

2. Anforderung, Projekt, Funktionalität

DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia
Hedrich Ádám
Pintér Balázs
Fucskár Patrícia
Tassi Timián

SOSK6A
H9HFFV
ZGY18G
XKYAOO
MYY53U

adamzsofi.mail@gmail.com
hedrichadam09@gmail.com
pinterbalazs21@gmail.com
fucskar.patricia@gmail.com
timian.tassi@gmail.com

17. Februar 2020

1. Anforderung, Projekt, Funktionalität

1.1. Einführung

1.1.1. Ziel

Dieses Dokument soll die grundlegenden Informationen für Planung dieses Softwareprojekt enthalten.

1.1.2. Fachgebiet

Das Software ist ein Computerspiel, das man lokal mit anderen spielen kann. Es braucht Zusammenspiel, damit kann es die Gruppendynamik zwischen den Benutzern entwickeln.

1.1.3. Definitionen, Kürzungen

Es gibt keine solche Kürzungen.

1.1.4. Linken

<https://www.tarsasjatek.club/2016/04/11/forbidden-desert-a-homok-rabjai/>

<https://www.iit.bme.hu/targyak/BMEVIIIA02/feladat>

1.1.5. Zusammenfassung

Dieses Dokument verlautet unsere ersten, frühzeitigen Planen über diese Software. Es enthält die gebrauchten Teile, wie Use-Cases, Anforderungen, die zu benutzende Ressourcen und die Funktionen.

1.2. Überblick

1.2.1. Allgemeiner Überblick

Die Spieler kommunizieren mit dem Spiel durch graphische Benutzeroberfläche. Sie sehen die Ausführung von ihren Befehlen auch hier. Die Spielfläche wird durch ein anderes wichtiges Subsystem realisiert, bei dem die Spieler, Eisplatten, Gegenstände am Anfang ins Spiel kommen.

Wichtigste Subsystem ist der Spielmotor, der das zügige und fehlerlose Spiel für die Spieler sichert.

Zu diesem Spiel braucht man keine besondere Datenbank. Die Spieler können miteinander ohne Netzwerkverbindung spielen.

1.2.2. Funktionen

In diesem Spiel können wir mit anderen lokal spielen. Alle Spieler gehören zu einem Team. Das Ziel ist das Spiel zu überleben und die Komponenten von einer Signalfackel zusammenzustellen. Ein Spieler kann entweder Eskimo oder Forscher sein. Am Anfang des Spiels können die Spieler ihre Art nicht auswählen, es wird zufällig aufgeteilt. So ist das Spiel ein bisschen schwieriger. Um das Spiel zu starten muss man die Anzahl von Spieler eingeben. Mindestens drei Spieler muss ins Spiel teilnehmen und maximal 6 (Weil das Spielfeld wird endlich groß).

Die Art von Spielern bestimmt die Körpertemperatur am Anfang. Ein Eskimo hat 5 ein Forscher hat nur 4 Einheit (Diese Temperatur kann später ab/zunehmen). Ein Eskimo kann

ein Iglu bilden, wo er (und die anderen auf das “Iglu Platte”) das Schneesturm ohne Temperaturverlust überleben kann. Iglo verschwindet in den Schneesturm. Ein Forscher kann die Stabilität eines benachbarten Eisplatten sehen. Beide Tätigkeiten kosten eine einheitliche Arbeit. Man kann das Spiel mit Maus und Tastatur kontrollieren. In einem Rund bekommt jeder Spieler die Möglichkeit, um etwas zu machen.

Das Eisfeld besteht aus Eisplatten (6x6 Platte). Die Spieler sehen die ganze Landkarte von oben. Eisplatten sind Rechtecken, mit 4 Nachbaren (am Rand natürlich weniger). Eine Eisplatte kann entweder stabil oder instabil sein. Auf die stabilen Eisplatten können unendlich viele Spieler stehen, aber auf die instabile Eisplatten können nur bestimmt viele Spieler stehen. Wenn mehr Spieler auf die Eisplatte steht als es vorgeschrieben ist, die Platte wird überschlagen und die Spieler, die auf diese Platte stehen, fallen ins Wasser. Die Position von stabilen und instabilen Platten ist zufällig aber das Verhältnis ist fest. Die Menge von Schnee auf die Platten wird zufällig zwischen 0 und 4 Einheit sein.

In der verschiedenen Eisplatten werden unterschiedliche Gegenstände frieren, wie Schaufel, Seil, Taucheranzug, Essen, Pistole, Leuchte, Patrone. In einer Eisplatte befindet sich maximal ein Gegenstand. Die Spieler können diese Gegenstände sehen und ausgraben, wo sie stehen, wenn die bestimmte Eisplatte klar ist, also befindet sich kein Schnee darauf. Wenn ein Spieler neben seinen Gegenstand ein anderer Gegenstand ausgräbt (1 Arbeit), wirft der älteren Gegenstand auf den Boden. Der Spieler (oder die anderen) kann es später wieder aufheben (1 Arbeit), das Objekt ist immer auf dem Boden, nicht in dem Eis. Die Spieler haben kein Inventar und sie können einen Gegenstand direkt nicht abwerfen. So sie können die verschiedenen Gegenstände einander nicht weitergeben. Es ist zweckmäßig einige Objekten sofort benutzen, das Taucheranzug anziehen oder das Essen verzehren.

Auf die Platten kann schneebedecktes Loch sein. Wenn ein Spieler Taucheranzug trägt oder befindet sich ein Freund mit einem Seil auf die benachbarten Eisplatte (vor seinem nächsten Runde kommt), überlebt er das Fallen. Die Spieler können ihren Kumpel vor ihrer Runde retten. Ein Spieler, der Taucheranzug trägt, bleibt im Wasser bis nächste Runde und danach muss er seine erste Arbeit zu dem Tritt auf eine andere Eisplatte verwenden.

Alle Spieler kann in ihre Runde maximum 4 einheitliche Arbeit leisten, also es ist möglich, weniger als 4 einheitliche Arbeit leisten. Solche Tätigkeiten sind: eine Einheit Schnee von Platte entfernen (1 Arbeit), einen gegrabenen Gegenstand aufnehmen (1 Arbeit), ins Wasser gefallene Spieler retten oder auf eine benachbarten Eisplatte treten (1 Arbeit). Mit einer Schaufel kann man 2 Einheit Schnee räumen. Die Spieler können sich ihre maximum Körpertemperatur übertreten.

Wenn ein Spieler Essen gefunden hat, wird seine Körpertemperatur mit einer Einheit erhöht. Wenn ein Spieler in einem Schneesturm gerate, dann wird seine Temperatur mit einer Einheit sinken. Ein Schneesturm entsteht nur auf ein paar Feld. Der Zeitpunkt/Häufigkeit der Schneestürme ist zufallsartig. Ein Spieler stirbt, falls seine Körpertemperatur auf 0 sinkt. Er stirbt auch, wenn er ohne Taucheranzug ins Wasser fällt und wird nicht gerettet vor sein Runde. Es ist ein kooperatives Spiel, deswegen muss jeder Spieler bis das Ende des Spieles leben.

Es gibt zwei mögliche Weg dieses Gesellschaftsspiel beenden: entweder jemand stirbt oder alle Spieler gehen auf eine Eisplatte mit den benötigten Gegenständen (Pistole, Leuchte, Patrone) und leisten eine einheitliche Arbeit. Im letzteren Fall das Team gewinnt sonst nicht.

1.2.3. Benutzern

Benutzern können den fundamentalen Schritten des Spieles sehr schnell erlernen, man braucht nur eine kleine Computerkenntnisse. Gleichzeitig kann ins Spiel 3-6 Spieler teilnehmen (lokal, auf demselben Rechner).

1.2.4. Beschränkungen

Quellenkode muss auf die Maschinen der Teilnehmer mit standardisierte JDK laufen. (Weil diese Computer sind für Demonstration benutzt.)

1.2.5. Voraussetzungen, Beziehungen

Die ausgegebene Aufgabe: <https://www.iit.bme.hu/targyak/BMEVIIIA02/feladat>

1.3. Anforderungen

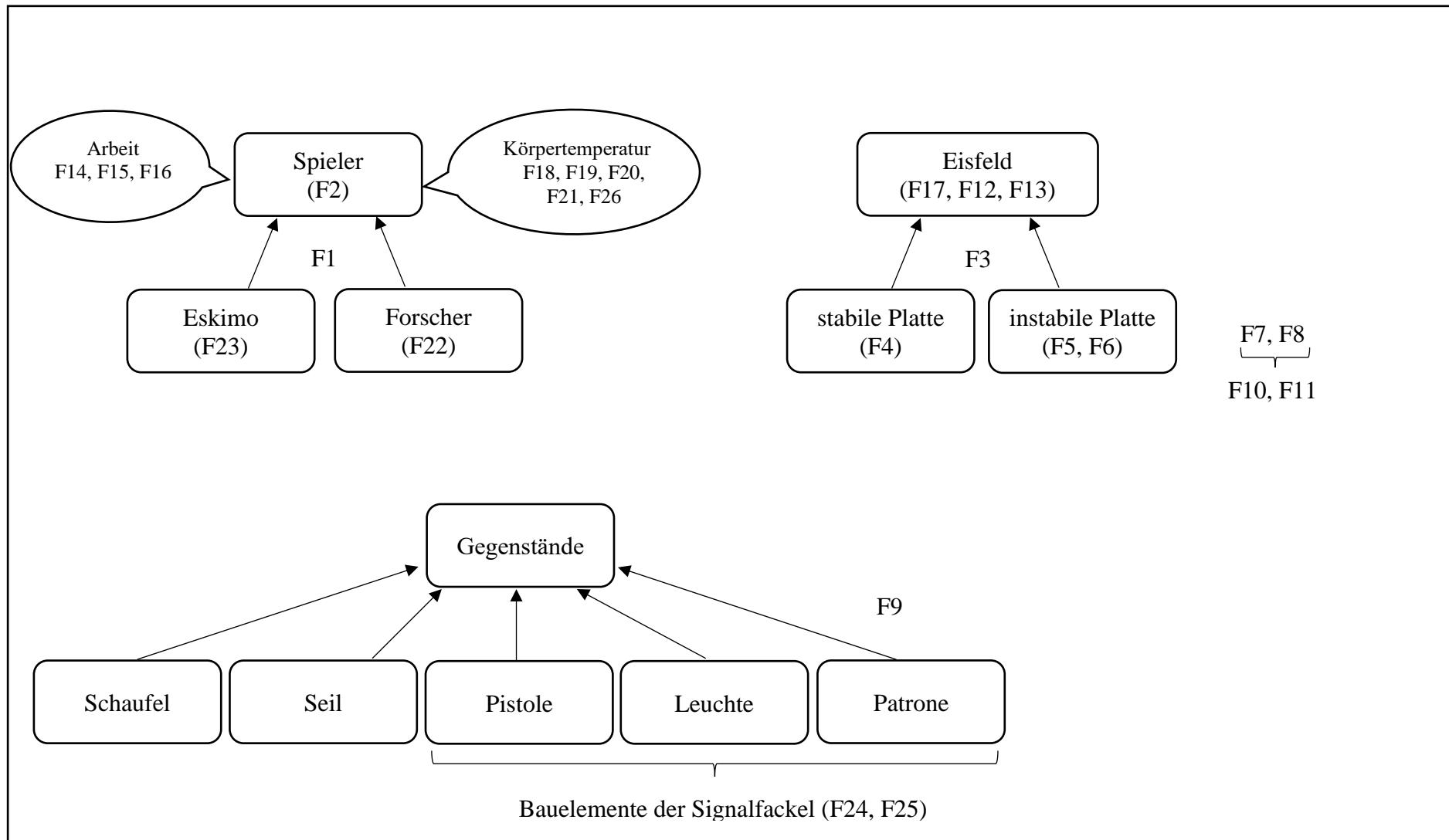
1.3.1. Funktionale Anforderungen

ID	Beschreibung	Check	Priorität	Quelle	Use-case	Komment
F1	<i>Ein Spieler kann entweder Eskimo oder Forscher sein</i>		grundsätzlich	IIT	Spiel steuern	
F2	<i>Mindestens gibt's 3 Spieler</i>		grundsätzlich	IIT	Spiel steuern	
F3	<i>Eisfeld besteht aus stabilen und instabilen Eisplatten</i>		grundsätzlich	IIT	Spiel steuern	Zusammenhang mit A4
F4	<i>Auf stabilen Eisplatten können beliebige Anzahl von Spieler stehen</i>		grundsätzlich	IIT	Spiel steuern	
F5	<i>Auf instabilen Platten kann nur eine beschränkte Anzahl von Spieler aufhalten</i>		grundsätzlich	IIT	Spiel steuern	
F6	<i>Wenn die Anzahl von Spielern auf instabilen Platten überschreitet wird, dann fallen die Spieler ins Wasser</i>		grundsätzlich	IIT	Spiel steuern, Überprüfen	
F7	<i>Die Platten haben eine zufällige Menge von Schnee</i>		wichtig	IIT	Spiel steuern	Zusammenhang mit A5
F8	<i>Die Platten können eingefrorene Gegenstände beinhalten</i>		grundsätzlich	IIT	Einheit von Arbeit leisten, Spiel steuern	
F9	<i>Die eingefrorenen Gegenstände sind: Schaufel, Seil, Taucheranzug, Essen, Pistole, Leuchte, Patrone</i>		grundsätzlich	IIT	Spiel steuern	Spezifizierung von F8
F10	<i>Man kann einen Gegenstand nur dann anschauen, wenn die Platte nicht mit Schnee bedeckt ist</i>		grundsätzlich	IIT	Spiel steuern	
F11	<i>Man kann einen Gegenstand nur dann aufnehmen, wenn die Platte nicht mit Schnee bedeckt ist</i>		grundsätzlich	IIT	Spiel steuern	
F12	<i>Zwischen den Platten können mit Schnee bedeckte Löcher vorhanden sein</i>		wichtig	IIT	Spiel steuern	

