1.1

Methode “getHexGrid”

Diese Methode gibt den Wert der Konstante “hexGrid“ zurück.

Parameterlose

Funktionalität:
Einfache Rückgabe des Wertes der Konstante “hexGrid“ mit “return“

Methode “getName”

Diese Methode gibt den Wert der Konstante “name“ zurück.

Parameterlose

Funktionalität:
Einfache Rückgabe des Wertes der Konstante “name“ mit “return“.

Methode “getID”

Diese Methode gibt den Wert der Konstante “id“ zurück.

Parameterlose

Funktionalität:
Einfache Rückgabe des Wertes der Konstante “id“ mit “return“.

Methode “getColor”

Diese Methode gibt den Wert der Konstante “color“ zurück.

Parameterlose

Funktionalität:
Einfache Rückgabe des Wertes der Konstante “color“ mit “return“.

Methode “isAi”

Diese Methode prüft, ob dem Spieler ein AiController zugewisen ist.

Parameterlose

Funktionalität:
1. Prüfen, ob dem Spieler ein AiController zugewisen ist.

2. Falls dem Spieler ein AiController nicht zugewiesen ist, geben “false“ zurück.

3. Falls dem Spieler ein AiController zugewiesen ist, geben “true” zurück.

1.2
Methode “getCredits”

Diese Methode gibt den Wert der Konstante “credits“ zurück.

Parameterlose

Funktionalität:
Einfache Rückgabe des Wertes der Konstante “credits“ mit “return“.

Methode “addCredits”

Diese Methode soll dem Spieler die übergebene Anzahl an Credits hinzuaddieren.

Parameter:

Anzahl der Credits, die zur aktuellen Anzahl der Credits des Spielers hinzugefügt werden sollen.

Funktionalität:
Einfache Addition der aktuellen Anzahl der Credits des Spielers mit der gewünschten Anzahl der Credits, die hinzugefügt werden sollen.

Methode “removeCredits”

Diese Methode soll dem Spieler die übergebene Anzahl an Credits abziehen. Die Methode gibt true zurück, wenn die übergebene Anzahl an Credits positiv ist und wenn der Spieler genug Credits zum Abziehen hat, und false in allen anderen Fällen.

Parameter:

Anzahl der Credits, die zur aktuellen Anzahl der Credits des Spielers abgezogen werden sollen.

Funktionalität:

1. Prüfen, ob das Spieler genuge Anzahl der Credits hat, damit man Subtrahieren machen kann. Außerdem prüfen, ob Anzahl der Credits, die zur aktuellen Anzahl der Credits des Spielers abgezogen werden sollen, nicht negativ ist

2. Falls beide Bedingungen erfüllt sind, machen einfaches Subtrahieren der aktuellen Anzahl der Credits des Spielers mit der gewünschten Anzahl der Credits, die hinzugefügt werden sollen, und geben den Wert “true“ zurück.

3. Falls nicht alle Bedingungen erfüllt sind, machen keines Subtrahieren der aktuellen Anzahl der Credits des Spielers mit der gewünschten Anzahl der Credits, die hinzugefügt werden sollen, und geben den Wert “falls“ zurück.

1.3

Methode “getRails“

Diese Methode soll eine unveränderbare Sicht auf die Schienen des im Parameter spezifizierten Spielers zurückgeben.

Parameter:
Der Spieler, dessen Schienen zurückgebracht werden müssen.

Funktionalität:

1. Bekommen ein “hexGrid“ des Spielers mit dem Aufruf der Methode “getHexGrid“.

2. Erstellen ein neues “HashMap“ mit der Name “rails“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Schienen besitzt, und Value ein Edge ist, das eine Kante besitzt, wo Schiene liegt. Dieses “HashMap“ wird benutzt, um Schienen des Spielers zu speichern.

3. Erstellen ein neues “HashMap“ mit der Name “edges“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition der Kante besitzt, und Value ein Edge ist, das eine Kante besitzt.

4. Erstes “for-Schleifen“ wird benutzt, um aus “HashMap edges“ alle “Key-Value“-Paare zu bekommen.

5. Zweites “for-Schleifen“ wird benutzt, um durch alle “Keys“ (TilePositions) des “HashMap edges“ zu überqueren.

6. Überqueren alle “Values” (Kanten).

7. Prüfen, ob die Kante eine Schiene enthält.

8. Falls diese Bedingung erfüllt ist, speichern diese Kante und ihre TilePosition in “HashMap rails“.

9. Wandeln “HashMap rails“ in einer unveränderbaren Sicht um und geben sie zurück, die alle Schienen des Spielers besitzt.

