Graz University of Technology

Engineering- and Business Informatics

Picture source: https://www.doppelmayr.com/de/systeme/kuppelbare-sesselbahnen/
last access: 30.07.2021

Final Project:

TUSkiSim

Auslastungssimulation eines Schigebietes

Änderungsindex

Index	Seite	Änderung
1	3	Kapazitäten der Strecken korrigiert
2	19	Flowchart korrigiert
	20	Beschreibung Punkt 3 und 4.1 dem Flowchart angepasst
3	13	Ergänzung, dass die Zeit aufgerundet werden muss in Methode `calculateNeededTime`
	16	Expert: Fehler in Berechnung `calculateNeededTime` korrigiert und Aufrunden hinzugefügt
	19	Flowchart korrigiert (geänderte Elemente fett umrahmt)
	20	Beschreibung Punkt 4.1.1 geändert
	21	Punkt 3.2 eingefügt, Beschreibung Punkt 4.2, 4.3, 4.4.1 geändert
	22	zusätzliche Folie, aufgrund des neuen Punktes 3.2
	23	Punkt 1 ergänzt
	24	Punkt 6 und 7 ergänzt
	25	Punkt 10 ergänzt
	28	Testszenario ergänzt

Hinweis: Textliche Änderungen durch Schriftfarbe blau kenntlich gemacht.

Ausgangssituation

TUSki ist ein kleineres Schigebiet mit einer **Infrastruktur**, wie in der Abbildung dargestellt, die aus Liften, Strecken mit unterschiedlichem Schwierigkeitsgrad und Hütten besteht.
Das Schigebiet ist sehr innovativ und hat vor kurzem den gesamten Ticketverkauf digitalisiert.

Nun soll ein Programm erstellt werden, das basierend auf den Daten der verkauften Tickets eine **Simulation** durchführt.
Ziel ist es, die **Auslastung** der verschiedenen Elemente in TUSkiSim zu ermitteln.

Inputdaten für Ihre Simulation:

- Ticketverkäufe des Tages als csv-Datei
- Infrastruktur des Schigebiets (Lifte, Strecken und Hütten)
- Kapazitäten der Lifte

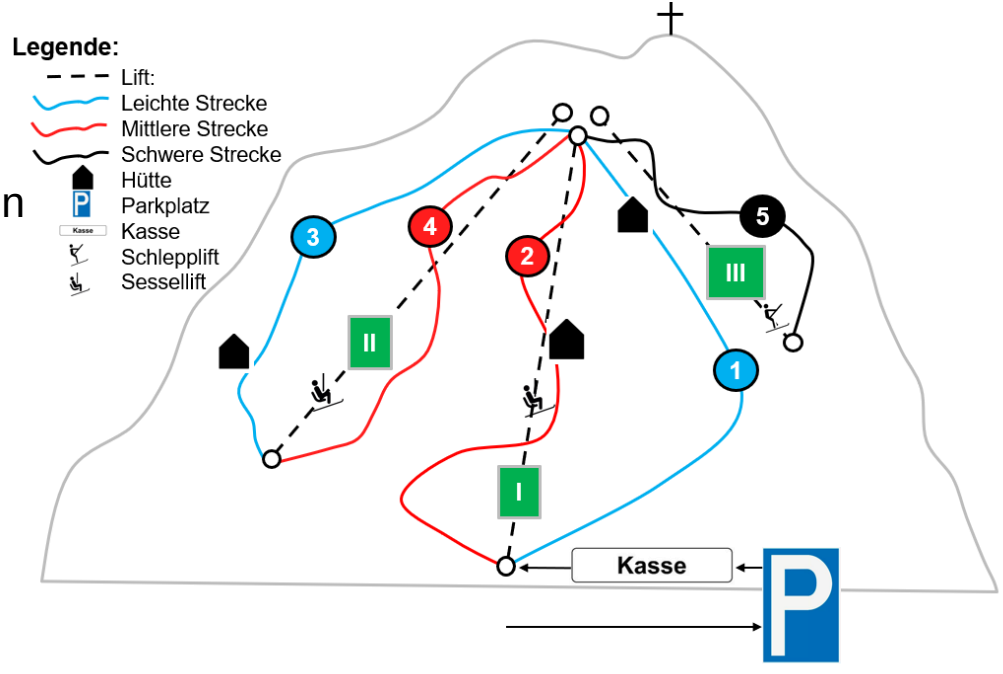


Abbildung 1: Infrastruktur des Schigebiets

Angaben zur Infrastruktur und den Schifahrern

Lifte:

Lifts					
number	type	velocity[m/min]	length[m]	elements	numberOfLanes/seats
1	Chair lift	100	1500	40	4
2	Chair lift	90	1200	30	2
3	Ski tow	50	600	30	2

Hütten:

Huts		
number	maxGuests	averageStay
1	200	40
2	150	45
3	100	25

Strecken:

Tracks					
number	level	Hut	length[m]	lift	capacity
1	1	Hut 1	2500	1	120
2	2	Hut 2	2200	1	80
3	1	Hut 3	1700	2	60
4	2	\	1600	2	70
5	3	\	800	3	50

Schifahrer:

Skiers			
type	velocity[m/min]	skillLevel	propHutBasic
Beginner	50	1	1,0
Advanced	150	2	0,8
Expert	250	3	0,2