F13	<i>Wenn ein Spieler Taucheranzug trägt oder befindet sich ein Freund mit einem Seil auf die benachbarten Eisplatte, überlebt er das Fallen ins Wasser</i>		wichtig	IIT	<i>Spiel steuern Überprüfen</i>	
F14	<i>Jeder Spieler kann in einer Runde 4 Einheit von Arbeit leisten</i>		grundsätzlich	IIT	<i>Einheit von Arbeit leisten, Weitergeben</i>	<i>Zusammenhang mit A10</i>
F15	<i>Einheit von Arbeit ist: eine Einheit Schnee von Platte entfernen, auf eine benachbarte Eisplatte treten, einen eingefrorenen Gegenstand aufnehmen, Verwendung einer Spezialfähigkeit</i>		grundsätzlich	IIT	<i>Einheit von Arbeit leisten</i>	<i>Spezifizierung von F14</i>
F16	<i>Mithilfe von Schaufel kann man 2 Einheit von Schnee mit einer Einheit von Arbeit leisten</i>		wichtig	IIT	<i>Einheit von Arbeit leisten</i>	<i>Spezifizierung von F15</i>
F17	<i>Auf das Eisfeld kann zufällig Schneesturm ankommen, damit einige Platten mit neuer Einheit von Schnee bedeckt werden</i>		wichtig	IIT	<i>Spiel steuern</i>	
F18	<i>Wenn Schneesturm eine solche Eisplatte mit Schnee bedecken würde, wo Spieler steht/stehen, Körpertemperatur von Spieler(n) wird mit einer Einheit verringert</i>		wichtig	IIT	<i>Spiel steuern, Überprüfen</i>	
F19	<i>Eskimos haben 5 Einheit von Körpertemperatur am Anfang</i>		wichtig	IIT	<i>Spiel steuern</i>	
F20	<i>Forscher haben 4 Einheit von Körpertemperatur am Anfang</i>		wichtig	IIT	<i>Spiel steuern</i>	
F21	<i>Essen erhöht Körpertemperatur mit einer Einheit</i>		wichtig	IIT	<i>Spiel steuern</i>	
F22	<i>Forscher können die Kapazität von einer Platte erkennen</i>		wichtig	IIT	<i>Einheit von Arbeit leisten</i>	
F23	<i>Eskimos können ein Iglu bauen, um die Schneestürme überleben</i>		wichtig	IIT	<i>Einheit von Arbeit leisten</i>	
F24	<i>Spieler sollen die Gegenstände von Signalfackel sammeln</i>		grundsätzlich	IIT	<i>Einheit von Arbeit leisten</i>	

F25	Zum Aufbau der Signalfackel sollen alle Team - Member auf einer Platte sein und sollen alle Gegenstände von Signalfackel (Pistole, Leuchte, Patrone) vorhanden sein und Team braucht eine Einheit von Arbeit		wichtig	IIT	Spiel steuern	
F26	Wenn jemand im Wasser stirbt oder abkühlt (Körpertemperatur= 0), ist Ende des Spieles (Fehlschlag)		wichtig	IIT	Spiel steuern	
F27	Wenn die Signalfackel gebaut wurde, dann ist Ende des Spieles (Sieg)		grundsätzlich	IIT	Spiel steuern	
F28	Iglo verschwindet in den Schneesturm.		wichtig	Team	Spiel steuern	
F29	Ein Platte hat 4 benachbarte Platte (maximal, am Rand natürlich weniger)		wichtig	Team	Spiel steuern	
F30	Ein Forscher kann die Kapazität von seiner Platte sehen		wichtig	Team	Spiel steuern	
F31	Ein Forscher kann die Kapazität den benachbarten Platten auch sehen		wichtig	Team	Spiel steuern	

Beziehungen der funktionalen Anforderungen



1.3.2. Ressourceanforderungen

ID	Beschreibung	Check	Priorität	Quelle	Komment
1	<i>IntelliJ IDEA</i>		wichtig	Team	<i>Java IDE</i>
2	<i>Github</i>		wichtig	Team	<i>Versionskontrollsysteme</i>
3	<i>OneDrive</i>		wichtig	Team	<i>Online Speicherplatz für Dokumentation</i>
4	<i>Trello</i>		wichtig	Team	<i>Zeit- und Aufgabenmanager</i>
5	<i>Geräte (Laptop, Computer, Monitor...)</i>		grundsätzlich	Team	<i>Arbeitsgerät</i>
6	<i>Messenger</i>		wichtig	Team	<i>Kommunikationsplattform</i>
7	<i>WhiteStarUML</i>		wichtig	Team	<i>UML Modellierer</i>
8	<i>PlantUML</i>		wichtig	Team	<i>UML Modellierer</i>
9	<i>Microsoft Office Online</i> Aber: <i>Microsoft Pro-Plus empfehlenswert</i>		wichtig	Team	

1.3.3. Übergabeanforderungen

ID	Beschreibung	Check	Priorität	Quelle	Komment
1	<i>Anforderungen, Projekt, Funktionalität Dokumentation</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 17. Februar</i>
2	<i>Analyse Modell 1.</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 26. Februar</i>
3	<i>Analyse Modell 2.</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 4. März</i>
5	<i>Skeleton eingaben, präsentieren</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 18. März</i>
6	<i>Prototyp Konzept</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 25. März</i>
7	<i>Detaillierte Pläne</i>	Präsentation	grundsätzlich	Besteller	<i>Präsentation am 1. April</i>

8	<i>Prototyp machen, testen</i>	Präsentation	grundsätzlich	Besteller	Präsentation am 8. April
9	<i>Prototyp und Testfälle eingeben, präsentieren</i>	Präsentation	grundsätzlich	Besteller	Präsentation am 22. April
10	<i>GUI Spezifikation</i>	Präsentation	grundsätzlich	Besteller	Präsentation am 29. April
11	<i>GUI entwickeln</i>	Präsentation	grundsätzlich	Besteller	Präsentation am 6. Mai
12	<i>Projekt mit GUI und Zusammenfassung eingeben, präsentieren</i>	Präsentation	grundsätzlich	Besteller	Präsentation am 13. Mai

1.3.4. Andere nicht funktionale Anforderungen

ID	Beschreibung	Check	Priorität	Quelle	Komment
A1	<i>Das Spiel funktioniert lokal auf einer Maschine</i>		wichtig	Team	
A2	<i>Alle Spieler gehören zu einem Team</i>		wichtig	Team	
A3	<i>Am Anfang des Spieles können die Spieler ihre Art (Eskimo oder Forscher) nicht auswählen (zufällig aufgeteilt)</i>		optional	Team	
A4	<i>Position von stabilen und instabilen Platten ist zufällig, aber das Verhältnis ist fest</i>		wichtig	Team	Zusammenhang mit
A5	<i>Auf die Platten wird zufällig zwischen 1 und 4 Einheit von Schnee sein</i>		wichtig	Team	Zusammenhang mit F7
A6	<i>Wenn die Platte nicht mit Schnee bedeckt ist, dann kann man die Gegenstände ausgraben</i>		optional	Team	
A7	<i>Wenn ein Spieler neben seinen Gegenstand ein anderer Gegenstand ausgräbt, wirft der älteren Gegenstand auf den Boden</i>		optional	Team	
A8	<i>Die Spieler haben kein Inventar und sie können einen Gegenstand nicht direkt abwerfen</i>		optional	Team	
A9	<i>Spieler können die Gegenstände einander nicht weitergeben</i>		optional	Team	

A10	<i>Spieler leisten maximal 4 Einheit von Arbeit, also es ist möglich weniger als 4 Einheit von Arbeit pro Runde leisten</i>		<i>optional</i>	<i>Team</i>	<i>Zusammenhang mit F14</i>
A11	<i>Die Spieler können sich ihre maximum Körpertemperatur übertreten</i>		<i>optional</i>	<i>Team</i>	
A12	<i>Wenn man ins Wasser ohne Taucheranzug fällt und wird nicht gerettet vor seiner Runde, dann stirbt er</i>		<i>optional</i>	<i>Team</i>	
A13	<i>Ein Spieler, der ins Wasser gefallen ist und Taucheranzug getragen hat, bleibt im Wasser bis nächste Runde und danach muss er seine erste Arbeit zu dem Tritt auf eine andere Eisplatte verwenden</i>		<i>optional</i>	<i>Team</i>	
A14	<i>Einen Gegenstand wieder aufheben kostet auch eine Einheit von Arbeit</i>		<i>optional</i>	<i>Team</i>	
A15	<i>Das Eisfeld ist 6x6 groß</i>		<i>optional</i>	<i>Team</i>	
A16	<i>Es gibt maximal 6 Spieler, weil das Spielfeld endlich groß ist</i>		<i>optional</i>	<i>Team</i>	
A17	<i>Das Spiel lässt sich mit Maus und Tastatur steuert werden</i>		<i>optional</i>	<i>Team</i>	

1.4. Wesentliche use-case-n

1.4.1. Use-case Beschreibungen

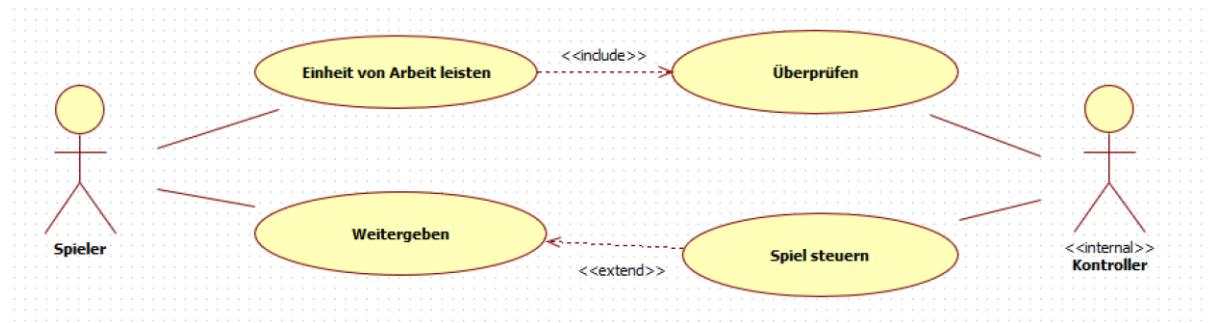
Name von Use-case	Einheit von Arbeit leisten
Kurze Beschreibung	Spieler leistet eine Einheit von Arbeit
Akteur	Spieler
Haupttätigkeit	<ul style="list-style-type: none"> 1. Der Spieler leistet eine Einheit von Arbeit Einheit von Arbeit ist: eine Einheit Schnee von Platte entfernen, auf eine benachbarte Eisplatte treten, einen eingefrorenen Gegenstand aufnehmen, Verwendung einer Spezialfähigkeit

Name von Use-case	Weitergeben
Kurze Beschreibung	Spieler gibt das Spiel weiter
Akteur	Spieler
Haupttätigkeit	<ul style="list-style-type: none"> 1. Der Spieler benutzt vollständig seinen 4 Einheit von Arbeit 2. Das System speichert, dass der Spieler seine Arbeit geleistet hat
Alternative Tätigkeit	<ul style="list-style-type: none"> 1.A.1. Der Spieler gibt das Spiel weiter (Arbeit übrig), sagt „ÜBERGEBEN“ 1.A.2. Das System speichert die Anfrage von Spieler 1.A.3. Das System speichert, dass der Spieler seine Arbeit geleistet hat

Name von Use-case	Überprüfen
Kurze Beschreibung	Kontroller überprüft das Spiel
Akteur	Kontroller
Haupttätigkeit	<ul style="list-style-type: none"> 1. Das System hat alles überprüft
Alternative Tätigkeit	<ul style="list-style-type: none"> 1.A.1. Team hat die Signalfackel aufgebaut 1.A.2. Das System benachrichtigt die Spieler über den Gewinn des Spieles 1.B.1. Jemand abkühlt, also Körpertemperatur eines Spielers ist 0. 1.B.2. Das System benachrichtigt die Spieler über den Verlust des Spieles 1.C.1. Jemand stirbt im Wasser 1.C.2. Das System benachrichtigt die Spieler über den Verlust des Spieles

Name von Use-case	Spiel steuern
Kurze Beschreibung	Kontroller steuert & koordiniert das Spiel
Akteur	Kontroller
Haupttätigkeit	1. Kontroller wählt den neuen Spieler aus 2. Das System zeigt den aktiven Spieler
Alternative Tätigkeit	1.A.1. Das System generiert einen Schneesturm