Methode “connectsTo”

Diese Methode erhält als Parameter eine andere Edge. Die Methode gibt genau dann true zurück, wenn die beiden Edges ein gemeinsames Tile besitzen.

Parameter:

Eine andere Edge

Funktionalität:

1. Prüfen, ob eine der Positionen der beiden Edges übereinstimmt.

2. Falls die Bedingung erfüllt ist, geben den Wert “true“ zurück.

3. Falls die Bedingung erfüllt ist, geben den Wert “false“ zurück.

Methode “getConnectedEdges”

Diese Methode gibt alle Kanten zurück, die an den beiden Tiles, die die Edge verbindet, anliegen.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashSet” vom Typ “Edge“ und mit der Name “connectedEdges“, um verbundene Edges zu speichern.

2. Erstellen ein neues “HashMap“ mit der Name “allEdges“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Edge besitzt, und Value ein Edge ist, das eine Kante besitzt. “HashMap“ besitzt alle Edges des HexGrides.

3. “For-Schleifen“ wird benutzt, um um durch alle “Values“ (Edges) des “HashMap allEdges“ zu überqueren.

4. Prüfen, ob “diese” Edge mit anderer Edge verbunden ist.

5. Falls die Bedingung erfüllt ist, speichern Edge in “HashSet connectedEdges“.

6. Geben “HashSet connectedEdges“ zurück, das alle verbundene Edges besitzt.

Methode “getConnectedRails”

Diese Methode gibt alle Schienen zurück, die an den beiden Tiles, die die Edge verbindet, anliegen und zum gegebenen Spieler gehören.

Parameter:

Der Spieler, dessen verbundene Schienen zurückgebracht werden müssen.

Funktionalität:

1. Erstellen ein neues “HashSet” vom Typ “Edge“ und mit der Name “connectedRails“, um verbundene Schienen zu speichern.

2. Erstellen ein neues “HashMap“ mit der Name “rails“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Schienen besitzt, und Value ein Edge ist, das eine Kante besitzt, wo eine Schiene ist.. “HashMap“ besitzt alle Schienen des Spielers.

3. “For-Schleifen“ wird benutzt, um um durch alle “Values“ (Edges) des “HashMap rails“ zu überqueren.

4. Prüfen, ob “diese” Edge mit anderer Edge verbunden ist.

5. Falls die Bedingung erfüllt ist, speichern Edge, wo eine Schiene ist, in “HashSet connectedRails“.

6. Geben “HashSet connectedRails“ zurück, das alle verbundene Schienen besitzt.

Methode “addRail”

Diese Methode erhält als Parameter einen Player, für den eine neue Schiene auf der Kante hinzugefügt werden soll. Die Schiene darf nur hinzugefügt werden, wenn der Spieler an dieser Stelle noch keine Schiene gebaut hat. Außerdem muss die Schiene, sollte der Spieler noch keine andere Schiene gebaut haben, an einer der Startstädte liegen. Sollte der Spieler jedoch schon mindestens eine andere Schiene gebaut haben, darf die neue Schiene nur gebaut werden, wenn sie an das bereits vorhandene Schienennetz des Spielers anliegt. Zuletzt gibt die Methode true zurück, wenn die Schiene erfolgreich hinzugefügt wurde und false in allen anderen Fällen.

Parameter:

Der Spieler, für den eine neue Schiene auf der Kante hinzugefügt werden soll.

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “rails“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Schienen besitzt, und Value ein Edge ist, das eine Kante besitzt, wo eine Schiene ist. “HashMap“ besitzt alle Schienen des Spielers.

2. Erstellen eine bool’esche Variable “added“, die entspricht, ob eine Schiene hinzugefügt wurde oder nicht.

3. Prüfen, ob Spieler Schienen hat.

4. Prüfen, ob “diese” Edge eine Schiene hat.

5. Falls die Bedingung nicht erfüllt ist, dann prüfen, ob es eine Stadt in einem des beiden Teil, das “diese“ Edge besitzt, gibt.

6. Falls die Bedingung erfüllt ist, dann prüfen, ob eine Stadt in einem des beiden Teil, das “diese“ Edge besitzt, eine Startstadt ist. Falls diese Bedingung erfüllt ist, dann zuweisen “added” den Wert “true“, anderfalls – “false“.

7. Falls die Bedingung (Punkt #5) erfüllt ist, erstellen ein neues “HashSet“ vom Typ “Edge“, das alle verbundene Schienen des Spielers besitzt.

8. Überqueren durch alle verbundene Schienen und prüfen, ob diese Edge mit einer der Edges von verbundenen Schienen verbindet.

