss.

3.3 Weitergestaltung des Spiels (17 Punkte)

In den folgenden Aufgaben entwickeln Sie das Spiel weiter und gestalten es nach Ihren Vorstellungen. Dies soll fundiert auf aktuellen wissenschaftlichen Ergebnissen aus Teilbereichen der Informatik, Psychologie und Wirtschaft geschehen (Stichworte: Usability, Design). Sie finden dazu in moodle einen kleinen Leitfaden zum Thema Benutzerfreundlichkeit. Gehen Sie die Punkte im Leitfaden durch und beachten Sie diese bei der Gestaltung der Oberfläche. Dokumentieren Sie in Ihrer PDF am Ende alle Erweiterungen und Änderungen an der Benutzeroberfläche und begründen Sie diese Änderungen hinsichtlich der Benutzerfreundlichkeit. Die Dokumentation sollte für die Tutoren, also externe Leser, leicht nachvollziehbar und verständlich sein. Kennzeichnen Sie in der PDF eindeutig die Aufgabennummer.

Sie können zusätzlich dazu auch eigene Literatur und Best-Practice-Beispiele suchen und einbringen, dokumentieren Sie diese Recherche und Ihre Ergebnisse ebenfalls. Bringen Sie das Wissen und Ihre Expertise aus Ihrem persönlichen Studiengang ein, um in Ihrer 4-er Gruppe ein möglichst stimmiges Gesamtergebnis zu erzeugen.

Denken Sie daran, dass die Tutoren am Anfang nicht mit Ihren Änderungen am Spiel vertraut sind und diese möglichst intuitiv sein sollten, so wie es auch in der Praxis durchaus üblich ist.

Hinweis: Aufgabe 3.3.4 muss von allen Gruppen bearbeitet werden, die mindestens 1 Mitglied haben, welches CE studiert. Dafür entfallen dann die Aufgaben 3.3.2 und 3.3.3,

welche in der Summe die gleiche Punktzahl liefern. Sollten Sie eine Gruppe ohne CE-Studierenden sein, so haben Sie die Wahl, ob Sie Aufgabe 3.3.4 oder die Aufgaben 3.3.2 + 3.3.3 bearbeiten.

3.3.1 Computergegner **7 Punkte**

Es gibt bereits eine rudimentäre Umsetzung eines Computergegners in der Klasse `BasicAI`, welche Sie bei Spielstart im Setup als Gegner auswählen können. Ihre Aufgabe ist es nun, noch mindestens eine weitere Umsetzung eines Computergegner zu implementieren.

Hierbei handelt es sich um eine offene Aufgabe, Ihrer Kreativität sind keine Grenzen gesetzt. Implementieren Sie diese neuen Computergegner in eigenen Klassen, sodass diese auch im Menü vor dem Spielstart auswählbar sind.

Recherchieren Sie im Internet nach Möglichkeiten, den Computergegner möglichst stark zu gestalten. Dokumentieren Sie den theoretischen Background in Ihrer PDF, die Sie am Ende mit abgeben. Dort soll detailliert festgehalten werden, worauf Ihre Implementierung basiert und wie Sie beim Umsetzen vorgegangen sind. Sie dürfen auch mehrere, unterschiedliche Ansätze implementieren.

Ihre Punktzahl in dieser Aufgabe hängt maßgeblich davon ab, wie komplex und intelligent Ihre Lösung arbeitet. Die nachvollziehbare Dokumentation, sowie der recherchierte theoretische Background fließen ebenfalls in die Bewertung mit ein, genauso die Umsetzung der Integration der Computergegner in das Spiel selbst.

Mindestanforderung: Ihre selbst erstellen Computergegner müssen die bereits vorgegebene Implementierung in 4 von 5 Spielen in der Mission *Eroberung* schlagen.

3.3.2 Neue Missionen **5 Punkte**

Standardmäßig gibt es im Spiel momentan die Mission Eroberung, bei der es darum geht, als erster Spieler alle Burgen zu besetzen. Erweitern Sie das Ganze um drei weitere Missionen, sodass diese im Menü vor dem Spielen auswählbar sind! Dokumentieren Sie Ihre Überlegungen und Änderungen in der abzugebenden PDF.

3.3.3 Joker **5 Punkte**

Bauen Sie Joker im Spiel ein. Diese müssen sinnvoll über das GUI auswählbar sein und dem aktuellen Spieler einen Vorteil bringen oder einen Gegner schwächen. Halten Sie Ihre Überlegungen und Regeln in der abzugebenden PDF fest, auch hier ist Ihrer Kreativität keine Grenze gesetzt. Ihre Punktzahl hängt maßgeblich davon ab, wie gut sich die Joker in das Spiel integrieren lassen. Das bedeutet konkret, dass Sie in der GUI sinnvoll auswählbar sein sollen und das Spiel spannender, aber nicht unfairer werden lassen.