1.4.2. Use-case Diagramm



1.5. Wörterbuch

Ungarisch	Englisch	Deutsch	Definition
jégmező	Icefield	Eisfeld	Die ganze “Spielplatz”, umzingelt mit dem Meer
Tenger	See	Meer/See	Meer, “Hintergrund”
stabil (Jég)tábla	Stable (Ice)tile	Stabile Platte	Hat unendliche Kapazität
instabil (Jég)tábla	Unstable (Ice)tile	Instabile Platte	Hat ein nicht unendliche Kapazität
havas Luk	Snowy hole	Schneebedecktes Loch	Hat Nullkapazität
havas Tábla	Snowy tile	Schneebedeckte Platte	Man kann die möglichst hier liegender Gegenstand auf diese Platte nicht sehen
Játékos/felhasználó	Player/user	Spieler/Benutzer	
Eszkimó	Eskimo	Eskimo	Spielertyp 1
Sarkkutató	Researcher	Forscher	Spielertyp 2
Tárgy	Item	Gegenstand	Siehe unten
Lapát	Shovel	Schaufel	Damit kann man mit 1 Arbeit 2 Einheit schneeräumen
Kötél	Rope	Seil	Damit kann man eine andere schnell von Wasser retten
Búvárruha	Diving Suit	Taucheranzug	Damit überlebt man das Wasser
Élelem	Food	Essen	Gibt +1 Körpertemperatur
Kézben tartott tárgy	Item in hand	im Hand gehalteten Gegenstand	Gleichzeitig max. 1
Ledobni egy tárgyat a földre	to throw an item to the ground	Einen Gegenstand auf den Boden werfen	Ein Spieler kann gleichzeitig max. 1 Gegenstand im Hand halten. Falls der Spieler nimmt ein Gegenstand auf, dann wirft er/sie die andere in der Hand
Munka (egység)	Work (unit)	Arbeit (Einheit)	4/Runde
Ásás	Shoveling	graben	Effektiveres schneeräumen (2 Einheit pro 1 Arbeit)
Lépés	Step	Schritt	Auf eine benachbarte Platte bewegen, kostet 1 Arbeit
Mentés	Save (someone)	(jemand) retten	Mithilfe der Seile, 1 Arbeit

Tárgyat felvenni	to Pick up an item	Einen Gegenstand aufnehmen	Falls es schone ausgegrabelt ist, 1 Arbeit
Vízbe esni	to fall in water	Im Wasser fallen	Wegen eins Loches/instabile Platte
Hó eltakarítás	to clear away snow	schneeräumen	Wenn das Spieler auf eine schneebedeckte Platte steht/es gibt schneebedeckte Nachbarplatten, kostet 1 Arbeit/1 Schnee Einheit oder 1 A./2 Sch.E., falls man eine Schaufel hat
Meghal	to die	sterben	
Fulladás	to suffocate	ersticken	
Kihűlés	freeze to death	erfrieren	
Túlél	Survive	überleben	
Speciális képesség	Special abilities	Speziale Fähigkeiten	Eskimos und Forscher haben verschiedene ~. ~ Kosten 1 Arbeit
Megnézi a tábla kapacitását	To detect the capacity of a tile	Kapazität von einer Platte erkennen	Die speziale Fähigkeit von Forschern
Jégtábla kapacitása	Capacity of a tile	Kapazität von Platte	Nach wie viele Menschen wendet eine instabile Platte herunter.
Iglu építés	to build an igloo	Ein Iglu bauen	Ein Eskimo kann das tun. Schützt von Schneesturm (siehe unten)
Testhő	Bodyheat	Körpertemperatur	“Gesundheitspunkte”/”Health”, im Anfang 4 oder 5, kann mehr sein.
Hóvihar	Snowstorm	Schneesturm	Bringt frischer Schnee. Es kann Spieler töten.
Jelzőrakéta	Signal flare	Signalfackel	Das Ziel dieses Spiel diese Fackel zusammenzubauen
Pisztoly	Gun	Pistole	Ein Teil der Signalfackel
Jelzőfény	Beacon-light	Leucht	Ein Teil der Signalfackel
Patron	Cartridge	Patrone	Ein Teil der Signalfackel
Összeszerelés és Elsütés	to assemble and to fire	Zusammenbauen und feuern	Wenn die Spieler mit der drei Teile der Signalfackel sind auf dieselbe Platte, dann können sie mit diesem Schritt das Spiel gewinnen. (Kostet 1 Arbeit)

Győzelem	Victory	Sieg	Wenn das Signalfackel ist zusammengebaut und gefeuert, dann die Spieler siegen
Vereség	Failure	Fehlschlag	Wenn jemand stirbt, dann jede verliert das Spiel
Kör	Round	Runde	In seinem Runde können Spieler 4 Arbeit spenden und damit etwas machen

1.6. Projekt Plan

1.6.1. Schritte des Projekts und Fristen

Aufgabe	Frist
Anforderungen, Projekt, Funktionalität Dokumentation	17. Februar
Analyse Modell 1.	26. Februar
Analyse Modell 2.	4. März
Skeleton Pläne	11. März
Skeleton eingaben, präsentieren	18. März
Prototyp Konzept	25. März
Detaillierte Pläne	1. April
Prototyp machen, testen	8. April
Prototyp und Testfälle eingeben, präsentieren	22. April
GUI Spezifikation	29. April
GUI entwickeln	6. Mai
Projekt mit GUI und Zusammenfassung eingeben, präsentieren	13. Mai

1.6.2. Benutzte Software-Tools

Die Programmcode wird im IntelliJ IDEA entwickelt. Die Versionskontrolle und die Verteilung von Sourcecode wird mithilfe von einer Github Repository verwirklicht (Linke in 2.1.4). Die Dokumentation speichern wir Online in einen OneDrive Verzeichnis. Die Dokumente erstellen und bearbeiten wir in Microsoft Office Online, so können alle über dieselbe Dokument arbeiten. Wir benutzen Trello für die Management der Aufgaben. Zur Kommunikation in dem Team benutzen wir einen Messenger-Gruppe. Für Herstellung der UML-Diagramme benutzten wir WhiteStarUML und PlantUML.

1.6.3. Aufbau des Teams

Das Team besteht aus fünf Personen:

Name	Aufgaben
Ádám Zsófia	Code entwickeln, Testen, Dokumentation, UML entwickeln
Hedrich Ádám	Code entwickeln, Testen, Dokumentation, UML entwickeln
Pintér Balázs	Teamleiter, Code entwickeln, Testen, Dokumentation, UML entwickeln
Fucskár Patrícia	Code entwickeln, Testen, Dokumentation, UML entwickeln
Tassi Timián	Code entwickeln, Testen, Dokumentation, UML entwickeln

Denn in IntelliJ IDEA ist es sehr einfach, aus den Code UML Diagramme und Dokumentation zu erzeugen, haben wir diese Aufgaben nicht aufgeteilt, jedes Mitglied wird seine eigene Codeteile selbst dokumentieren und die UML Diagramme erzeugen. Jedes Mitglied wird auch beim Präsentationen teilnehmen.

3. Entwicklung des Analysemodells

3.1 Objektkatalog

3.1.1 Players (Eskimos und Researchers)

Diese Objekte sind für die Zustände und Zustandsänderungen des Spielers verantwortlich. Sie speichern, ob die Spieler etwas im Hand haben, durch diese können die Benutzer Tätigkeiten initiieren und sie senden Signale über Sieg/Fehlschlag.

3.1.2 Platten

Sie sind für die Bewegung des Spielers verantwortlich (Sp. nehmen und geben). Sie speichern Positionen, sie können ihre Nachbarn erreichen und sie können entwerten, ob es zu viele Spieler auf sich stehen oder nicht (Kapazitäten). Es gibt 3 Typen (Loch, stabil instabil), die sich anders verhalten, wie es in Aufgabe gegeben war.

3.1.3 Gegenstände (mehrere vererbte “Typen”)

Es gibt mehrere Typen dieser Objekte. Von jedem Typen gibt es immer ein (*Wenn jemand ein Essen gegessen hat, dann wird ein neues gefrorenes Essen generiert*). Sie haben drei Zustände: gefrieren, geworfen und in der Hand, es ist in einem Enum. Diese Gegenstände erlauben verschiedenen Tätigkeiten (*falls jemand ein Essen in der Hand hat, kann er dann Essen; mit Schaufel effizienter graben; usw.*)

Gegenstandstypen:

- Essen (Food)
- Seil (Rope)
- Schaufel (Shovel)
- Taucheranzug (DivingSuit)
- SignalFackel Elemente (Signal Flare parts)

3.1.4 Schneesturm

Der Schneesturm ist verantwortlich für die Schneestürme (sehr überraschend).

Es hat immer eine Chance nach jedem Rund für ein Schneesturm, dieser Zufall wird bei dem Schneesturm “gerechnet”. Falls es kommt, dann es lost die “stürmige” Platten aus und kommuniziert mit dem Spieler und die Platten über den neuen Schnee und die Spieler Verletzungen. Es zerstört die Iglus.

3.1.5 RoundController

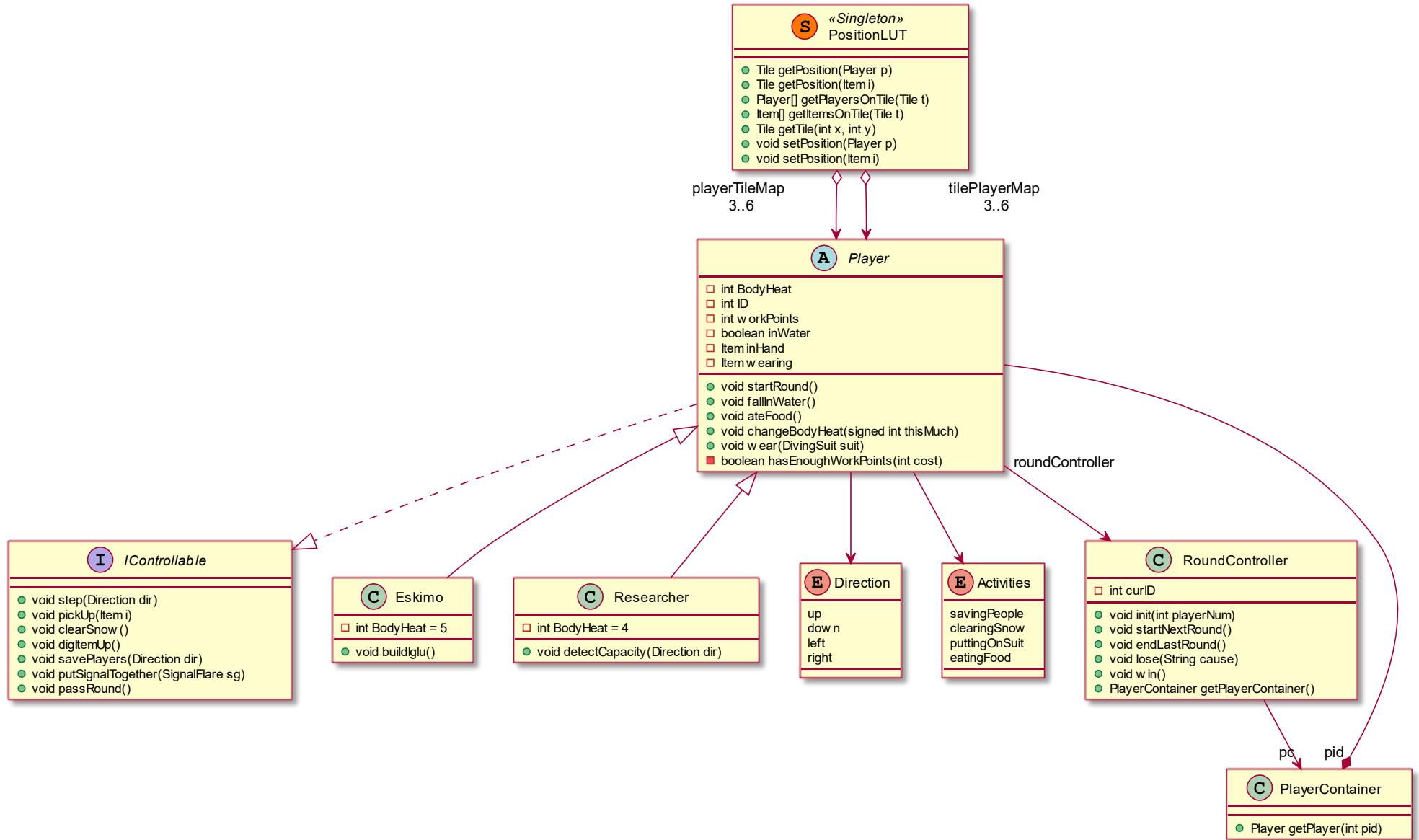
Dieses einzige Objekt wird für Initiierung und Laufen des Spiels verantwortlich. Es wird nach der Anzahl der Spieler fragen und dann die anderen Objekte initialisieren. Es macht die gebrauchte „checks“ und „cleanup“/Initiierung zwischen die Runde der Spieler und behandelt Sieg/Fehlschlag.

3.1.6 SignalFlare

Dieses Objekt speichert die 3 Signalflacke Teile (Komposition), und kann zusammengebaut werden, falls die Umstände genügend sind.