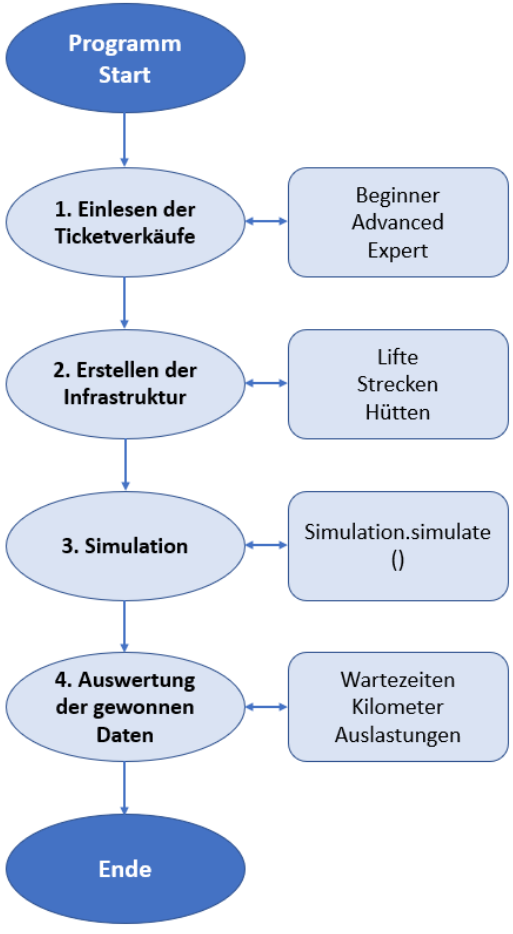
Grundidee

Folgende Schritte werden durch das Programm abgearbeitet:

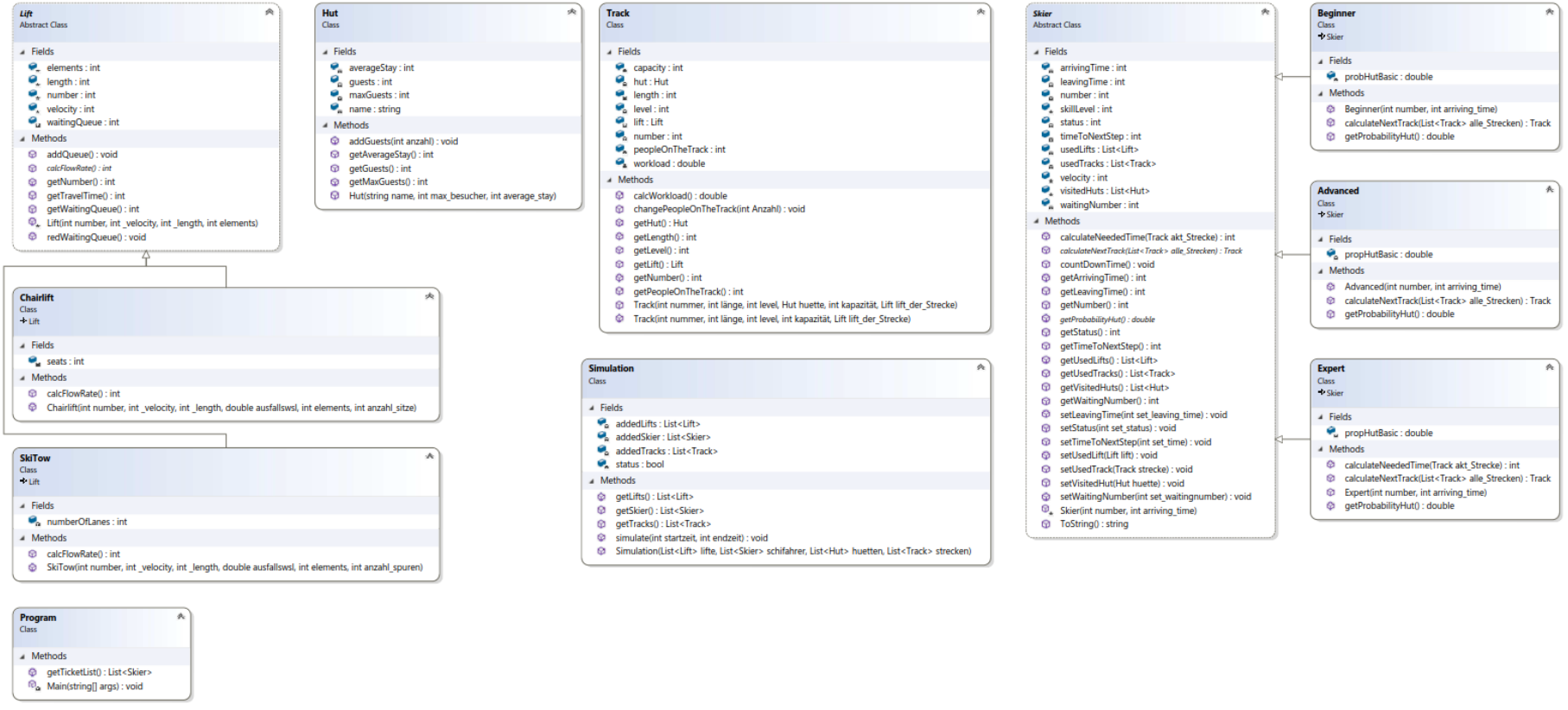
1. **Einlesen der Ticketverkäufe:**
   Mithilfe der importierten Daten wird für jedes Ticket ein Objekt Schifahrer*in erzeugt. Jede*r Schifahrer*in verfügt über ein Skill-Level, der in Beginner, Advanced und Expert unterteilt ist. Diese Objekterstellung erfolgt im Hauptprogramm mit Hilfe einer statischen Methode.
2. **Erstellen der Infrastruktur:**
   Diese Objekte werden direkt im Hauptprogramm erstellt. Achten Sie bei den Liften und Strecken auf die spezifischen Besonderheiten des jeweiligen Elementes.
3. **Simulation:**
   Die Klasse Simulation beinhaltet die eigentliche Logik dieses Programms. Die zuvor definierten Objekte werden der Simulation übergeben und für eine definierte Zeitspanne bearbeitet. Das Ziel dieser Simulation ist es, für jede*n Schifahrer*in Informationen über die gefahrenen Strecken, benutzten Lifte und besuchten Hütten zu generieren.
4. **Auswertung der gewonnen Daten:**
   Durch die Simulation (Punkt 3) und die daraus gewonnen Daten können nun Rückschlüsse auf die Auslastungen der einzelnen Elemente gezogen werden. Es ist ebenfalls möglich, einzelne Schifahrer*innen nach deren Aufenthaltsdauer oder zurückgelegten Wegstrecke zu gliedern.