9. Falls diese Bedingung erfüllt ist, dann zuweisen “added“ den Wert “true“.

10. Geben den Wert von “added“ zurück.

1.4

Methode “getNeighbour”

Diese Methode soll das Tile zurückgeben, welches in dieser Richtung an das Tile grenzt.

Parameter:

Eine EdgeDirection.

Funktionalität:

1. Prüfen die Richtung der Edge.

2. Geben den Tile an einer Position zurück, das den Richtungskoordinaten entspricht.

Methode “getEdge”

Diese Methode soll die Edge zurückgeben, welche in dieser Richtung das Tile mit seinem Nachbarn verbindet.

Parameter:

Eine EdgeDirection.

Funktionalität:

1. Erstellen einen Nachbar in einer gegebenen Richtung durch den Aufrud der Methode “getNeighbour“.

2. Prüfen, ob ein Nachbar existiert. Falls die Bedingung nicht erfüllt ist, geben den Wert “null“ zurück.

3. Falls die Bedingung erfüllt ist, geben eine Edge zurück. Es passiert durch die Aufrufe der verschidenen Methoden. Zuesrt rufen die Methode “getHexGrid” auf, um “HexGrid” zu bekommen. Dann rufen eine Methode “getEdge“ für früher aufgerufende “HexGrid” auf, um eine Edge zwischen “diese Tile“ und ihren Nachbar zu bekommen.

Methode “getConnectedNeighbours”

Diese Methode gibt die Menge von benachbarten Tiles zurück, die über mindestens eine dieser Edges verbunden sind.

Parameter:

Eine Menge von verbundenen Edges.

Funktionalität:

1. Erstellen ein neues “HashSet“ mit der Name “connectedNeighbours“ vom Typ “Tile“, um benachbarte Tiles zu spreichern.

2. Überqueren durch eine Menge von verbundenen Edges.

3. Erstellen ein neues “HashSet” mit der Name “adjancentTilePositions“ vom Typ “TilePosition“ , um Positionen der Tiles zu speichern, zwischen denen Edge liegt.

4. Überqueren durch alle solche Tilepositionen.

5. Erstellen ein neues Tile mit der Name “neighbour“ und durch Aufrufe von Methoden “getHexGrid“ und “getTileAt“ ein der benachbarten Tiles speichern.

6. Prüfen, ob bekommende Tile nicht “diese Tile“ ist. Falls die Bedingung erfüllt ist, speichern Tile “neighbour“ in einem “HashSet connectedNeigbours“.

7. Geben “HashSet connectedNeighbours“ (Menge von benachbarten Tiles) zurück.

Methode “getConnectedCities”

Diese Methode soll eine unveränderbare Sicht auf alle Städte zurückgeben, welche an mindestens ein Schienennetz angeschlossen sind.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “connectedCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Man braucht es, um weiter verbundene Städte zu speichern.

2. Erstellen ein neues “HashMap” mit der Name “cities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Dieses “HashMap“ besitzt alle Städte, die “diese HexGrid“ besitzt.

3. Erstellen ein neues “HashMap” mit der Name “edges“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Edges besitzt, und Value ein Edge ist. “HashMap“ besitzt alle Edges des “diesen HexGrid“.

4. Überqueren durch alle solche Edges.

5. Prüfen, ob Edge eine Schiene besitzt. Falls diese Bedingung erfüllt ist, überqueren durch alle TilePositionen der Edges.

6. Prüfen, ob es eine Stadt in dieser Position gibt. Falls diese Bedingung erfüllt ist, speichern eine Stadt mit seiner Position und seiner Name im “HashMap connectedCities“.

7. Wandeln “HashMap connectedCities“ in einer unveränderbaren Sicht um und geben sie zurück, die alle verbundene Städte besitzt.

Methode “getUnconnectedCities”

Diese Methode soll eine unveränderbare Sicht auf alle Städe zurückgeben, welche bisher an kein Schienennetz angeschlossen sind.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “unconnectedCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Man braucht es, um weiter nicht verbundene Städte zu speichern.

2. Erstellen ein neues “HashMap” mit der Name “cities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Dieses “HashMap“ besitzt alle Städte, die “diese HexGrid“ besitzt.

3. Erstellen ein neues “HashMap” mit der Name “connectedCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Dieses “HashMap” besitzt verbundene Städte.

4. Überqueren durch alle Städte.

5. Prüfen, ob es eine Stadt im “HashMap“ von verbundenen Städten gibt. Falls diese Bedingung nicht erfüllt ist, speichern eine Stadt mit seiner Position und seiner Name im “HashMap unconnectedCities“.