3.2 Klassendiagramme

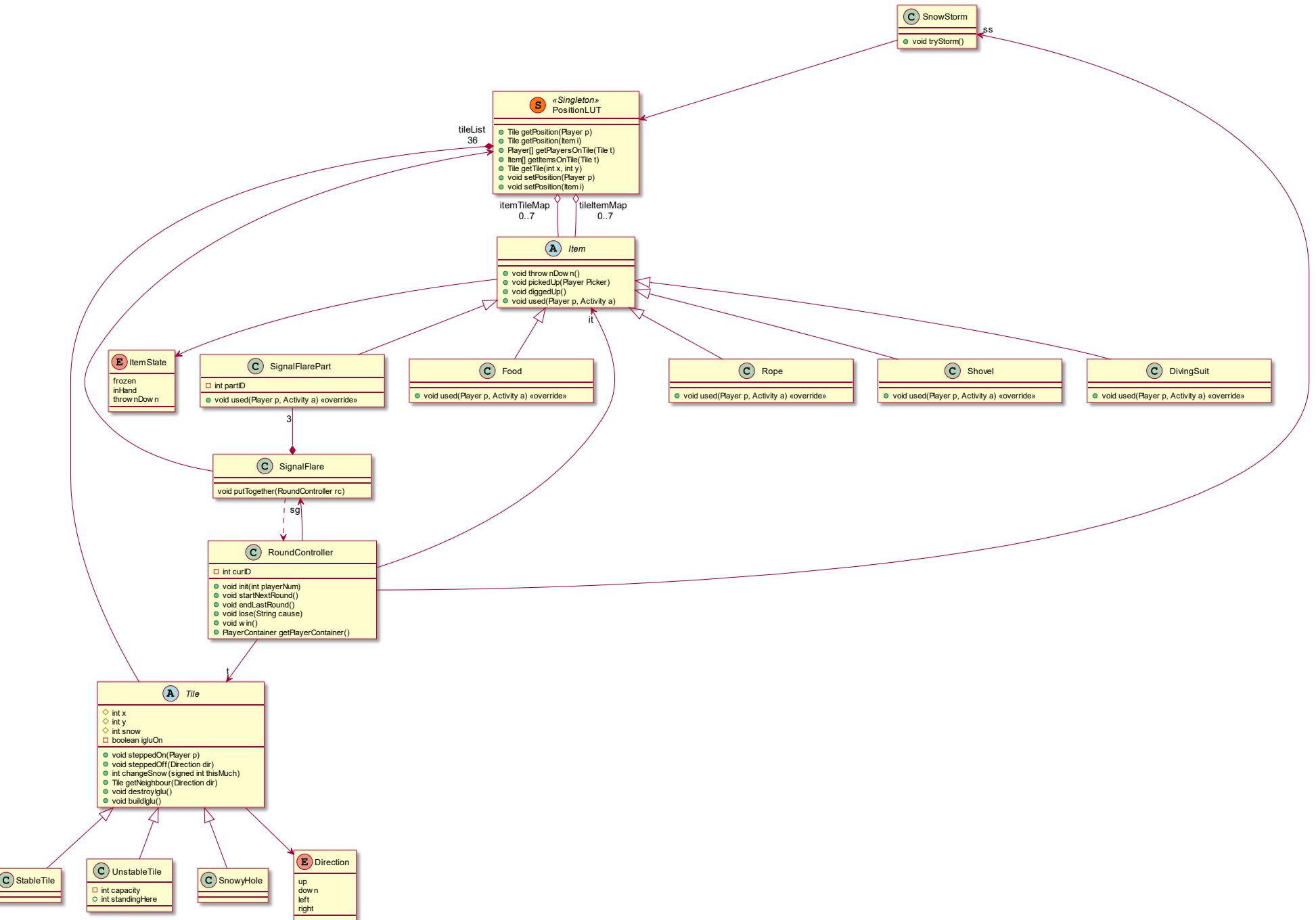
Spieler - Class Diagramm Teil 1



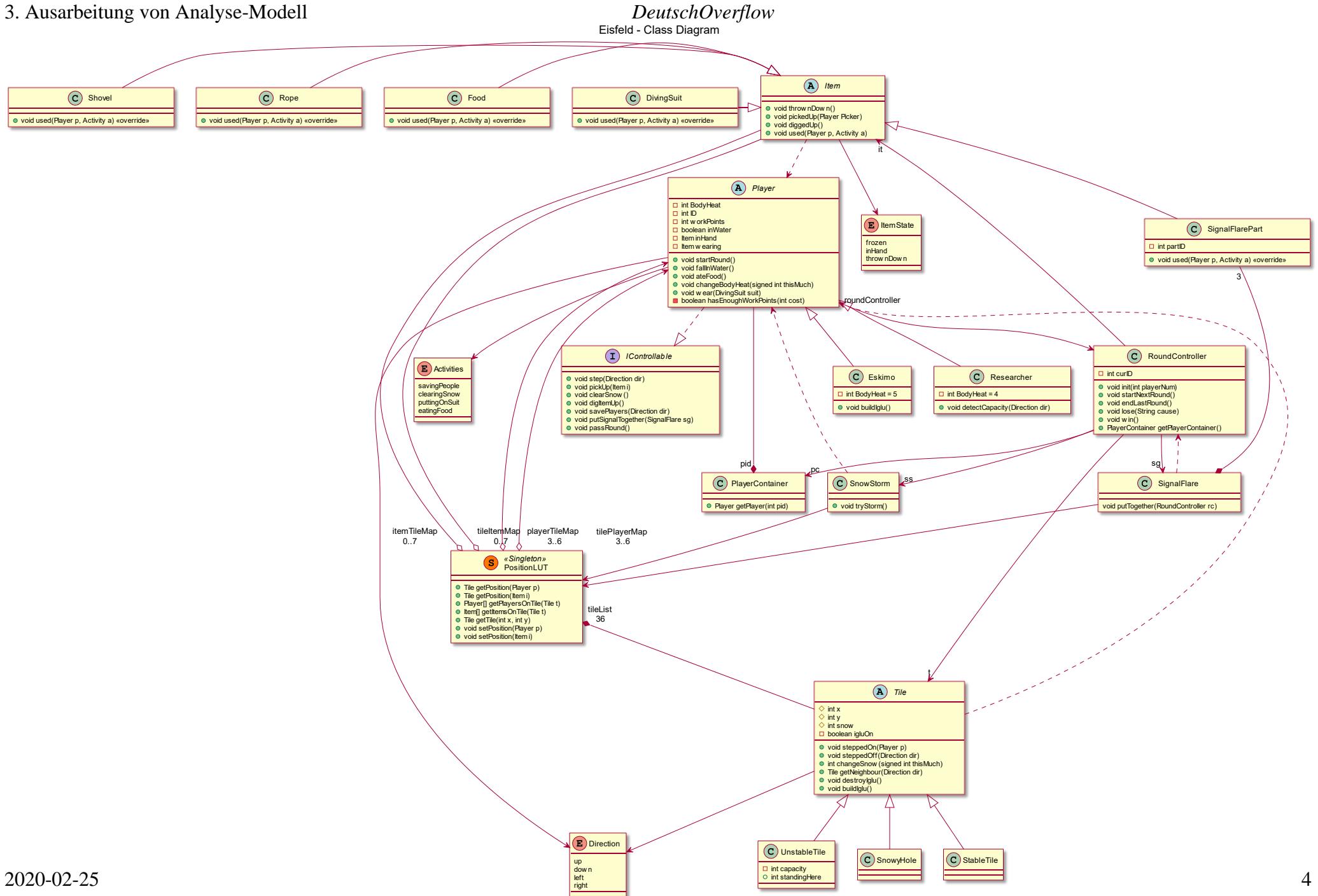
3. Ausarbeitung von Analyse-Modell

DeutschOverflow

Felder, Gegenstände, Schneesturm - Class Diagram Teil2



3. Ausarbeitung von Analyse-Modell



3.3 Beschreibung der Klassen

3.3.1 Player

- Verantwortung

Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanzieren, weil ein Spieler entweder Eskimo oder Forscher ist. Diese Klassen enthält die grundsätzlichen Eigenschaften (Attributen) und die gemeinsamen Methoden von beiden Art von Spielern. Natürlich die Eskimos und Forscher Klasse spezialisiert diese Klasse, also enthält weitere Methoden.

- Superklasse

-

- Schnittstellen

- IControllable

- Attributen

- -int **Bodyheat**: Die Körpertemperatur des Spielers. Es ist 5 Einheit für Eskimos und 4 Einheit für Forscher.
- -int **ID**: für eindeutige Charakterisierung des Spielers
- -int **workPoints**: Der Spieler hat so viele Arbeit (Einheit) noch.
- -Item **inHand**: Der Gegenstand, der in der Hand des Spielers sich befindet.
- -DivingSuit **wearing**: Es ist eigentlich der Taucheranzug, wenn der Spieler es trägt.
- -boolean **inWater**: Es zeigt, ob der Spieler im Wasser ist. (true: Ja, false: Nein)
- -RoundController **roundController**: Es ist zu dem Spieler gehörende RoundController.

- Methoden

- +void **startRound()**: Mit dieser Methode startet ein Spieler der Round.
- +void **fallInWater()**: Der Spieler fällt ins Wasser. Er wartet auf die Rettung.
- +void **changeBodyHeat(signed int thisMuch)**: Der Spieler kann sich seine Körpertemperatur erhöhen (mit dem Essen) oder es wird durch eine Schneesturm verringert.
- +void **AteFood()**: Der Spieler isst sein Essen, so seine Körpertemperatur wird sich erhöht.
- +void **wear(DivingSuit suit)**: Der Spieler trägt den Taucheranzug. Wenn er ins Wasser fällt, wird nicht gestorben.
- -boolean **hasEnoughWorkPoints(int costs)**: Diese Methode gibt zurück, ob der Spieler die bestimmten Tätigkeit durchführen kann. Es hängt von dem gebliebenen Arbeit Einheit ab. (true: ja, false: nein)

3.3.2 IControllable

- Verantwortung

Diese Klasse ist eine Schnittstelle. Einige Methoden befinden sich hier. Diese Methoden werden durch Klasse "Player" benutzt: Treten, Gegenstand aufnehmen, schneeräumen, Gegenstand ausgraben, Teile vom Signalfackel zusammenbauen, Runde passen.

- Superklasse

-

- Schnittstellen

- **Attributen**
 -

- **Methoden**
 - **+void passRound()**: Mit dieser Methode kann der Spieler passen, also das Recht für Tritt weitergeben.
 - **+void digItemUp()**: Der Spieler räumt mit Hilfe von einem Gräber Schnee, wenn er ein Gräber hat.
 - **+void pickUp(Item i)**: Der Spieler kann einen Gegenstand aufnehmen, wenn der Gegenstand in die Eisplatte nicht einfriert.
 - **+void clearSnow()**: Der Spieler macht eine Eisplatte sauber.
 - **+void step(Direction dir)**: Der Spieler kann in bestimmten Richtungen treten. (auf eine andere Eisplatte) Direction ist eine Enumeration, die die 4 Richtung beinhaltet.
 - **+void putSignalTogether(Signalflare sf)**: Am Ende des Spiels feuern alle Spieler die Signalfackel. Es ist möglich, wenn alle Spieler in einer Eisplatte stehen und die Teile von der Signalfackel da sind. Es kostet 1 Arbeit (Einheit).
 - **+void savePlayers(Direction dir)**: Der Spieler rettet sein Kumpel mit Hilfe vom Seil. Man muss eingeben, auf welche Eisplatte (Direction) wird der Spieler gerettet.

3.3.3 Eskimo

- **Verantwortung**

Diese Klasse spezialisiert die “Spieler” Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Superklasse, weil allen Attributen beiden Subklassen vorkommt, nur die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **Superklasse**

Player->Eskimo

- **Schnittstellen**
 -

- **Attributen**
 - **-int BodyHeat=5**: Die Eskimos haben am Anfang des Spiels 5 Einheit Körpertemperatur.

- **Methoden**
 - **+void buildIglu()**: Die Eskimos können ein Iglu bauen. Es schützt vor dem Schneesturm.

3.3.4 Researcher

- **Verantwortung**

Diese Klasse spezialisiert die “Spieler” Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann, als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Vorklasse, weil allen Attributen beiden Subklassen vorkommt, nur die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **Superklasse**

Player->Researcher

- **Schnittstellen**

-

- **Attributen**

- **-int BodyHeat=4:** Den Forschern haben am Anfang des Spiels 4 Einheit Körpertemperatur.

• **Methoden**

- **+void detectCapacity():** Die Forscher können mit dieser Methode anschauen, ob eine Eisplatte stabil oder instabil ist, also wie viel Spieler kann auf eine bestimmten Eisplatte stehen.

3.3.5 Item

- **Verantwortung**

Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanzieren. Es ist ein Container von verschiedenen Gegenständen. Die gemeinsame Eigenschaft von verschiedenen Gegenständen ist, dass es in dem Spieler immer 1 von jedem gibt.

- **Superklasse**

-

- **Schnittstellen**

-

- **Attributen**

• **Methoden**

- **+void thrownDown():** Der Gegenstand wird abgeworfen. In diesem Fall ist der Gegenstand in die Eisplatte nicht eingefroren und bleibt ebendort.
- **+void pickedUp(Player Picker):** Der Gegenstand wird durch ein bestimmten Spieler aufgenommen. Der Spieler wird im Parameter gegeben.
- **+void diggedUp():** Mit dieser Methode können wir den Gegenstand ausgraben.
- **+void used(Player player, Activity activity):** Die verschieden Gegenstände können durch ein bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten.

3.3.6 Shovel

- **Verantwortung**

Diese Klasse spezialisiert die "Item" Klasse. Diese Klasse verwirklicht den Schaufel Gegenstand.

- **Superklasse**

Item->Shovel

- **Schnittstellen**

-

- **Attributen**

-

- **Methoden**

- **+void used(Player player, Activity activity):** Die “*used*” Methode von der Superklasse (Item) wird überschrieben. (override)

3.3.7 Rope

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Seil Gegenstand.

- **Superklasse**

Item->Rope

- **Schnittstellen**

-

- **Attributen**

-

- **Methoden**

- **+void used(Player player, Activity activity):** Die “*used*” Methode von der Superklasse (Item) wird überschrieben. (override)

3.3.8 DivingSuit

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Taucheranzug Gegenstand.