Programmablauf

Dauer der Simulation
Startzeit: 8:00h
Endzeit:17:00h



Entwurf der Klassenstruktur (eine größere Version ist im TC zu finden)



Klassenbeschreibung

Abstrakte KLASSE: Lift

Attribute:

`number`	Fortlaufende Nummer
`velocity`	Fördergeschwindigkeit des Liftes
`length`	Länge des Liftes
`elements`	Anzahl der Tragelemente (Bügel, Sesselbänke, ...)
`waitingQueue`	Warteschlange

Methoden:

`calcFlowRate`	Abstrakte Methode zur Berechnung des Durchsatzes
`getTravelTime`	Berechnet die Fahrzeit: $travelTime = length/velcocity$
`getNumber`	Gibt die Nummer des Liftes zurück
`addQueue`	Erhöht die Warteschlange um Eins
`redWaitingQueue`	Reduziert die Warteschlange um den Durchsatz. Ein negativer Wert ist nicht vorgesehen.
`getWaitingQueue`	Gibt die aktuelle Läng der Warteschlange zurück

Klassenbeschreibung

KLASSE: Chairlift : Lift

Attribute:

`seats` Anzahl der Sitze pro Sesselbank

Methoden:

`calcFlowRate` Berechnung des Durchsatzes abhängig von der Bauart des Liftes:

$$Durchsatz = seats * velocity * \frac{elements}{length}$$

KLASSE: SkiTow : Lift

Attribute:

`numberOfLanes` Anzahl der vorhandenen Fahrspuren

Methoden:

`calcFlowRate` Berechnung des Durchsatzes abhängig von der Bauart des Liftes:

$$Durchsatz = numberOfLanes * velocity * \frac{elements}{length}$$

Klassenbeschreibung

KLASSE: Hut

Attribute:

`name`	Name der Hütte
`maxGuests`	Maximale Anzahl an Gästen pro Tag
`guests`	Anzahl der Schifahrer*innen welche die Hütte besuchten. Achtung → reine Zählvariable. Der*Die einzelne Schifahrer*in m muss nicht vermerkt werden.
`averageStay`	Durchschnittliche Verweildauer (siehe Folie 3)

Methoden:

`getMaxGuests`	Gibt die Anzahl der maximalen Besucher*innen pro Tag zurück.
`addGuests`	Erhöht *guests* um 1.
`getGuests`	Gibt die Variable *guests* zurück.
`getAverageStay`	Gibt die durchschnittliche Verweildauer zurück.

Klassenbeschreibung

KLASSE: Track
Für diese Klasse soll die Möglichkeit bestehen Objekte mit oder ohne Hütte zu erstellen!

Attribute:

`number`	Fortlaufende ID der Strecke → siehe Abbildung 1.
`length`	Länge der Strecke
`level`	Schwierigkeitsgrad der Strecken 1 → blaue Strecke, einfach 2 → rote Strecke, mittel 3 → schwarze Strecke, schwer
`hut`	Der Strecke zugewiesene Hütte → siehe Abbildung 1 bzw. Angaben zur Infrastruktur.
`lift`	Der Strecke zugewiesener Lift → siehe Abbildung 1 bzw. Angaben zur Infrastruktur.
`capacity`	Kapazität der Strecke
`workload`	Auslastung der Strecke
`peopleOnTheTrack`	Anzahl der Schifahrer*innen auf der Piste

Methoden:

`calcWorkload`	Berechnung der Auslastung (achten Sie auf den Datentyp). $$Workload = peopleOnTheTrack/capacity$$
`changepeopleOnTheTrack`	Bekommt eine Anzahl von Personen übergeben und setzt damit *peopleOnTheTrack*.
`getPeopleOnTheTrack`	Gibt *peopleOnTheTrack* zurück.
`getLength`	Gibt die Länge der Strecke zurück.
`getHut`	Gibt die der Strecke zugewiesene Hütte zurück.
`getLift`	Gibt den der Strecke zugewiesenen Lift zurück.
`getLevel`	Gibt den Schwierigkeitsgrad zurück.
`getNumber`	Gibt die Nummer der Strecke zurück.