6. Wandeln “HashMap unconnectedCities“ in einer unveränderbaren Sicht um und geben sie zurück, die alle nicht verbundene Städte besitzt.

Methode “getStartingCities”

Diese Methode soll alle Startstädte zurückgeben.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “startingCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Startstadt besitzt, und Value eine Name der Startstadt ist. Man braucht es, um weiter Startstädte zu speichern.

2. Erstellen ein neues “HashMap” mit der Name “cities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Dieses “HashMap“ besitzt alle Städte, die “diese HexGrid“ besitzt.

3. Überqueren durch alle Städte.

5. Prüfen, ob eine Stadt eine Startstadt ist. Falls diese Bedingung erfüllt ist, speichern eine Stadt im “HashMap startingCities“ mit seiner Position und seiner Name.

6. Wandeln “HashMap startingCities“ in einer unveränderbaren Sicht um und geben sie zurück, die alle Startstädte besitzt.

2.1

Methode “canBuildRail”

Die Methode prüft, ob der Spieler an der gegebenen Kante bauen darf.

Parameter:

Eine Edge.

Funktionalität:

1. Prüfen, ob der Spieler schon an gegebener Edge gebaut hat. Falls Spieler schon an gegebener Edge gebaut hat, dann geben den Wert “false“ zurück.

2. Erstellen Basiskosten und Credits des Spielers.

3. Prüfen, ob Spieler genug Geld hat, um zu bauen.

4. Erstellen Parallelkosten.

5. Prüfen, ob der Spieler in Bauphase ist und genug Geld hat, um Parallelkosten zu bezahlen, oder ob der Spieler genug Geld hat, um alle Kosten zu bezahlen.

6. Falls die Bedingung erfüllt ist, geben den Wert “true“ zurück. Anderfalls geben den Wert “false” zurück. Erstellen ein neues “HashMap” mit der Name “startingCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Man braucht es, um weiter nicht Startstädte zu speichern.

Methode “getBuildableRails”

Die Methode gibt eine Liste von Kanten zurück, an denen der Spieler bauen kann.

Parameterlose

Funktionalität:

1**.** Erstellene ein “HashSet” vom Typ “Edge“ mit der Name “buildableRails“. Man braucht es, um weiter Kanten zu speichern, an denen der Spieler bauen kann.

2. Erstellen ein neues “HashMap” mit der Name “railsOfPlayer“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Schienen besitzt, und Value ein Edge ist, das eine Kante besitzt, wo eine Schiene ist. “HashMap“ besitzt alle Schienen des Spielers.

3. Erstellen ein neues “HashMap” mit der Name “edges“, wo Key ein “HashSet“ vom Typ “TilePosition“ ist, das TilePosition von Edges besitzt, und Value ein Edge ist. Das sind alle Edges.

4. Prüfen, ob Spieler keine Schienen hat. Falls diese Bedingung erfüllt ist, erstellen ein neues “HashMap” mit der Name “startingCities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Startstadt besitzt, und Value eine Name der Startstadt ist.

5. Überqueren durch alle Positionen der Startstädten mit erstem “for-Schleifen”.

6. Überqueren durch alle Edges mit zweitem “for-Schleifen”.

7. Prüfen, ob man an der Edge bauen kann, und ob Edge an einer der Startstadt grenzt. Falls beide Bedingungen erfüllt sind, speichern Edge im “HashMap buildableRails”.

8. Falls die Bedingung (Punkt #4) nicht erfüllt ist, überqueren durch alle Edges mit erstem “for-Schleifen”, wo der Spieler seine Schienen schon gebaut hat. Dann überqueren durch alle Edges mit zweitem “for-Schleifen”.

9. Prüfen, ob man an der Edge bauen kann, und ob Edge mit Schienennety des Spielers grenzt. Falls beide Bedingungen erfüllt sind, speichern Edge im “HashMap buildableRails”.

10. Geben “HashMap buildableRails” zurück, das Edges besizt, an denen der Spieler bauen kann.

2.2
Methode “buildRail”

Diese Methode baut eine Schiene an der angegebenen Kante.

Parameter:

Eine Edge.

Ausnahme:

IllegalActionException

Funktionalität:

1. Prüfen, ob man eine Schiene an gegebener Edge bauen kann. Falls die Bedingung nicht erfüllt ist, werfen eine “ IllegalActionException“-Ausnahme mit der Nachricht "Can't build Rail for edge".

2. Erstellen Parallelkosten.

3. Prüfen, ob die Schiene parallel zu bestehenden Schienen anderer Spieler verläuft. Falls diese Bedingung erfüllt ist, zahlt der bauende Spieler Parallelkosten anderen Spielern.