- **Superklasse**

Item->DivingSuit

- **Schnittstellen**

-

- **Attributen**

-

- **Methoden**

- **+void used(Player player, Activity activity):** Die “*used*” Methode von der Superklasse (Item) wird überschrieben. (override)

3.3.9 Food

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Essen Gegenstand. Mit diesem Gegenstand können die Spieler sich ihre Körpertemperatur erhöhen.

- **Superklasse**

Item->Food

- **Schnittstellen**
 -
- **Attributen**
 -
- **Methoden**
 - **+void used(Player player, Activity activity):** Die “*used*” Methode von der Superklasse (Item) wird überschrieben. (override)

3.3.10 ItemState

- **Verantwortung**

Diese Klasse ist eine Enumeration. In dieser Enumeration befinden sich die Zustände von einem Gegenstand. Anderer Zustand existiert nicht. In einem Augenblick kann sich zu einem Gegenstand pünktlich ein Zustand gehören.

- **Superklasse**
 -
- **Schnittstellen**
 -
- **Attributen**
 - **frozen:** Wenn ein Gegenstand gefroren ist, gehört sich zu diesem Zustand.
 - **inHand:** Wenn ein Gegenstand sich in der Hand von einem Spieler befindet, gehört sich zu diesem Zustand.
 - **thrownDown:** Ein Gegenstand kann auf eine Eisplatte nicht in gefroren Zustand sein, also durch einen Spieler abgeworfen wird. In diesem Fall gehört sich zu diesem Zustand

3.3.10.1 Methoden

3.3.11 Activities

- **Verantwortung**

Diese Klasse ist eine Enumeration. Es ist gegeben, welche Aktivitäten ein Spieler durchführen kann. Diese Enumeration speichert diese Möglichkeiten. Andere Aktivitäten können während des Spiels nicht durchgeführt werden.

- **Superklasse**
 -
- **Schnittstellen**
 -
- **Attributen**
 - **+savingPeople:** Ein Spieler kann einen anderen Spieler von dem Wasser retten.
 - **+clearingSnow:** Die Spieler können die Eisplatten saubermachen, also der Schnee wird von der Eisplatte weggeputzt.

3. Ausarbeitung von Analyse-Modell

DeutschOverflow

- **+eatingFood:** Wenn ein Spieler sich mit einem Essen begegnet, isst automatisch es. Danach wird ein anderes Essen irgendwo erschien. Während dieser Tätigkeit wird sich die Körpertemperatur von dem bestimmten Spieler erhöht.
- **+PuttingOnSuit:** Der Spieler nimmt den Taucheranzug auf sich. Es wird automatisch durchgeführt, wenn ein Spieler einen Taucheranzug aufnehmen

• Methoden

3.3.12 RoundController

• Verantwortung

Diese Klasse steuert das Ende und der Beginn eines Spiels, also es ist ein Kontroller. Es beinhaltet ein Spieler Container. In diesem Container befinden sich den Spielern. Die Runden werden auch in dieser Klasse kontrolliert. Am Anfang des Spiels kann man durch diese Klasse den Spielern initialisieren.

• Superklasse

• Schnittstellen

• Attributen

- **-int curID:** Es speichert, welche Spieler kommt. Es speichert den Identifikationsnummer des Spielers.
- **+SignalFlare sg:** Es ist der Signalfackel als Attribut in dieser Klasse.
- **+Item it:** Es ist der Gegenstand als Attribut in dieser Klasse.
- **+PlayerContainer:** Es ist die Spielercontainer als Attribut in dieser Klasse.
- **+SnowStorm SS:** Es ist der Schneesturm als Attribut in dieser Klasse.
- **+Tile t:** Die Klasse *Gegenstand* befindet sich in dieser Klasse.

• Methoden

- **+void init(int playerNum):** Mit dieser Methode können wir einen Spieler initialisieren. Dazu müssen wir ein Spieler Nummer eingeben. Diese Nummern identifiziert eindeutig den Spielern, also müssen unterschiedlich sein.
- **+void nextRound():** Am Ende der Runde wird die nächste gerufen. Der erste Spieler der Round setzt das Spiel fort.
- **+void lose(String cause):** Das Spiel beendet. Der Grund der Niederlage wird eingegeben.

3.3.13 SignalFlare

• Verantwortung.

Diese Klasse ist eigentlich die Signalfackel. Dieser Gegenstand besteht aus 3 Teile: Pistole, Leuchte, Patron. Mit diesem kann die Spieler das Spiel gewinnen.

• Superklasse

• Schnittstellen

- **Attributen**
- **-SignalFlarePart sfp[3]:** Es ist ein Array und dieser Array beinhaltet die 3 Teile vom Signalfackel. Der Größe von diesem Array ist fest 3, weil es 3 Teile vom Signalfackel gibt und kein Teil kann wegnehmen.

- **Methoden**
- **+void putTogether(RoundController rc):** Mit dieser Methode können die Spieler das Spiel beenden. Wenn alle Spieler in derselbe Eisplatte stehen, dann es kostet eine Arbeit diese Methode zu rufen und das Spiel zu beenden.

3.3.14 PlayerContainer

- **Verantwortung**

Diese Klasse beinhaltet die Spieler. Die Spieler werden durch ihren Identifikationsnummer eindeutig identifiziert.

- **Superklasse**
-

- **Schnittstellen**
-

- **Attributen**

- **-Player[] pid:** Alle Player wird in einer Container (*PlayerContainer*) auch speichert.

- **Methoden**

- **+Player getPlayer(int pid):** Diese Methode gibt ein Spieler zurück. Wir müssen den Identifikationsnummer dem bestimmten Spieler eingeben. Die verschiedenen Nummern werden zu verschiedenen Spielern zugeordnet.

3.3.15 SnowStorm

- **Verantwortung**

Diese Klasse erzeugt den Schneesturm. Dieser Sturm verringert die Körpertemperatur des Spielers, fall er sich nicht in einem Iglu und erhöht sich die Dicke der Schneeschichten auf die verschiedenen Eisplatten.

- **Superklasse**
-

- **Schnittstellen**
-

- **Attributen**
-

- **Methoden**

- **+void tryStorm():** Diese Methode erzeugt eigentlich den Schneesturm.

3.3.16 **SignalFlarePart**

- **Verantwortung**

Diese Klasse speichert ein Teil von Signalfackel. Es gibt 3 Teile und das Ziel des Spiels ist, diese 3 einzubauen und abzuschießen. Die verschiedenen Teile sind verschiedene Instanz von dieser Klasse.

- **Superklasse**

Item->SignalFlarePart

- **Schnittstellen**

-

- **Attributen**

- **-int partID:** Die Identifikationsnummer von dem Teil. Es muss eindeutig sein.
- **+void used(Player player, Activity activity):** Die “*used*” Methode von der Superklasse (Item) wird überschrieben. (override)

- **Methoden**

-

3.3.16.1 **PositionLUT**

- **Verantwortung**

Diese Klasse handelt sich die Bewegungen von den Spielern. Den Spielern können mit Hilfe von dieser Klasse von einer Platte auf eine andere Platte springen. Diese Klasse ist ein so genannt “Singleton”, also in dem Spieler befindet sich nur eine von dieser Klasse. Des Weiteren können wir die Position von verschiedenen Spielern abfragen oder welchen Gegenständen befindet sich in einer bestimmten Eisplatte.

- **Superklasse**

-

- **Schnittstellen**

-

- **Attributen**

- **-Item[] itemTileMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays)
- **-Item[] tileItemMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays)
- **-Player[] playerTileMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays) Es gibt mindestens 3 Spieler (Es ist eine Anforderung, damit man das Spiel beginnen können), und maximum 7.
- **-Player[] tilePlayerMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays) Es gibt mindestens 3 Spieler (Es ist eine Anforderung, damit man das Spiel beginnen können), und maximum 7.
- **-Tile tileList[36]:** Die Eisplatten (alle) werden in dieser Klasse (*PositionLUT*) in diesem Array gespeichert. Es ist 36, weil es fest 36 Eisplatte gibt.

- **Methoden**

- **+Tile getPosition(Player p):** Mit dieser Methode können wir die Position des bestimmten Spieler abfragen. Wir müssen einen Spieler eingeben und es wird zurückgegeben, auf welche Eisplatte steht er.
- **+Tile getPosition(Item i):** Mit dieser Methode können wir die Position des bestimmten Gegenstand abfragen. Wir müssen einen Gegenstand eingeben und es wird zurückgegeben, auf welche Eisplatte befindet sich es.
- **+Player[] getPlayersOnTile(Tile t):** Diese Methode ergibt eine Spieler Array. In diesem Array befindet sich ein Spieler, falls er auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.
- **+Item[] getItemOnTile(Tile t):** Diese Methode ergibt eine Gegenstand Array. In diesem Array befindet sich ein Gegenstand, falls es auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.
- **+void setPosition(Player p):** Mit dieser Methode können wir die Position von dem eingegebenen Spieler einstellen.
- **+void setPosition(Item i):** Mit dieser Methode können wir die Position von dem eingegebenen Gegenstand einstellen. Es kann vorkommen, wenn zum Beispiel ein Spieler ein Essen isst. Dann wird das Essen auf eine zufällige Eisplatte generiert.

3.3.17 Tile

- **Verantwortung**

Diese Klasse ist ein Container, also zusammenfasst alle Art von den Eisplatten. Die Klasse ist abstrakt, man kann es nicht instanzieren. Es gibt 3 Arten von den Eisplatten. Es kann stabil, instabil oder ein schneebedecktes Loch sein. Diese Arten sind verschiedene Subklassen, vererben diese abstrakten Klasse.

- **Superklasse**

-

- **Schnittstellen**

-

- **Attributen**

- **#int x:** Es ist die x Koordinate (horizontale) von der bestimmten Eisplatte.
- **#int y:** Es ist die y Koordinate (vertikale) von der bestimmten Eisplatte.
- **#int snow:** Diese Attribute speichert, wie viel Schnee (wie viel Einheit) sich auf die Eisplatte befindet.
- **-boolean igluOn:** Diese Attribute bestimmt, ob ein Iglu sich auf die Eisplatte befindet. (true: Ja, false: Nein)

- **Methoden**

- **+void steppedOn(Player p):** Ein Spieler tritt auf die Eisplatte. Wir müssen den Spieler eingeben.
- **+void steppedOff(Direction dir):** Ein Spieler tritt auf eine andere Eisplatte. Wir müssen die Richtung eingeben. Direction ist eine Enumeration, also von bestimmten Richtungen (4 Himmelsrichtung) können wir wählen.
- **+int changeSnow(signed int thisMuch):** Die Höhe des Schneeschicht verändert sich. Es wird eingegeben, mit wie vielen. Es kann höher oder kleiner sein. Zum Beispiel höher, falls ein Schneesturm kommt.
- **+Tile getNeighbour(Direction dir):** Mit dieser Methode können wir die benachbarten Eisplatten von einer Platte abfragen. Wir müssen die Richtung auswählen und eingeben.

3.3.18 SnowyHole

- **Verantwortung**

Diese Klasse spezialisiert die “Tile“ Klasse. Diese Klasse verwirklicht das schneebedeckte Loch. Diese Eisplatten siehe so aus, wie die andere schneebedecktes Platte, aber wenn ein Spieler auf diese tritt, fällt ins Wasser. Auf diesen Platten können 0 Spieler stehen.

- **Superklasse**

Tile-> SnowHole

- **Schnittstellen**

-

- **Attributen**

-

- **Methoden**

-

3.3.19 StableTile

- **Verantwortung**

Diese Klasse spezialisiert die “Tile“ Klasse. Diese Klasse verwirklicht die stabile Eisplatte. Auf diesen Platten können unendlich viel Spieler stehen. Auf diesen Platten kann sich natürlich auch Schnee befinden.

- **Superklasse**

Tile-> SnowHole

- **Schnittstellen**

-

- **Attributen**

-

- **Methoden**

-

3.3.20 UnstableTile

- **Verantwortung**

Diese Klasse spezialisiert die “Tile“ Klasse. Diese Klasse verwirklicht die instabile Eisplatte. Auf diesen Platten können nur begrenzt viel Spieler stehen. Die Anzahl den Spielern ist nicht sichtbar, aber die Forscher können es anschauen.

- **Superklasse**

Tile-> SnowHole

- **Schnittstellen**

-

- **Attributen**
- **-int capacity:** So viele Spieler können auf diese Platte stehen. Dieser Wert ist größer gleich null und wird zufällig eingestellt.
- **+int standingHere:** Die Anzahl der Spieler, die auf diese Platte stehen. Es kann nicht größer oder gleich als "capacity" sein.

- **Methoden**
-

3.3.21 Direction

- **Verantwortung**

Diese Klasse ist eine Enumeration und beinhaltet die verschiedenen Richtungen. Wenn ein Spieler treten möchte, sollte er von diesen Richtungen wählen. Es ist eigentlich die 4 Himmelrichtungen. Andere Richtung existiert in diesem Spiel nicht, also können die Spieler diagonale weise nicht treten.