Klassenbeschreibung

Abstrakte KLASSE: Skier

Attribute:

`usedTracks`	Liste von gefahrenen Strecken
`usedLifts`	Liste der benutzten Lifte
`visitedHuts`	Liste der besuchten Hütten
`number`	ID der ankommenden Schifahrer*innen
`skillLevel`	1 → Beginner; 2 → Advanced; 3 → Expert
`arrivingTime`	Ankunftszeit im Schigebiet
`leavingTime`	Zeitpunkt des Verlassens des Schigebietes
`status`	Variable welche die Position des Schifahrers definiert: -1 → Status vor der ersten Liftbenutzung 0 → Im Lift oder im Wartebereich des Liftes 1 → Auf der Strecke (beinhaltet Pausen in Hütten) 2 → Schifahrer hat das Schigebiet verlassen
`waitingNumber`	Falls es zu Überlastungen einzelner Elemente kommt, gibt diese Nummer die aktuelle Position im Wartebereich an.
`timeToNextStep`	Zählvariable: Wird beim Betreten eines Elements gesetzt und bei jedem Zeitschritt um 1 reduziert. Nächste Aktion des Schifahrers erst möglich, wenn timeToNextStep=0.
`velocity`	Durchschnittliche Geschwindigkeit eines Schifahrers. Werte siehe Folie 3.

Klassenbeschreibung

Abstrakte KLASSE: Skier

Methoden (1/2):

`setUsedTrack`	Die Methode bekommt eine Strecke übergeben und fügt diese der Liste *usedTracks* hinzu.
`setUsedLift`	Die Methode bekommt einen Lift übergeben und fügt diesen der Liste *usedLifts* hinzu.
`setVisitedHut`	Die Methode bekommt eine Hütte übergeben und fügt diese der Liste *visitedHuts* hinzu.
`getUsedTracks`	Gibt die Liste *usedTracks* zurück.
`getUsedLifts`	Gibt die Liste *usedLifts* zurück.
`getVisitedHuts`	Gibt die Liste *visitedHuts* zurück.
`getArrivingTime`	Gibt die Ankunftszeit des*r Schifahrers*in im Schigebiet zurück.
`getStatus`	Gibt den aktuellen Status des*r Schifahrers*in zurück.
`getNumber`	Gibt die Nummer/ID des*r Schifahrers*in zurück.
`setStatus`	Bekommt einen neuen Status übergeben und setzt diesen.
`getWaitingNumber`	Gibt die *WaitingNumber* zurück.
`setWaitingNumber`	Bekommet die neue *WaitingNumber* übergeben und setzt diese.
`getTimeToNextStep`	Gibt *timeToNextStep* zurück.
`setTimeToNextStep`	Bekommt eine Zeit übergeben und setzt *timeToNextStep* auf diese.
`countDownTime`	Zählt *timeToNextStep* herunter. Achtung! → Negative Werte sollen verhindert werden.
`calculateNextTrack`	Abstrakte Methode. Berechnet je nach *skillLevel* eine neue Strecke. Genauere Beschreibung folgt in den erbenden Klassen.

Klassenbeschreibung

Abstrakte KLASSE: Skier

Methoden (2/2):

`getProbabilityHut`	Abstrakte Methode. Berechnet je nach *skillLevel* die Wahrscheinlichkeit eines Hüttenbesuches. Genauere Beschreibung folgt in den erbenden Klassen.
`calculateNeededTime`	Virtuelle Methode, welcher eine Strecke übergeben wird. Berechnet die Zeit, welche für die Abfahrt auf einer Strecke benötigt wird. Die Zeit ist aufzurunden! $neededTime = Länge\ der\ Strecke / velocity$
`getLeavingTime`	Gibt den Zeitpunkt des Verlassens des Schigebiets zurück.
`setLeavingTime`	Bekommt eine Zeit übergeben und setzt *leavingTime* auf diese.
`ToString`	Methode welche die Nummer, den Skill-Level, die Ankunftszeit und die Endzeit des Schifahrers zurückgibt.

Klassenbeschreibung

KLASSE: Beginner : Skier

Attribute:

`propHutBasic` Variable zur Berechnung der Wahrscheinlichkeit eins Hüttenbesuches. `propHutBasic=1`

Methoden:

`calculateNextTrack` Diese Methode bekommt eine Liste aller Strecken übergeben und berechnet damit die vom*von der Schifahrer*in als nächstes gewählte Strecke. Folgende Logik soll hier verwendet werden:

- Es können nur Strecken ausgewählt werden bei denen folgendes gilt: „level_Track“ <= „skillLevel_Skier“.
- Die Wahrscheinlichkeit für eine Strecke des Levels 1 liegt bei 0,5 (andere Strecken sollten in Ihrer Berechnung für den Beginner nicht auftauchen, es ist also eine 50/50 Auswahl zwischen den beiden Strecken mit dem Level 1).
- Wird eine Strecke ausgewählt, soll die Auswahl beendet werden.
- Wurde keine Strecke ausgewählt soll die Strecke 1 (laut Abbildung 1) gewählt werden.

`getProbabilityHut` Berechnet die Wahrscheinlichkeit eines Hüttenbesuches.

Liegt die Anzahl der besuchten Hütten unter 3 gilt:

$$propHut = propHut_{basic} * (3 -' number\ of\ visited\ huts')$$

Liegt die Anzahl der besuchten Hütten bei 3 oder größer gilt:

$$propHut = propHut_{basic} * 0.5$$

Klassenbeschreibung

KLASSE: Advanced : Skier

Attribute:

`propHutBasic` Variable zur Berechnung der Wahrscheinlichkeit eins Hüttenbesuches. `propHutBasic=0.8`

Methoden:

`calculateNextTrack` Diese Methode bekommt eine Liste aller Strecken übergeben und berechnet damit die vom Schifahrer als nächstes gewählte Strecke. Folgende Logik soll hier verwendet werden:

- Es können nur Strecken ausgewählt werden bei denen folgendes gilt: „level_Track“ <= „skillLevel_Skier“.
- Die Wahrscheinlichkeit für eine Strecke des Levels 1 liegt bei 0,7.
- Die Wahrscheinlichkeit für eine Strecke des Levels 2 liegt bei 0,3.
- Wird eine Strecke ausgewählt, soll die Auswahl beendet werden.
- Wurde keine Strecke ausgewählt soll die Strecke 1 (laut Abbildung 1) gewählt werden.