4. Der Bau reduziert das Baubudget des Spielers um die entsprechenden Basis-Baukosten der Schiene.

5. Prüfen, ob der bauende Spieler in der Bauphase ist. Falls diese Bedingung erfüllt ist, zahlt der Spieler zusätzlich die Parallelbaukosten. Falls diese Bedingung nicht erfüllt ist, zahlt der Spieler die gesamten Baukosten.

6. Bauen eine Schiene an der gegebenen Edge.

7. Erstellen ein neues “HashMap” mit der Name “cities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. “HashMap” besitzt alle Städte.

8. Erstellen ein neues “HashSet” mit der Name “adjancentTilePositions“ vom Typ “TilePosition“ , um Positionen der Tiles zu speichern, zwischen denen gegebene Edge liegt.

9. Überqueren durch alle “TilePositions“ der Tiles, zwischen denen gegebene Edge liegt. Erstellen eine Stadt, die ein Tile besitzt.

10. Prüfen, ob die Stadt existiert und ob die Stasdt eine Startstadt ist. Falls beide Bedingungen erfüllt sind, erhält der Spieler eine Bonusgutschrift in Höhe von Config.CITY_CONNECTION_BONUS.

2.3

Methode “executeBuildingPhase”

Diese Methode organisiert die Bauphase, in der Spieler abwechselnd würfeln und Schienen bauen, um unverbundene Städte miteinander zu verbinden

Parameterlose

Funktionalität:

1. Erstellen Anzahl der unverbundenen Städten.

2. Benutzen “while-Schleifen”, um Bauphase zu wiederholen.

3. Erhöhen die Anzahl der Runden um 1.

4. Bestimmen den Spieler, der in der aktuellen Runde würfeln darf. Erstellen diesen Spieler und seinen “Controller“.

5. Führen den Wurf des Spielers aus und speichern das Ergebnis des Wurfes.

6. Überqueren durch alle Spieler mit “for-Schleifen“.

7. Aktualisiere das Baubudget des Spielers basierend auf dem Würfelergebnis.

8. Führen Bauaktion. Aktualisieren Anzahl der unverbundenen Städten.

2.4

Methode “chooseCities”

Diese Methode wählt eine zufällige Start- und Zielstadt aus allen noch nicht gewählten Städten aus.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “cities“, wo Key vom Typ “TilePosition“ ist, das TilePosition von Stadt besitzt, und Value eine Name der Stadt ist. Dieses “HashMap“ besitzt alle Städte.

2.Erstellen ein neues “HashSet” mit der Name “chosenCities“ vom Typ “City“, das alle schon gewählte Städte besizt.

3. Filtern die Städte, ausgenommen die bereits gewählten.

4. Prüfen, ob es mindestens 2 solsche Städte gibt. Falls diese Bedingung erfüllt ist, erstellen Start- und Zielstadt.

5. Prüfen, ob gewählte Städte nicht identisch sind. Falls diese Bedingung nicht erfüllt ist, wählen eine neue Zielstadt, solange bis Bedingung erfüllt wird.

6. Markieren gewählte Städte, als gewählte.

2.5

Methode “canDrive”

Die Methode gibt zurück, ob der Spieler fahren darf oder nicht.

Parameterlose

Funktionalität:

1. Prüfen, ob der Spieler in der Fahrphase ist und der Spieler auch in der aktuellen Fahrerliste vorhanden ist.

2. Falls beide Bedingungen erfüllt sind, geben den Wert “true” zurück. Anderfalls geben den Wert “false” zurück.

Methode “drive”

Die Methode führt die Aktion des Fahrens aus.

Parameter:

Zielkachel vom Typ “Tile”.

Ausnahme:

IllegalActionException.

Funktionalität:

1**.** Prüfen, ob Spieler fahren darf und man die Zielkachel erreichen kann. Falls nicht beide Bedingungen erfüllt sind, werfen eine Ausnahme mit der Nachricht.

2. Falls beide Bedingungen erfüllt sind, führen den Wurf aus und speichern das Ergebins. Erstellen ein neues “HashMap” mit der Name “drivableTiles“, wo Key vom Typ “Tile“ ist, und Value ein “ArrayList“ vom Typ “Tile“. Dieses “HashMap“ besitzt alle Tiles, wo man fahren kann.

3. Überqueren durch alle Tiles und speichern Anzahl von Tiles, wo fman fahren kann.

4. Berechnen und speichern überschüssige Würfelpunkte.

5. Überqueren durch alle Tiles, wo fman fahren kann.