- **Superklasse**
-

- **Schnittstellen**
-

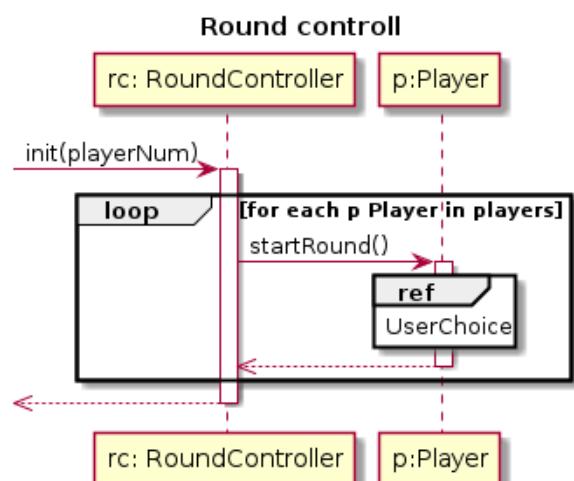
- **Aufzählungsliterale**

- **up:** Auf die nördliche Platte treten. (Nord)
- **down:** Auf die untere Platte treten. (Süd)
- **left:** Auf die linke Platte treten. (West)
- **right:** Auf die rechte Platte treten. (Ost)

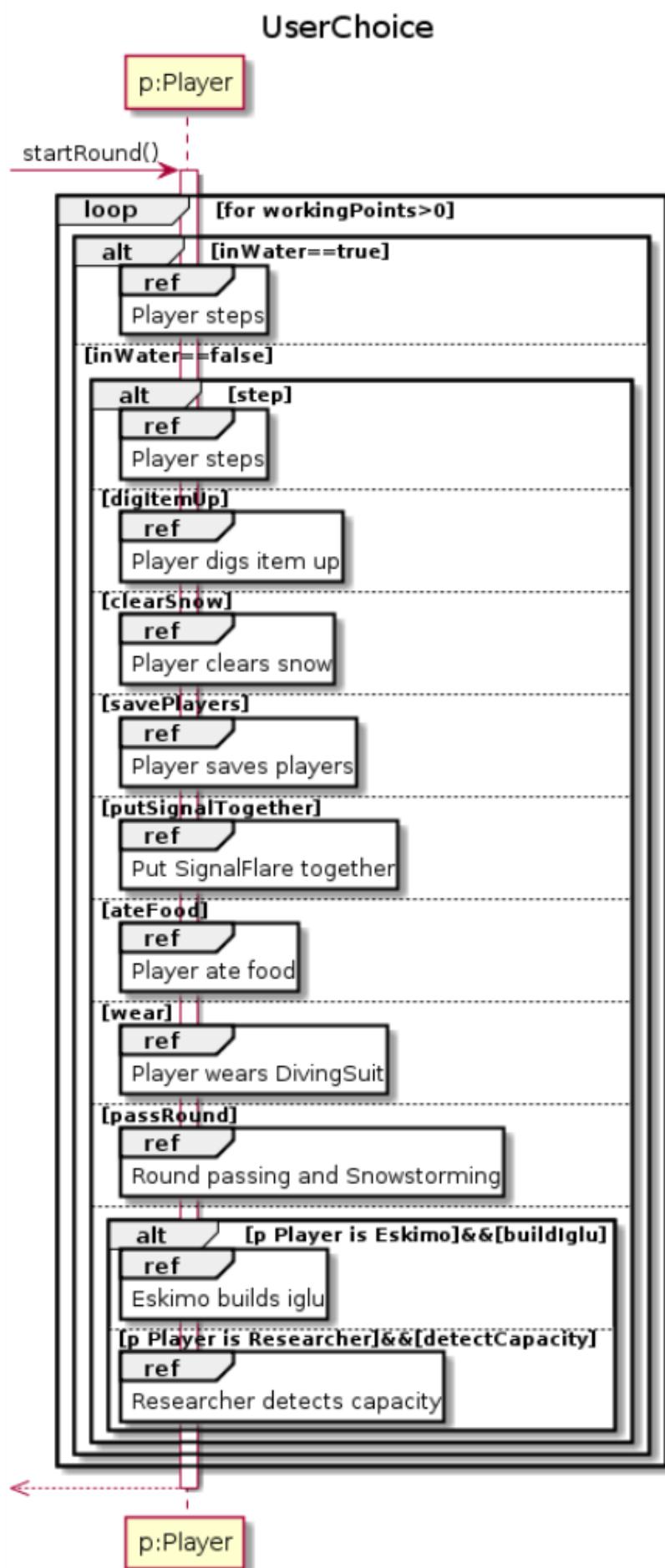
- **Methoden**
-

3.4 Sequenz Diagramme

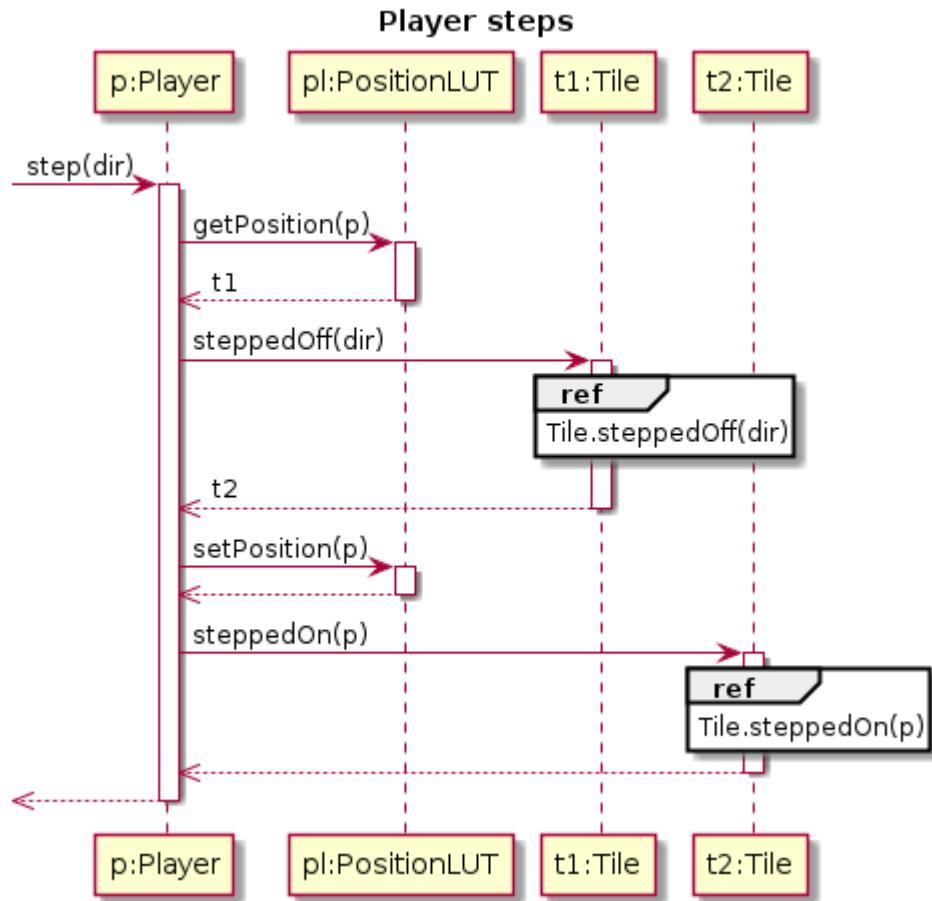
3.4.1 Round control



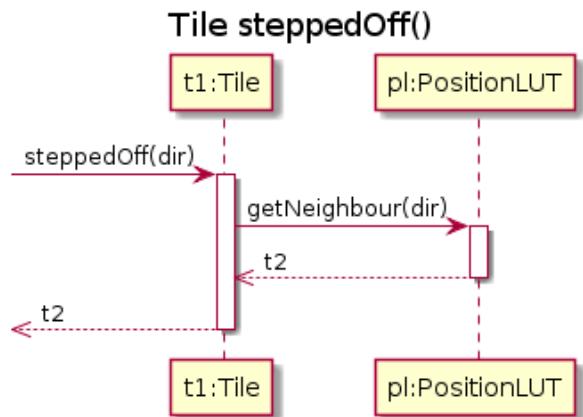
3.4.2 UserChoice



3.4.3 Player steps

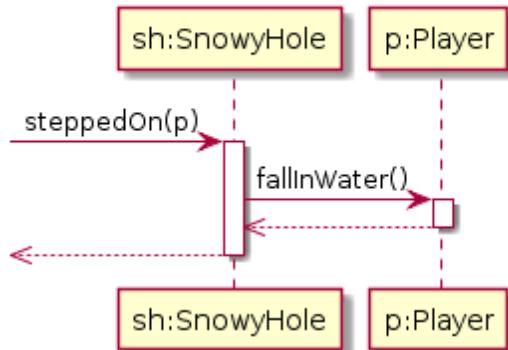


3.4.4 Tile steppedOff



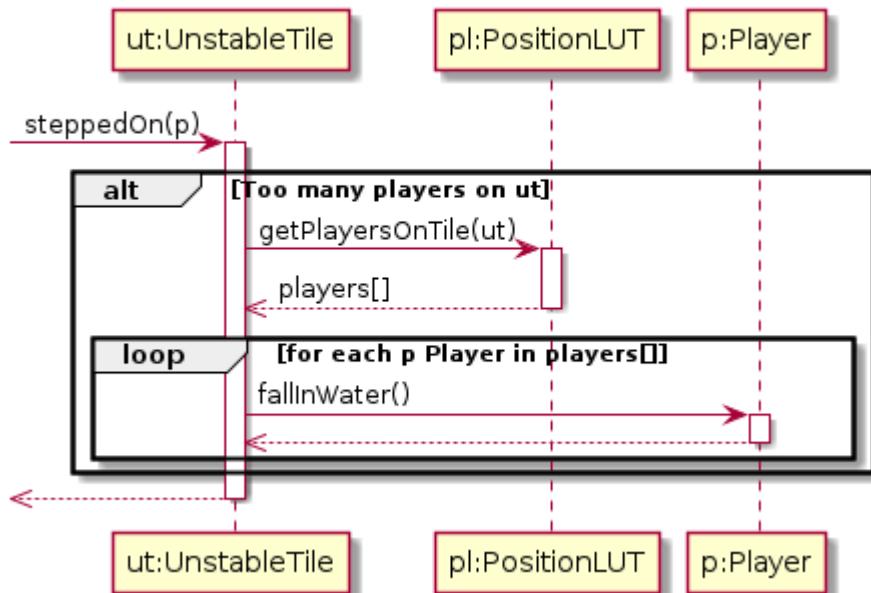
3.4.5 SnowyHole steppedOn

SnowyHole steppedOn()



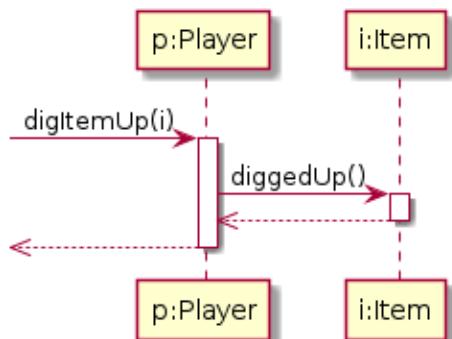
3.4.6 UnstableTile steppedOn

UnstableTile steppedOn()