`getProbabilityHut` Berechnet die Wahrscheinlichkeit eines Hüttenbesuches.

Liegt die Anzahl der besuchten Hütten unter 2 gilt:

$$propHut = propHut_{basic} * (2 -' number\ of\ visited\ huts')/2$$

Liegt die Anzahl der besuchten Hütten bei 2 oder größer gilt:

$$propHut = propHut_{basic} * 0.5$$

Klassenbeschreibung

KLASSE: Expert : Skier

Attribute:

`propHutBasic` Variable zur Berechnung der Wahrscheinlichkeit eins Hüttenbesuches. `propHutBasic=0.2`

Methoden:

`calculateNextTrack` Diese Methode bekommt eine Liste aller Strecken übergeben und berechnet damit die vom Schifahrer als nächstes gewählte Strecke. Folgende Logik soll hier verwendet werden:

- Es können nur Strecken ausgewählt werden bei denen folgendes gilt: „level_track“<=„skillLevel_Skier“.
- Die Wahrscheinlichkeit für eine Strecke des Levels 1 liegt bei 0,2.
- Die Wahrscheinlichkeit für eine Strecke des Levels 2 liegt bei 0,3.
- Die Wahrscheinlichkeit für eine Strecke des Levels 3 liegt bei 0,5.
- Wird eine Strecke ausgewählt soll die Auswahl beendet werden.
- Wurde keine Strecke ausgewählt soll die Strecke 1 (laut Abbildung 1) gewählt werden.

`getProbabilityHut` Berechnet die Wahrscheinlichkeit eines Hüttenbesuches.

Liegt die Anzahl der besuchten Hütten unter 1 gilt:

$$propHut = propHut_{basic} * (1 -' number\ of\ visited\ huts')/2$$

Liegt die Anzahl der besuchten Hütten bei 1 oder größer gilt:

$$propHut = propHut_{basic} * 0.5$$

`calculateNeededTime` Es gilt folgende Berechnung (Die Zeit ist aufzurunden!):

$$neededTime = \frac{Länge\ der\ Strecke}{velocity} * \left(1 + \frac{Auslastung\ der\ Strecke}{2}\right)$$

Klassenbeschreibung

KLASSE: Simulation

Attribute:

`addedSkier`	Liste von Schifahrern*innen im Schigebiet, für jene die Simulation durchgeführt werden soll.
`addedTracks`	Liste der vorhandenen Strecken im Schigebiet.
`addedLifts`	Liste der vorhandenen Lifte im Schigebiet.
`status`	Statusvariable, welche nach erfolgreicher Simulation auf „True“ gesetzt wird.

Methoden:

`simulate`	Führt die Simulationslogik durch. Eine exakte Beschreibung dieser Methode finden Sie ab Folie 19. Dieser Methode wird eine Start- und Endzeit übergeben.
`getSkier`	Nach einer erfolgreichen Simulation gibt diese Methode die Liste der Schifahrer*innen zurück.
`getTracks`	Nach einer erfolgreichen Simulation gibt diese Methode die Liste der Strecken zurück.
`getLifts`	Nach einer erfolgreichen Simulation gibt diese Methode die Liste der Lifte zurück.

Klassenbeschreibung

KLASSE: Programm

Methoden:

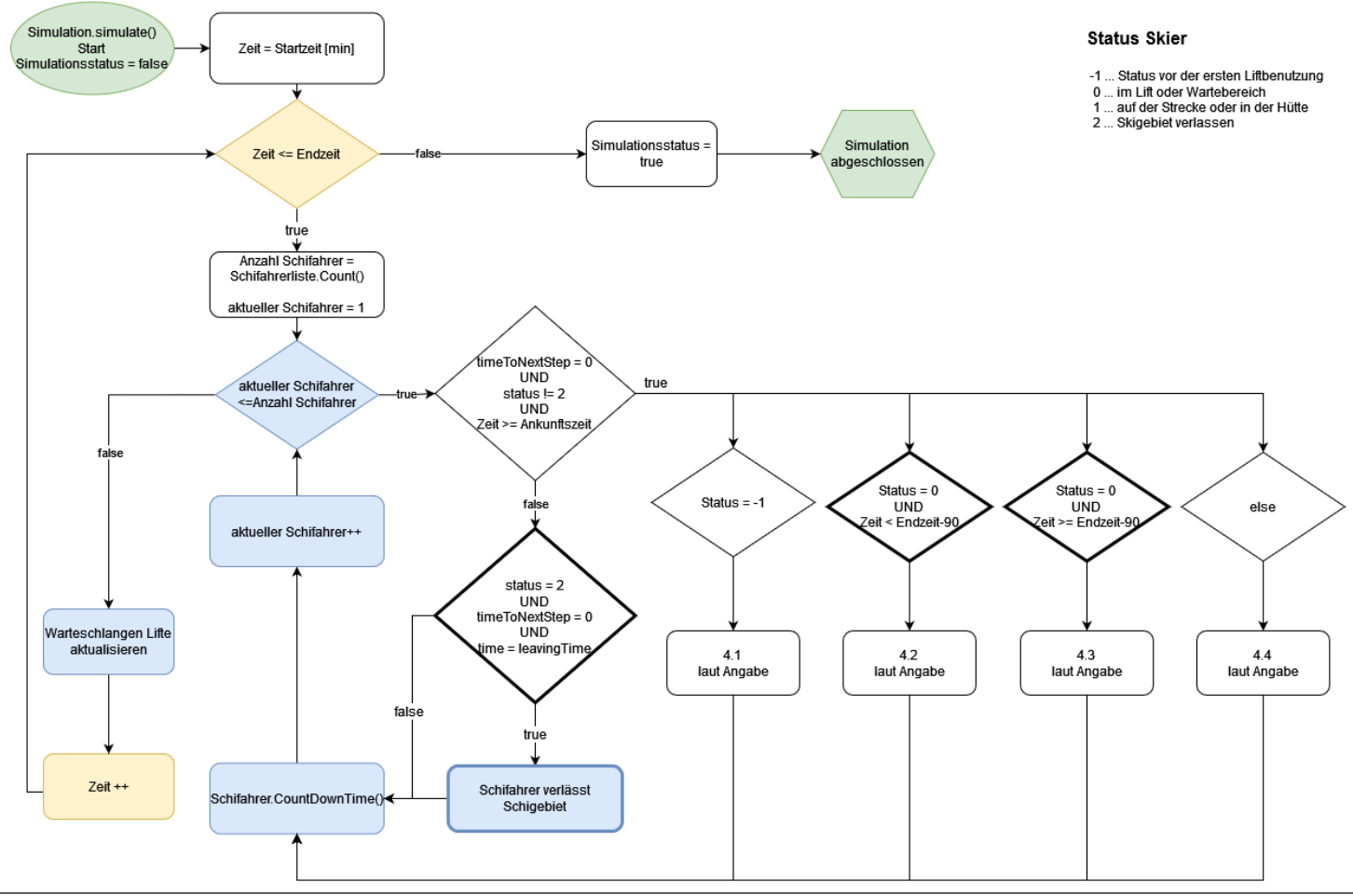
`Main` Ihr Hauptprogramm

`getTicketList` Gibt die eingelesene Liste mit den erstellten Schifahrern*innen zurück.

Verwenden Sie dazu eine in der Übung gezeigte Einlesemethode und implementieren Sie diese als statische Methode.

Lesen Sie dazu das gegebene CSV-File ein und erstellen Sie je nach Skilllevel ein passendes Objekt.

Logikbeschreibung Simulation.simulate() - Flowchart



Logikbeschreibung Simulation.simulate()

Aufbau der Simulation:
Hier erfolgt die Beschreibung der logischen Elemente. Die erforderlichen Aktionen 1-11 werden im Anschluss genauer erläutert. Der Aufbau der Logik kann ebenfalls dem beigelegten Flussdiagramm entnommen werden.

Ebene 1: Eine Schleife durchläuft den Zeitraum von der Start- bis zur Endzeit der Simulation.

- Start- und Endzeit werden in **Stunden** übergeben.
- Ein Schleifendurchlauf entspricht **einer Minute** → **Umrechnung notwendig.**
- **90 min** vor Erreichen der Endzeit der Simulation ist nur noch eine letzte Abfahrt in Richtung Parkplatz möglich (siehe 6. letzte Abfahrt).

Ebene 2: Eine weitere Schleife durchläuft pro Zeitschritt jede*n Schifahrer*in

Ebene 3: 3.1 Abfrage ob
- timeToNextStep = 0
- Schigebiet wurde noch nicht verlassen
- Ankunftszeit erreicht

Ebene 4: Statusabfrage und setzen von neuen Aktionen
4.1: Wenn (Status = -1)
→ *1.Warteschlange aktualisieren*

4.1.1: Wenn (Durchsatz_Lift1 ≥ Wartenummer des*r Schifahrers*in)
→ *2.Lift wählen und Status setzen*

4.1.2: Sonst
→ *3.Wartenummer reduzieren*

Logikbeschreibung Simulation.simulate()

Aufbau der Simulation (Fortsetzung):

4.2: Sonst, wenn (Status = 0 UND Zeit < Endzeit der Simulation - 90 Minuten)
→ *4.Nächste Strecke auswählen*
4.2.1: Wenn (Strecke besitzt Hütten)
4.2.1.1: Wenn (Hütte wird angefahren UND maxGuests nicht erreicht)
→ *5.Hüttenbesuch vermerken*

4.3: Sonst, wenn (Status = 0 UND Zeit ≥ Endzeit der Simulation - 90 Minuten)
→ *6.Letzte Abfahrt*

4.4: Sonst
→ 7.Nächsten Lift auswählen und Warteschlange aktualisieren
4.4.1: Wenn (Durchsatz_Lift ≥ Wartenummer des*r Schifahrers*in)
→ *8.Lift und Status setzen*
4.4.2: Sonst
→ *9.Wartenummer reduzieren*

3.2 Abfrage ob
- timeToNextStep = 0
- Status = 2
- Zeitpunkt des Verlassens des Skigebiets erreicht
→ *10. Schifahrer verlässt Schigebiet*

3.3: *11. Schifahrer*in CountDownTime()*

Logikbeschreibung Simulation.simulate()

Aufbau der Simulation (Fortsetzung):

2: Nächste*r Schifahrer*in

2.1: *12. Warteschlange aller Lifte aktualisieren*

1: time++

Simulation abgeschlossen

Logikbeschreibung Simulation.simulate()

Aufbau der Simulation: Detailbeschreibung der Aktionsblöcke

1. Warteschlange aktualisieren:
   - Warteschlange des Liftes 1 erhöhen (addQueue()) Achtung!: Schifahrer*in darf der Warteschlange nur einmal hinzugefügt werden
   - Wartenummer des*r Schifahrers*in instanziieren (setWaitingNumber()) Achtung!: Wartenummer des*r Schifahrers*in darf nur einmal gesetzt werden und nicht überschrieben werden!