6. Prüfen, ob Ziel erreicht wurde. Falls diese Bedingung nicht erfüllt ist, aktualisieren Position des Spielers (Fahren weiter). Anderfalls setzen Position des Spieles in Zieltile und addieren Bonuspunkte.

2.6

Methode “ letPlayersChoosePath“

Die Methode fordert jeden Spieler auf, eine Strecke zu wählen, die er Fahren und Mieten möchte.

Parameterlose

Funktionalität:

1**.** Erstellen “PlayerController“ und Spieler, dem dieses “Controller“ gehört.

2. Setzen “PlayerController” für die Fahrphase zurück und setzen die Spielerposition auf die Startstadt.

3. Warten, bis der Spieler den Weg wählt und bestätigt.

2.7

Methode “handleDriving”

Diese Methode führt die Aktionen der Spieler aus, die die Strecken fahren.

Parameterlose

Funktionalität:

1. Erstellen ein neues “ArrayList“ vom Typ “Player“ und mit der Name “drivingPlayers“, das alle fahrende Spieler besitzt.

2. Prüfen, ob es fahrende Spieler gibt. Falls diese Bedingung nicht erfüllt ist, tut diese Methode nichts.

3. Falls es nur ein fahrender Spieler gibt, wird seine Position auf die der Zielstadt gesetzt und wird er als Winner markiert. Dann tut Methode nichts.

4. Erstellen ein neues “ArrayList“ vom Typ “Player“ und mit der Name “finishedPlayers“. Man braucht es, um weiter Spieler zu speichern, die Ziel erreicht haben.

5. Prüfen, ob es mehr als einen fahrenden Spieler gibt oder die Anzahl der Siegpunkte erreicht ist. Falls eine der diesen Bedingungen erfüllt ist, erstellen ein neues “HashMap” mit der Name “positions“, wo Key vom Typ “Player“ ist, und Value vom Typ “TilePosition“. Dieses “HashMap“ besitzt Positionen der Spieler.

6. Überqueren durch alle fahrende Spieler. Prüfen, ob Spieler ein fahrender Spieler ist. Dann prüfen, ob er eine Zielstadt erreicht hat. Falls diese Bedingung nicht erfüllt ist, subrahieren Credits des Spielers. Anderfalls werfen den Spieler aus fahrenden Spielern und markieren, als ein Spieler, der Ziel erreicht hat.

7. Sortieren die verbleibenden Spieler nach Anzahl der Credits.

8. Erstellen ein neues “HashMap” mit der Name “playerControllers“, wo Key vom Typ “Player“ ist, und Value vom Typ “PlayerController“. Dieses “HashMap“ besitzt “PlayerControlers” der Spieler.

9. Zuerst überqueren durch alle “PlayerControlers”. Dann überqueren durch alle fahrende Spieler. Prüfen, ob Spieler fährt. Falls diese Bedingung erfüllt ist, würfelt fahrende Spieler würfelt zuerst, wie weit er fahren kann, und dann wird die gewürfelte Distanz gefahren.

2.8

Methode “getWinners”

Die Methode getWinners() bestimmt die Gewinner des Rennens.

Parameterlose

Funktionalität:

1. Erstellen ein neues “HashMap” mit der Name “positions“, wo Key vom Typ “Player“ ist, und Value vom Typ “TilePosition“. Dieses “HashMap“ besitzt Positionen der Spieler.

2. Erstellen ein neues “HashMap” mit der Name “surplus“, wo Key vom Typ “Player“ ist, und Value vom Typ “Integer“. Dieses “HashMap“ besitzt Bonuspunkte für jeden Spieler.

3. Erstellen Position der Zielstadt.

4. Erstellen “ArrayList“ von Typ “Player“ und mit der Name “winners“. Dieses “ArrayList“ wird alle Gewinner besetzen.

5. Wandeln “winners“ in “Stream“ um. Zuerst werfen alle Spieler, die Zielstadt nicht erreicht haben. Dann sortieren alle Spieler nach Bonuspunkten. Dann erstellen ein Grenze, um richtige Anzahl der Gewinner zu bekommen. Am Edne wandeln “Stream“ in “ArrayList“ um.

6. Geben alle Gewinner zurück.

2.9

Methode “executeDrivingPhase”

Die Methode führt die Fahrphase aus, in der die Spieler abwechselnd würfeln und Strecken fahren.

Parameterlose

Funktionalität:

1. Methode funktioniert, solange noch nicht alle Städte angefahren (ausgewählt) wurden.

2. Erhöhen die Anzahl der Runden um 1.

3. Setzen die Liste der fahrenden Spieler, der Positionen der Spieler, die aktuell am Rennen teilnehmen und die Überschüsse im GameState zurück.