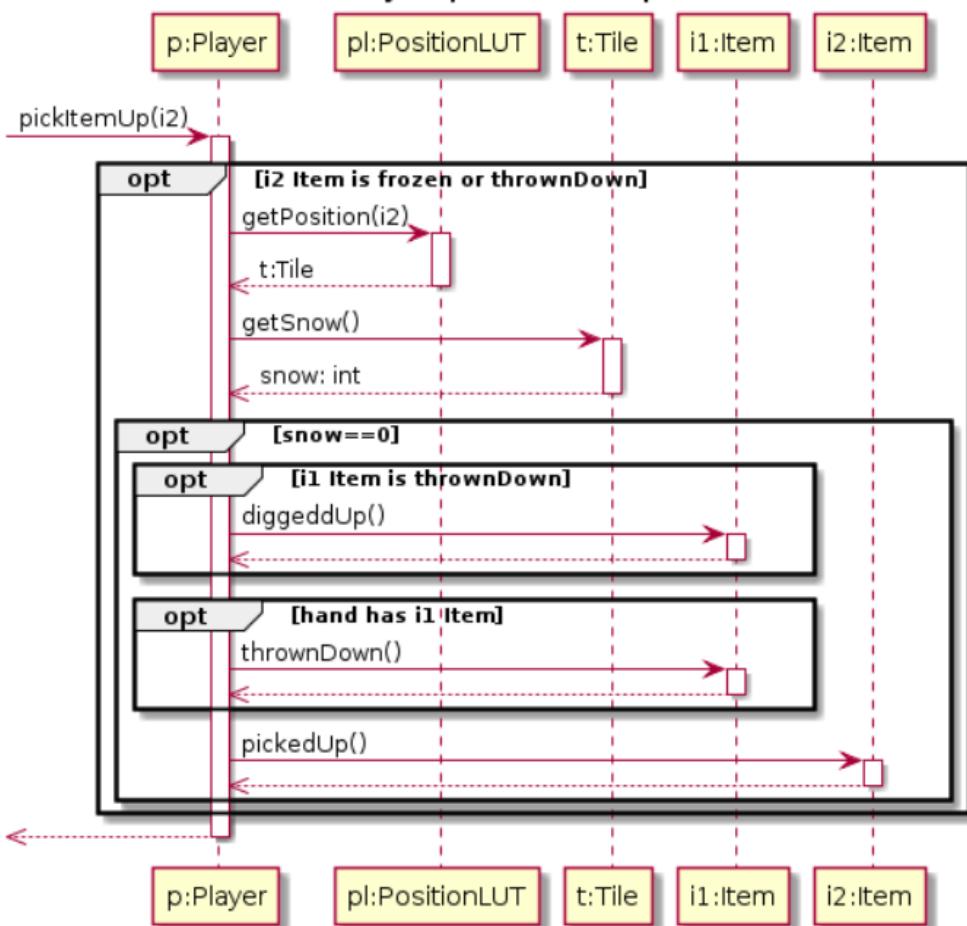
3.4.7 Player digs item up

Player digs item up

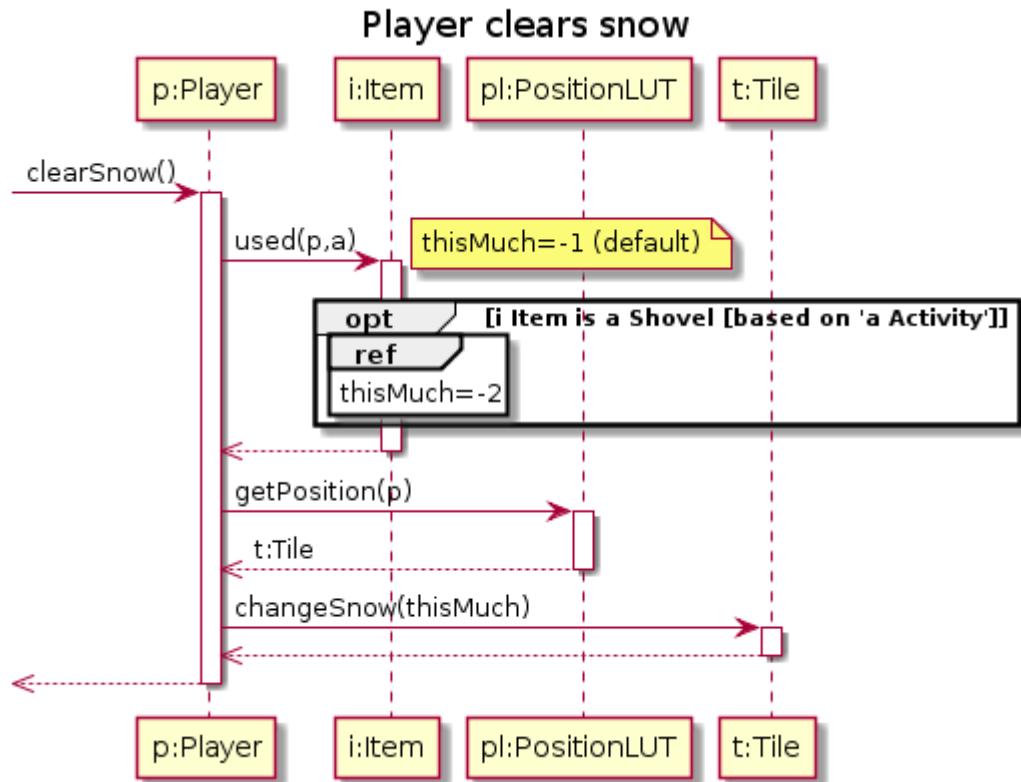


3.4.8 Player picks item up

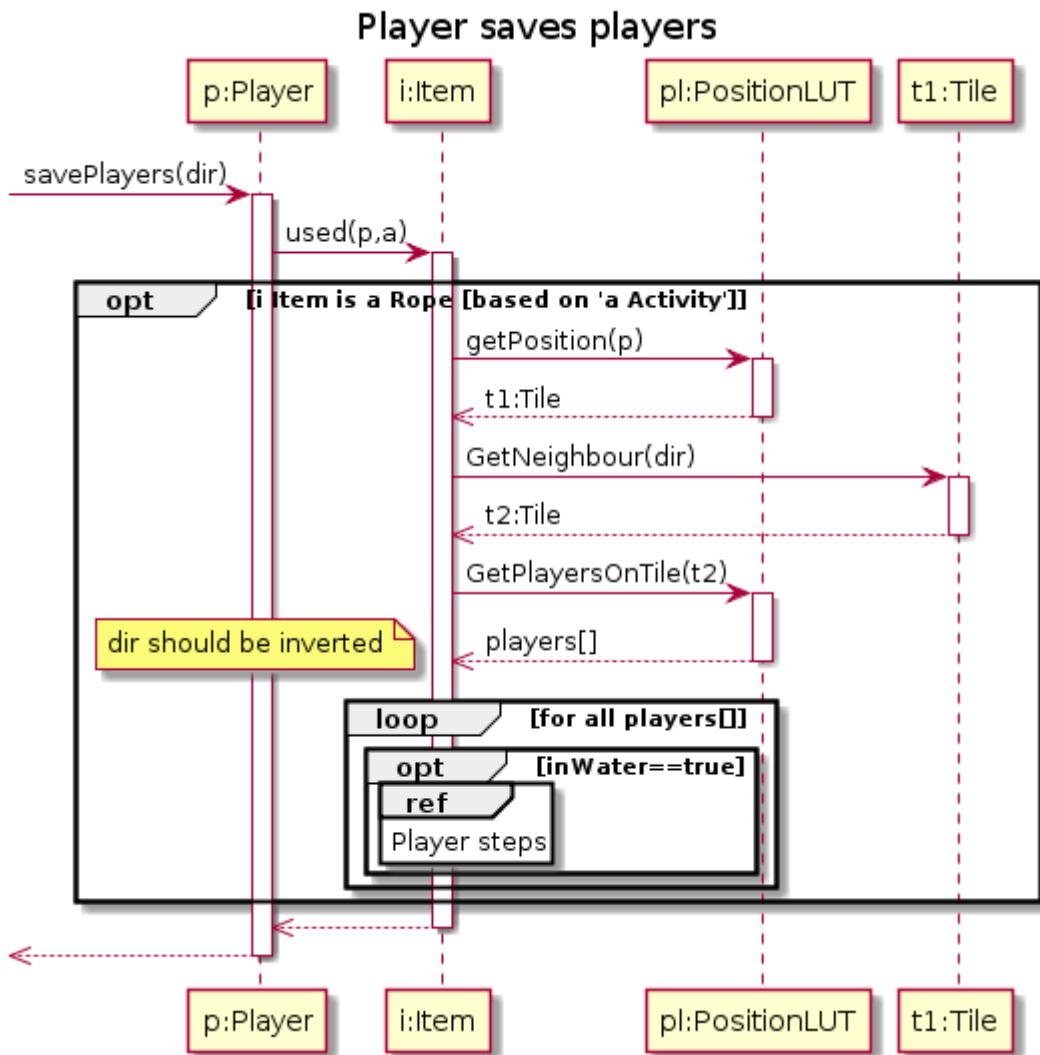
Player picks item up



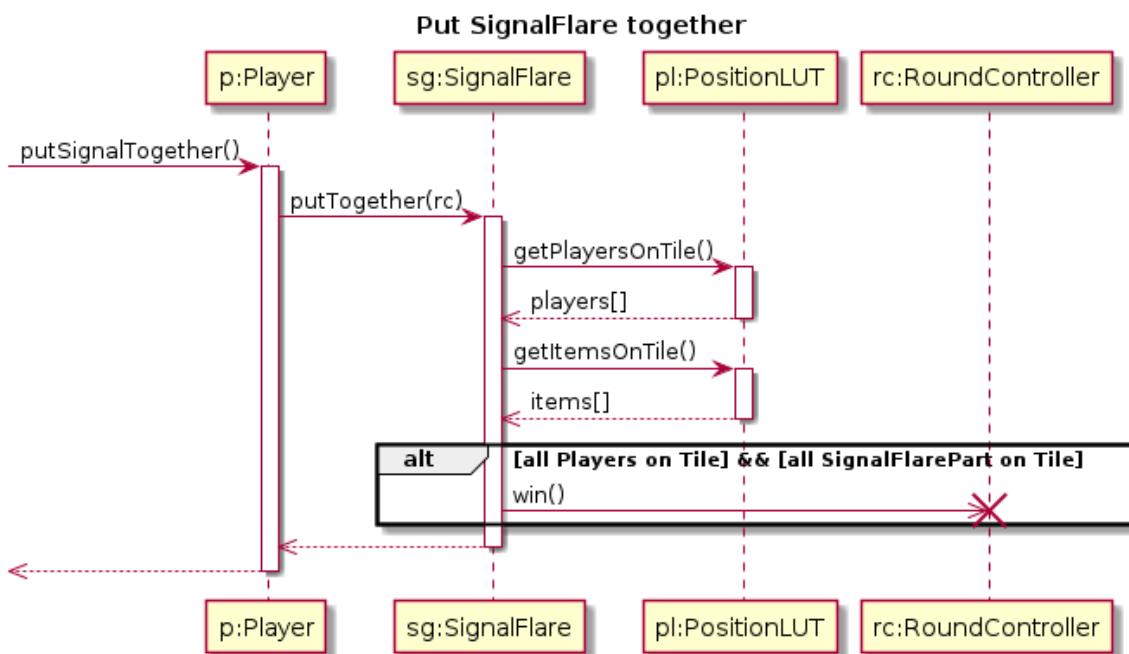
3.4.9 Player clears snow



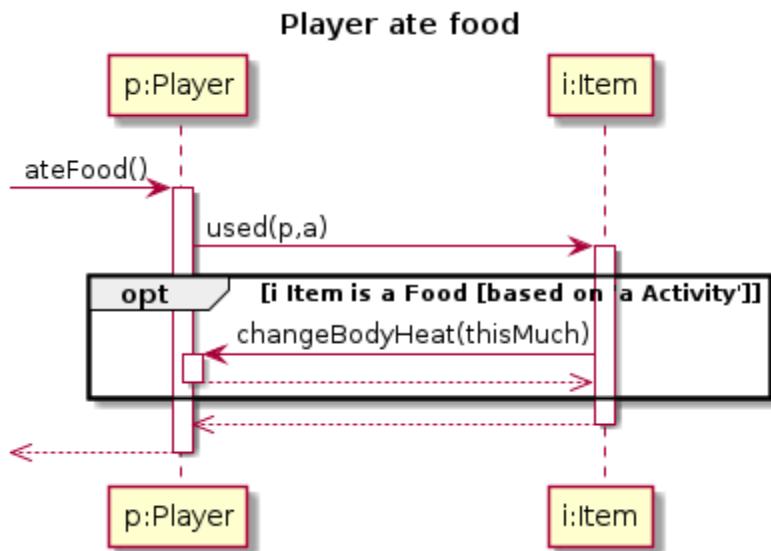
3.4.10 Player saves players



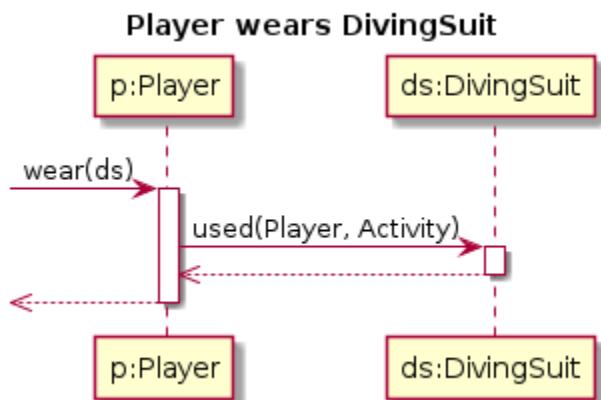
3.4.11 Put SignalFlare together