2. Lift wählen und Status setzen:
   - Status des*r Schifahrers*in auf 0 setzen (setStatus())
   - Lift als benutzten Lift vermerken (setUsedLift())
   - Timer des*r Schifahrers*in setzen (setTimeToNextStep(getTraveltime_Lift1))
   - Wartenummer des*r Schifahrers*in zurücksetzen

3. Wartenummer reduzieren:
   - Berechnung der neuen Wartenummer ($Wartenummer - Durchsatz$)
   - setWaitingNumber (neue Wartenummer)

4. Nächste Strecke auswählen:
   - Nächste Strecke auswählen (calculateNextTrack())
   - Personenanzahl auf der ausgewählten Strecke erhöhen (changePeopleOnTheTrack())
   - Status des*r Schifahrers*in auf 1 setzen (set Status())
   - Strecke als gefahrene Strecke vermerken (setUsedTrack())
   - Timer des*r Schifahrers*in setzen (setTimeToNextStep (calculateNeededTime())

Logikbeschreibung Simulation.simulate()

Aufbau der Simulation: Detailbeschreibung der Aktionsblöcke (Fortsetzung)

5. Hüttenbesuch vermerken:
   - Timer des*r Schifahrers*in um die Zeit in der Hütte erhöhen
   - Hütte als besuchte Hütte vermerken (setVisitedHut())
   - Zähler der Hütte erhöhen

6. Letzte Abfahrt:
   - Status des*r Schifahrers*in auf 2 setzen (setStatus())
   - Auswahl der Strecke 1 oder 2 (setUsedTrack())
   - Personenanzahl auf der ausgewählten Strecke erhöhen (changePeopleOnTheTrack())
   - Timer des*r Schifahrers*in setzen (setTimeToNextStep (calculateNeededTime())
   - setLeavingTime (time + calculateNeededTime(Strecke))

7. Lift auswählen und Warteschlange aktualisieren:
   - Auswahl des nächsten Liftes. Hinweis: jede Strecke hat einen Lift zugewiesen.
   - Personen auf der zuvor gefahrenen Strecke reduzieren
   - Warteschlange des Liftes erhöhen: Achtung!: Schifahrer*in darf der Warteschlange nur einmal hinzugefügt werden.
   - Wartenummer des*r Schifahrers*in instanziieren: Achtung!: Wartenummer des*r Schifahrers*in darf nur einmal gesetzt werden und nicht überschrieben werden!

Logikbeschreibung Simulation.simulate()

Aufbau der Simulation: Detailbeschreibung der Aktionsblöcke (Fortsetzung)

8. Lift und Status setzen:
   - Status des*r Schifahrers*in auf 0 setzen (setStatus())
   - Lift als benutzten Lift vermerken (setUsedLift())
   - Timer des*r Schifahrers*in setzen (setTimeToNextStep(getTraveltime_Lift1))
   - Wartenummer des*r Schifahrers*in zurücksetzen
   - Personen der letzten Strecke um 1 reduzieren (changePeopleOnTheTrack())

9. Wartenummer reduzieren:
   - Berechnung der neuen Wartenummer ($Wartenummer - Durchsatz$)
   - setWaitingNumber (neue Wartenummer)

10. Schifahrer verlässt Schigebiet:
   - Personen der letzten Strecke um 1 reduzieren (changePeopleOnTheTrack())

11. Schifahrer*in CountDownTime:
   - Für jede*n Schifahrer*in: countDownTime()

12. Warteschlangen der Lifte aktualisieren:
   - Für jeden Lift: redWaitingQueue()

Auswertungen und Konsolenausgaben

Folgende Konsolenausgaben sollen Sie in Ihr Programm implementieren:
(Beachten Sie: Aufgrund der zufälligen Auswahlen der Pisten und Hütten, werden Ihre Werte von den gezeigt Werten auf den Konsolenausgaben abweichen)

1. Halbstündliche Ausgabe der Auslastung aller Strecken während der Simulation.

```
Uhrzeit: 8,5h   Die Strecke: 1 hat aktuell eine Auslastung von: 5%
Uhrzeit: 8,5h   Die Strecke: 2 hat aktuell eine Auslastung von: 12%
Uhrzeit: 8,5h   Die Strecke: 3 hat aktuell eine Auslastung von: 0%
Uhrzeit: 8,5h   Die Strecke: 4 hat aktuell eine Auslastung von: 4%
Uhrzeit: 8,5h   Die Strecke: 5 hat aktuell eine Auslastung von: 0%

Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 9h     Die Strecke: 1 hat aktuell eine Auslastung von: 9%
Uhrzeit: 9h     Die Strecke: 2 hat aktuell eine Auslastung von: 14%
Uhrzeit: 9h     Die Strecke: 3 hat aktuell eine Auslastung von: 2%
Uhrzeit: 9h     Die Strecke: 4 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9h     Die Strecke: 5 hat aktuell eine Auslastung von: 0%

Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 9,5h   Die Strecke: 1 hat aktuell eine Auslastung von: 42%
Uhrzeit: 9,5h   Die Strecke: 2 hat aktuell eine Auslastung von: 50%
Uhrzeit: 9,5h   Die Strecke: 3 hat aktuell eine Auslastung von: 12%
Uhrzeit: 9,5h   Die Strecke: 4 hat aktuell eine Auslastung von: 12%
Uhrzeit: 9,5h   Die Strecke: 5 hat aktuell eine Auslastung von: 10%
```