4. Prüfen, ob die Anzahl der Runden ein Vielfaches von 3 ist. Falls diese Bedingung erfüllt ist, löse die Bauphase durch die Methode “buildingDuringDrivingPhase()“ aus.

5. Erstellen Spieler, der in diesem Round spielt, und seinen “PlayerController“. Dann wird gewählter Spieler mit dem “Objective CHOOSE_CITIES“ dazu aufgefordert, die Start- und Zielstadt zu wählen.

6. Lassen den Spieler die Strecke wählen und dann die gewählte Strecke fahren.

7. Erstellen “ArrayList“ von Typ “Player“ und mit der Name “winners“. Dieses “ArrayList“ wird alle Gewinner besitzt.

8. Überqueren durch alle Gewinner und aktualisieren ihre Credits.

3.1:

Methode “savePlayerData”

Diese Methode speichert die Spieldaten eines Spielers (Name, Punktzahl, ob KI) in einer CSV-Datei.

Parameters:

- playerName (String) – Name des Spielers.
- score (int) – Punktzahl.
- ai (boolean) – Ob der Spieler eine KI ist.

Funktion:

1. Initialisiert die CSV-Datei.
2. Erstellt eine Zeichenkette mit den Spieldaten (Name, KI, Zeit, Punktzahl).
3. Schreibt die Daten in die CSV-Datei.

Ausnahmen:

- IOException – Wenn die Datei nicht gespeichert werden kann.

3.2:

Methode “loadLeaderboardData”

Diese Methode lädt die Spieldaten aus einer CSV-Datei in eine Liste.

Return:

- List<LeaderboardEntry> – Eine Liste mit den Spielern und ihren Ergebnissen.

Funktion:

1. Öffnet die CSV-Datei zum Lesen.
2. Überspringt die Kopfzeile.
3. Liest die Zeilen und zerlegt sie in Komponenten (Name, KI, Zeit, Punktzahl).
4. Erstellt LeaderboardEntry-Objekte und fügt sie der Liste hinzu.
5. Gibt die Liste zurück.

Ausnahmen:

- IOException – Wenn die Datei nicht gelesen werden kann.

4.1:

Methode “updateUIBasedOnObjective”

Aktualisiert die Benutzeroberfläche basierend auf dem aktuellen Ziel des Spielers.

Parameter:

- Objective (PlayerObjective)

Funktion:

1. Setzt die Benutzeroberfläche in den Ausgangszustand zurück.
2. Entfernt alle Hervorhebungen.
3. Aktualisiert die Spielerinformationen.
4. Prüft, ob der Spieler eine KI ist (wenn ja, wird die Methode beendet).
5. Aktiviert die entsprechenden UI-Elemente basierend auf den erlaubten Aktionen des Spielers.

4.2:

Methode “trimPath”

Schneidet den Pfad an der Stelle ab, die durch die terminateFunction-Funktion definiert wird.

Parameter:

- terminateFunction (BiFunction<Pair<Integer, Integer>, Integer, Boolean>) – Funktion, die das Ende des Pfades bestimmt.
- path (List) – Ursprünglicher Pfad.

Return:

- List<Edge> – Gekürzter Pfad.

Funktion:

1. Erstellt eine Liste kleinpath.
2. Durchläuft alle Kanten des Pfades.
3. Prüft, ob die terminateFunction-Bedingung erfüllt ist.
4. Falls ja – beendet die Verarbeitung.
5. Gibt den gekürzten Pfad zurück.

Methode “highlightPath”

Hebt einen Pfad auf dem Spielfeld hervor.

Parameter:

- path (List) – Pfad, der hervorgehoben werden soll.

Funktion:

1. Ruft den Spielfeld-Controller ab.

2. Durchläuft alle Kanten des Pfades.
3. Findet den entsprechenden EdgeController.
4. Ruft die Methode highlight() auf, um die Kanten hervorzuheben.

4.3:

Methode “highlightStartingTiles”

Hebt die Startfelder auf dem Spielfeld hervor.

Funktionsweise

1. Setzt das ausgewählte Feld zurück.
2. Ruft den Spielfeld-Controller ab.
3. Bestimmt, welche Felder hervorgehoben werden sollen.
4. Durchläuft alle relevanten Felder und aktiviert die Hervorhebung.

4.4:

Methode „addBuildHandlers“

Konfiguriert die Benutzeroberfläche, sodass der Spieler Schienen bauen kann. Hebt baubare Tiles hervor und setzt Event-Handler für die Interaktion.