3.3.4 Anpassung für Studierende im Studiengang CE **10 Punkte**

Gemäß den Vorgaben der Akkreditierung des Studiengangs Computational Engineering muss das im Rahmen von FOP absolvierte FOP-Projekt ebenfalls einen CE-Anteil besitzen. **Dies bedeutet, dass CE-Studierende - egal, ob sie sich in einer reinen CE-Gruppe oder mit anderen Studierenden zusammen in gemischten Gruppe**

befinden - die folgende Aufgabe absolvieren müssen. Beachten Sie dafür nochmals die Hinweise am Anfang des Kapitels 3.3.

In dieser Aufgabe wollen wir einen Würfelwurf möglichst realistisch in 3D modellieren. Gegeben ist dafür das Package `dice3d`. Sie dürfen die Klassen dabei beliebig erweitern. Am Ende der Aufgabe soll nach Ausführen der Klasse `MainWindow` ein Würfelwurf zu sehen sein.

Die Klasse `Vertex` ist bereits gegeben und stellt einen Punkt im dreidimensionalen Raum dar. Passen Sie die `update`-Methode dahingehend an, dass die Position des Punktes pro Zeitschritt richtig gesetzt werden. Sie können dabei die Verlet-Integration verwenden, welche ein numerisches Lösungsverfahren für Newton'sche Bewegungsgleichungen darstellt. Bezeichnen wie gewöhnlich x die Position, t die Zeit, v die Geschwindigkeit und a die Beschleunigung, dann sagt uns die Verlet-Integration in ihrer Grundaussage das folgende:

Für die Differentialgleichung zweiter Ordnung $\ddot{\vec{x}}(t) = \vec{a}(\vec{x}(t))$ mit Startbedingungen $\vec{x}(t_0) = \vec{x}_0$ und $\dot{\vec{x}}(t_0) = \vec{v}_0$ finden wir eine numerische Approximation $\vec{x}_n \approx \vec{x}(t_n)$ an den Zeitpunkten $t_n = t_0 + n\Delta t$ mit Schrittgröße $\Delta t > 0$ über folgende Zusammenhänge:

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}(\vec{x}_n)\Delta t^2 \quad \text{und} \quad \vec{x_1} = \vec{x}_0 + \vec{v}_0\Delta t + \frac{1}{2}\vec{a}(\vec{x}_0)\Delta t^2$$

Beachten Sie außerdem, dass Sie je nach Implementation die `reset`-Methode gegebenenfalls anpassen müssen. Diese dient dazu, den Würfelwurf im Fenster erneut auszuführen.

Die Klasse `Edge` verbindet zwei Punkte miteinander. Beim Würfelwurf und der Anwendung der Verlet-Integration ändern sich die Längen der Kanten des Würfels, dadurch sieht er deformiert aus. Passen Sie die Klasse und insbesondere die `update`-Methode dahingehend an, dass der Abstand zwischen den beiden Punkten einer Kante immer konstant bleibt.

In der Klasse `Cuboid`, welche einen Quader repräsentiert, ergänzen Sie die Methode `notMoving`, die genau dann `true` zurückgibt, sofern sich der Quader nicht mehr bewegt. Außerdem ergänzen Sie in der Klasse die `updateCollision`-Methode. Diese bekommt einen anderen Quader übergeben und setzt den Wert des Attributes `collided` auf `true`, genau dann, wenn die beiden Quader miteinander kollidieren. Andernfalls setzt die Methode das Attribut auf `false`.

Die Klasse `Dice` stellt letztendlich unseren Würfel dar. Ergänzen Sie dort die Methode `getNumberRolled`, welche die Zahl zurückliefert, welche der Augenzahl der obersten Seite des Würfels entspricht (also die Augenzahl, die gewürfelt wurde).

Im Konstruktor der Klasse `World` finden Sie die Methode `run` einer `TimerTask`. Diese Methode wird in einem bestimmten Zeitintervall aufgerufen und ruft dabei die `update`-Methode eines jeden `Cuboid`s, der sich in der Welt befindet, auf. Achten Sie darauf, dass Sie diese Methode eventuell entsprechend an Ihre Implementierung anpassen müssen.

In der Vorlage sehen Sie bereits einen Untergrund gegeben. Achten Sie in Ihrer Umsetzung darauf, dass die Kollision des Würfels mit diesem Untergrund möglichst realistisch ist und er nach einiger Zeit zum Erliegen kommt.

Dokumentieren Sie all Ihre Überlegungen und geben Sie die Quellen Ihrer Recherche an. Sie sind völlig frei in der Umsetzung und sollen eine möglichst optisch ansprechende Umsetzung finden.

Abschließend integrieren Sie Ihren Würfelwurf in das Spiel, sodass dieser anstelle der normalen, vorgegebenen Würfelsimulation ausgeführt wird. Nutzen Sie auch hier nochmal den hochgeladenen Benutzerleitfaden, um die Simulation möglichst optisch ansprechend einzubinden.