Auswertungen und Konsolenausgaben (Fortsetzung)

2. Nach der Simulation soll Ihr Programm für jede*n Schifahrer*in folgende Informationen auf der Konsole ausgeben:
   - Fortlaufende Nummer des*r Schifahrers*in
   - Skilllevel
   - Ankunftszeit
   - Zeitpunkt des Verlassens des Schigebietes
   - Gefahrene Kilometer

```
Fortlaufende Nummer: 1   Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 17,4  gefahrene Kilometer: 20
Fortlaufende Nummer: 2   Skilllevel: 3   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,7  gefahrene Kilometer: 43
Fortlaufende Nummer: 3   Skilllevel: 3   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,7  gefahrene Kilometer: 45
Fortlaufende Nummer: 4   Skilllevel: 2   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,9  gefahrene Kilometer: 34
Fortlaufende Nummer: 5   Skilllevel: 3   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,8  gefahrene Kilometer: 44
Fortlaufende Nummer: 6   Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 16,7  gefahrene Kilometer: 18
Fortlaufende Nummer: 7   Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 16,8  gefahrene Kilometer: 19
Fortlaufende Nummer: 8   Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 16,9  gefahrene Kilometer: 19
Fortlaufende Nummer: 9   Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 17,2  gefahrene Kilometer: 19
Fortlaufende Nummer: 10  Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 17    gefahrene Kilometer: 19
Fortlaufende Nummer: 11  Skilllevel: 3   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,7  gefahrene Kilometer: 43
Fortlaufende Nummer: 12  Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 17    gefahrene Kilometer: 19
Fortlaufende Nummer: 13  Skilllevel: 1   Ankunftszeit: 8         Zeitpunkt des Verlassens: 16,5  gefahrene Kilometer: 17
Fortlaufende Nummer: 14  Skilllevel: 2   Ankunftszeit: 8         Zeitpunkt des Verlassens: 15,9  gefahrene Kilometer: 33
Fortlaufende Nummer: 15  Skilllevel: 2   Ankunftszeit: 8         Zeitpunkt des Verlassens: 16,1  gefahrene Kilometer: 34
```

Testszenario

Um Ihnen die Möglichkeit zu bieten, Ihr Programm vorab zu testen, wurden zwei Testszenarien entworfen.

Testszenario „groß“: CSV-Datei „Ticketverkäufe“+ Angaben zur Infrastruktur laut Angabe

Testszenario „klein“: CSV-Datei „Test“+ Attribut „seats“ von „Chairlift1“ hat den Wert 2

Weitere Annahmen:

1. Die Methode *getProbabilityHut() aller Schifahrertypen gibt immer den Wert 0 zurück.*
2. Die Methode *calculateNextTrack() aller Schifahrertypen gibt immer die Strecke 1 zurück.*

Ausgabe des Testszenarios “groß”:

```
Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 8,5h    Die Strecke: 1 hat aktuell eine Auslastung von: 13%
Uhrzeit: 8,5h    Die Strecke: 2 hat aktuell eine Auslastung von: 0%
Uhrzeit: 8,5h    Die Strecke: 3 hat aktuell eine Auslastung von: 0%
Uhrzeit: 8,5h    Die Strecke: 4 hat aktuell eine Auslastung von: 0%
Uhrzeit: 8,5h    Die Strecke: 5 hat aktuell eine Auslastung von: 0%

Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 9h      Die Strecke: 1 hat aktuell eine Auslastung von: 13%
Uhrzeit: 9h      Die Strecke: 2 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9h      Die Strecke: 3 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9h      Die Strecke: 4 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9h      Die Strecke: 5 hat aktuell eine Auslastung von: 0%

Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 9,5h    Die Strecke: 1 hat aktuell eine Auslastung von: 86%
Uhrzeit: 9,5h    Die Strecke: 2 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9,5h    Die Strecke: 3 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9,5h    Die Strecke: 4 hat aktuell eine Auslastung von: 0%
Uhrzeit: 9,5h    Die Strecke: 5 hat aktuell eine Auslastung von: 0%

Halbstündliche Ausgabe der Auslastung der Strecken
Uhrzeit: 10h     Die Strecke: 1 hat aktuell eine Auslastung von: 83%
Uhrzeit: 10h     Die Strecke: 2 hat aktuell eine Auslastung von: 0%
Uhrzeit: 10h     Die Strecke: 3 hat aktuell eine Auslastung von: 0%
Uhrzeit: 10h     Die Strecke: 4 hat aktuell eine Auslastung von: 0%
Uhrzeit: 10h     Die Strecke: 5 hat aktuell eine Auslastung von: 0%
```

Einzelne Zwischenausgaben und Ergebnisse des Testszenarios „klein“ , finden Sie in den beigelegten Textdateien

Gehen Sie bei der Erstellung Ihres Programmes systematisch vor und testen Sie einzelne Programmschritte (z.B. sind alle Objekte erstellt worden, sind meine Listen richtig befüllt, werden bei Verzweigungen die richtigen Entscheidungen getroffen…)