Funktion:

1. Löscht den aktuell gewählten Schienenpfad.
2. Entfernt vorhandene Abonnements für Tile-Auswahl.
3. Prüft, ob der Spieler noch Budget hat – falls nicht, wird die Methode beendet.
4. Aktiviert einen Listener, um Änderungen an selectedRailPath zu überwachen.
5. Setzt Event-Handler für Mausbewegungen und Klicks auf Tiles:

- Mausbewegung: Berechnet möglichen Baupfad und hebt ihn hervor, falls er bezahlbar ist.
- Klick: Falls ein gültiger Pfad ausgewählt wurde, wird die Bauaktion ausgelöst.

5.1:

Klasse „DiceEvent“

Die Klasse DiceEvent repräsentiert zufällige Ereignisse, die nach einem Würfelwurf auftreten können. Es hat 10% zum Auslösen und ist in RollDiceAction Klasse initialisiert.

Methode „triggerEvent“

Löst ein zufälliges Ereignis aus und wendet dessen Effekt auf den Spieler an. Das Ereignis wird zufällig aus einer Liste von möglichen Effekten ausgewählt.

Parameter:

- player (PlayerController) - Der Spieler, der vom Ereignis betroffen ist.

Funktion:

- Erstellt eine String-Liste der möglichen Ereignisse
- Wählt ausfällige Event von der Liste
- Anwendet es mit der Methode „applyEvent“

Methode „applyEvent“

Wendet das ausgewählte Ereignis auf den Spieler an. Je nach Ereignis kann es sich um einen Bonus oder eine Strafe handeln.

Parameter:

- player (Player) - Der Spieler, der vom Ereignis betroffen ist.
- event (String) - Das auszulösende Ereignis.

Funktion:

- 5 Credits hinzufügen oder entfernen

5.2:

Klasse „BetterAiController“

Der BetterAiController ist eine verbesserte KI für das Spiel. Er trifft strategische Entscheidungen, indem er:

1. Die beste Schiene baut, die ins Budget passt.
2. Den optimalen Fahrweg zur Zielstadt wählt.
3. Nur erlaubte Aktionen ausführt, basierend auf dem aktuellen Spielstatus.

Konstruktor:

Macht die neue „bessere“ KI contoller.

Parameters:

- playerController (PlayerController) - Steuert den aktuellen Spieler
- hexGrid (HexGrid) - Repräsentiert das Spielfeld
- gameState (GameState)- Enthält den aktuellen Spielstatus
- activePlayerController (Property<PlayerController>)- Gibt an, welcher Spieler gerade am Zug ist
- diceRollProperty (IntegerProperty)- Speichert das letzte Würfelergebnis
- roundCounterProperty (IntegerProperty) - Zählt die aktuellen Spielrunden
- chosenCitiesProperty (ReadOnlyProperty<Pair<City, City>>)- Enthält die vom Spieler gewählten Zielstädte

Funktion:

- Erzeugt eine neue KI mit Zugriff auf Spielfeld, Spieler und Spielstatus.

Methode „executeActionBasedOnObjective“:

Führt eine oder mehrere Aktionen aus, basierend auf den erlaubten Spielzügen.

Parameter:

- objective (PlayerObjective)- das aktuelle Ziel des Spielers (z. B. Bauen, Fahren)

Funktion:

- Würfeln, Bauen, Fahren oder Städte wählen, je nach Spielstatus.

Ausnahmen:

- InterruptedException – Falls der Thread unterbrochen wird.

Methode „buildBestRail“:

Baut die beste mögliche Schiene.

Funktion:

- Erstellt eine Liste aller baubaren Schienen. Falls es leer ist machts mehr nichts.
- Falls keine Schienen baubar sind, passiert nichts.
- Speichert den Budget
- Sucht nach der besten Schiene, die gebaut werden kann, und baut sie dann

Methode „driveToTarget“:

Sucht nach beste Tile für die Bewegung.

Funktion:

- Erstellt eine Liste aller fahrbaren Tiles. Falls es leer ist machts mehr nichts.
- Falls keine Bewegung möglich ist, passiert nichts.
- Wählt das beste Tile für die Bewegung, um effizient zum Ziel zu gelangen, und fährt dort.

Methode „getPathLengthToTarget“:

Hilfsmethode, die die Länge der Bewegung zurückgibt.

Return:

- int – Die Menge von Tiles, die durchgegangen müssen werden.

Parameter:

- tile (Tile) - Das Ziel-Tile, dessen Pfadlänge berechnet wird.

Funktion:

- Gibt die Anzahl der Tiles im gespeicherten Fahrpfad zurück.
- Falls kein Pfad existiert, gibt die Methode Integer.MAX_VALUE zurück (damit das Tile nicht gewählt wird).