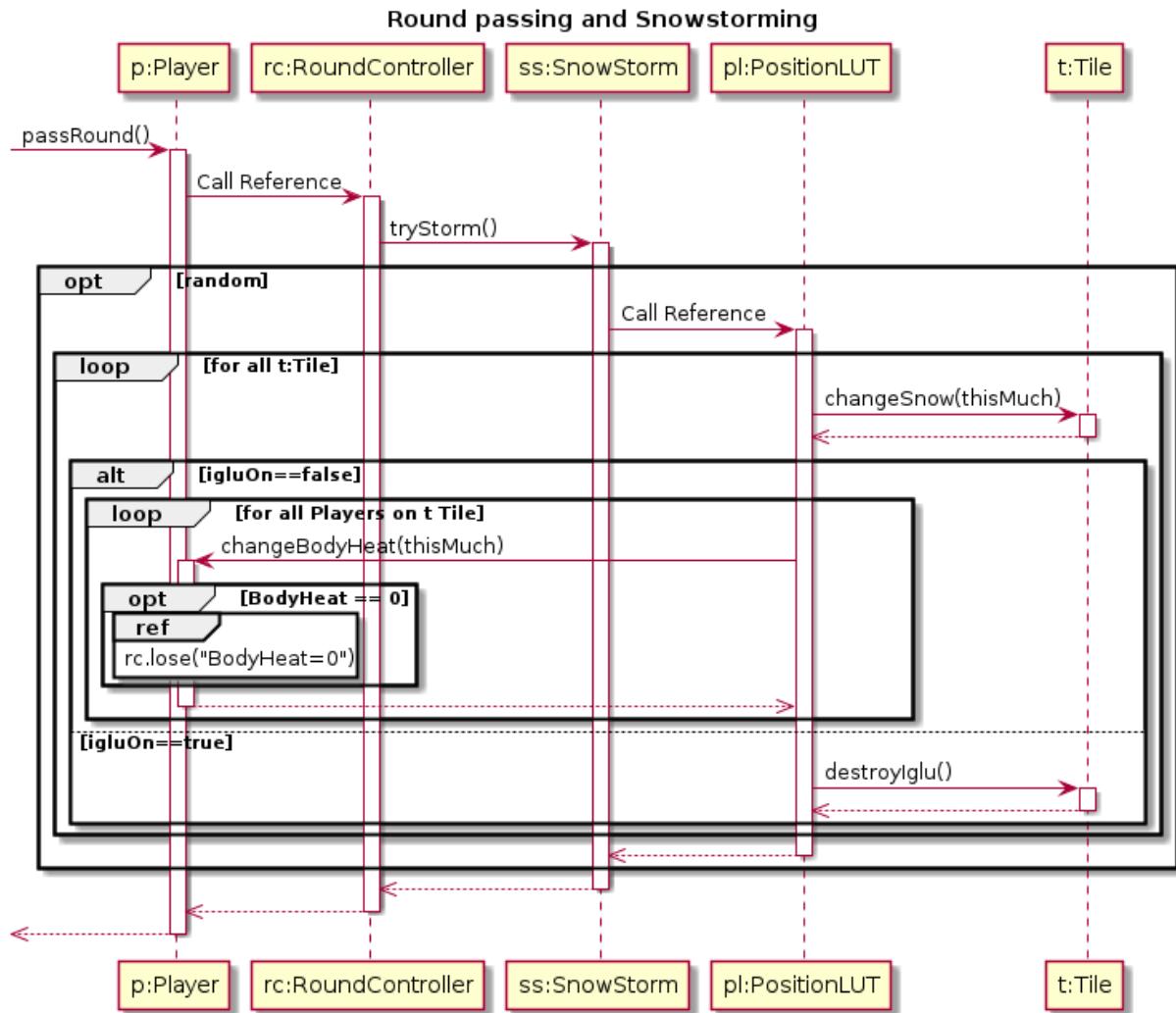
3.4.12 Player ate food



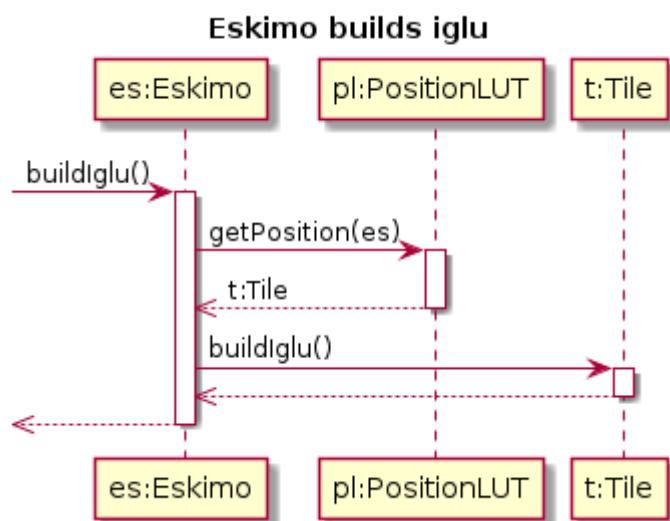
3.4.13 Player wears DivingSuit

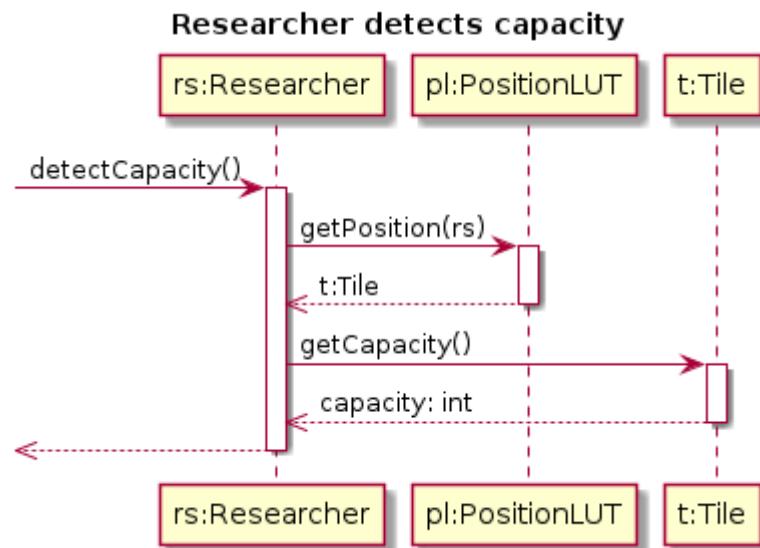


3.4.14 Round passing and Snowstorming



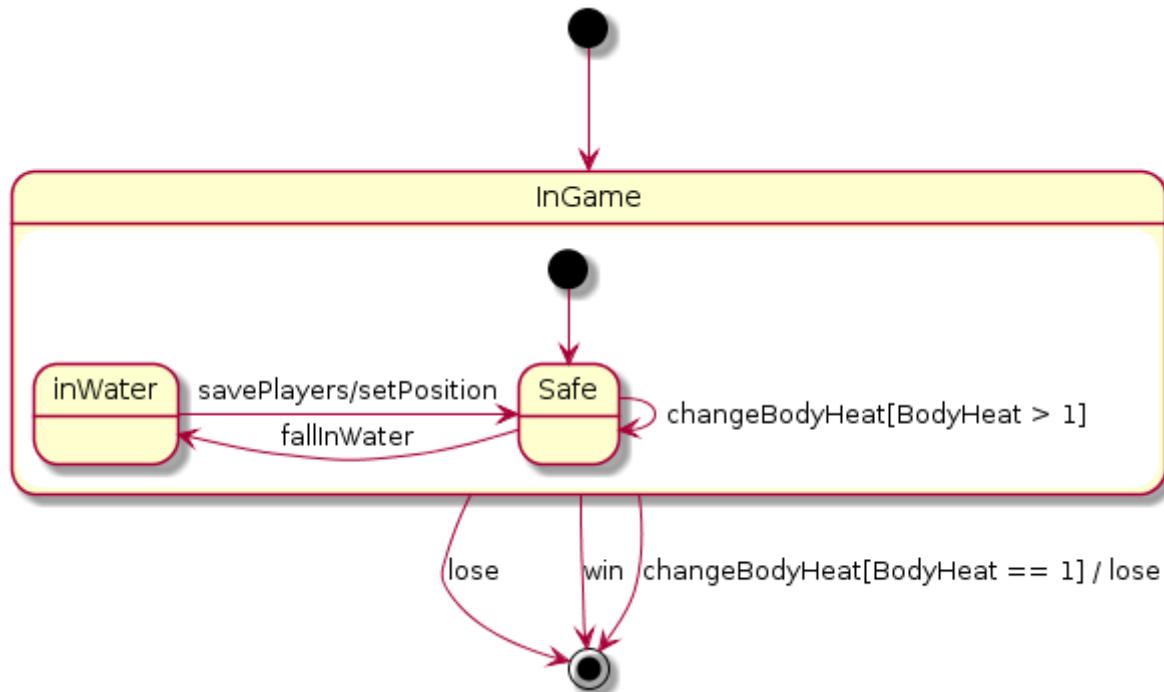
3.4.15 Eskimo builds iglu

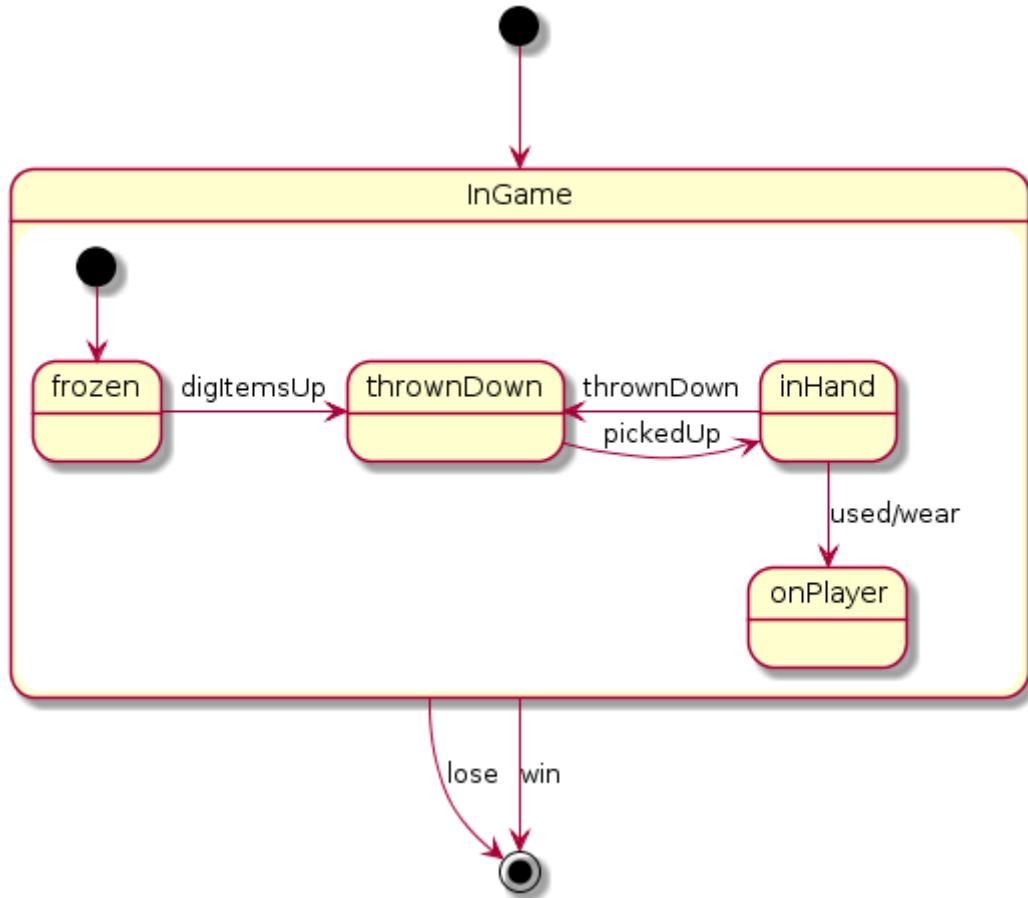




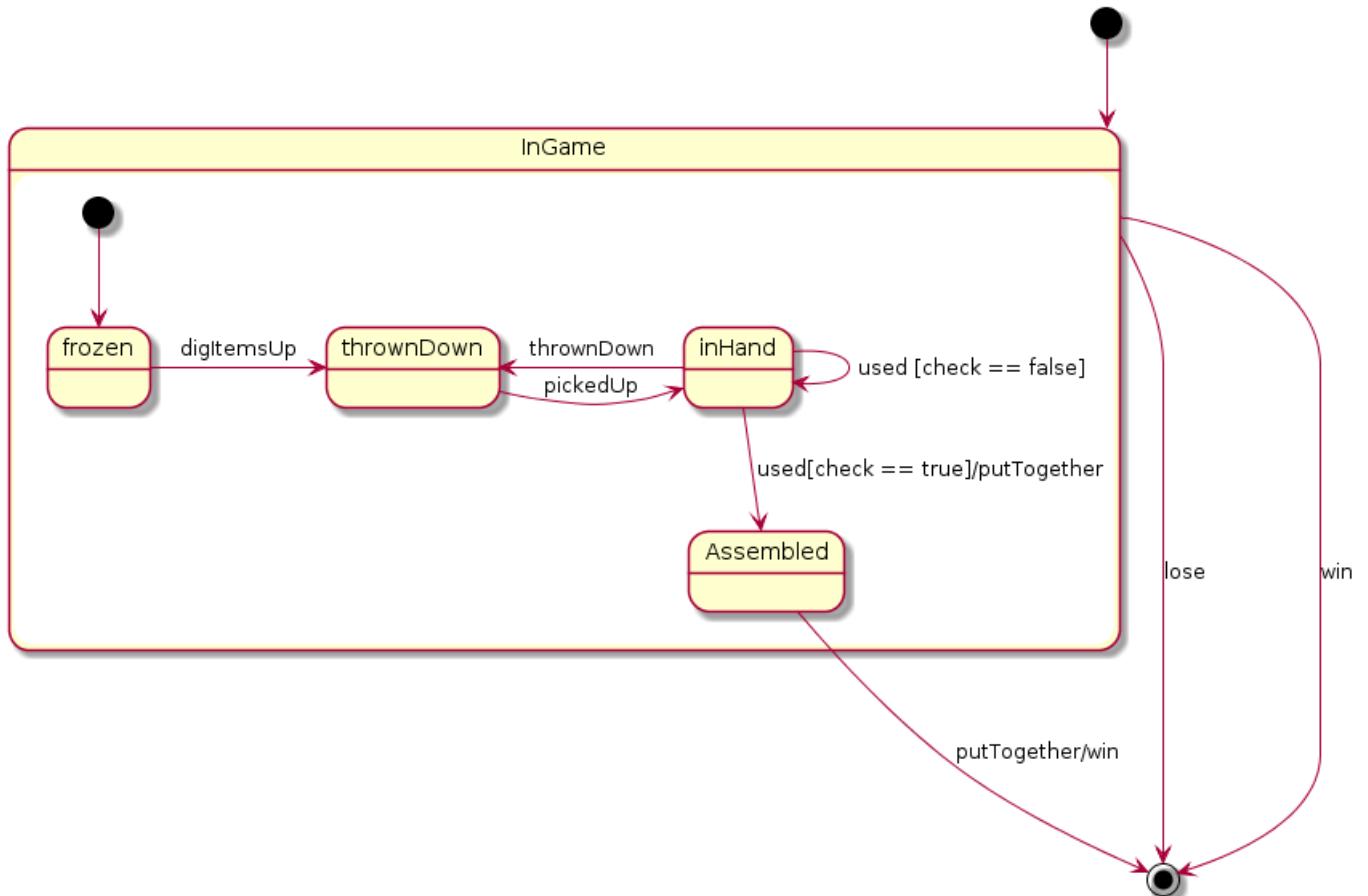
3.5 Zustandsdiagramme (State-charts)

PlayerStates:

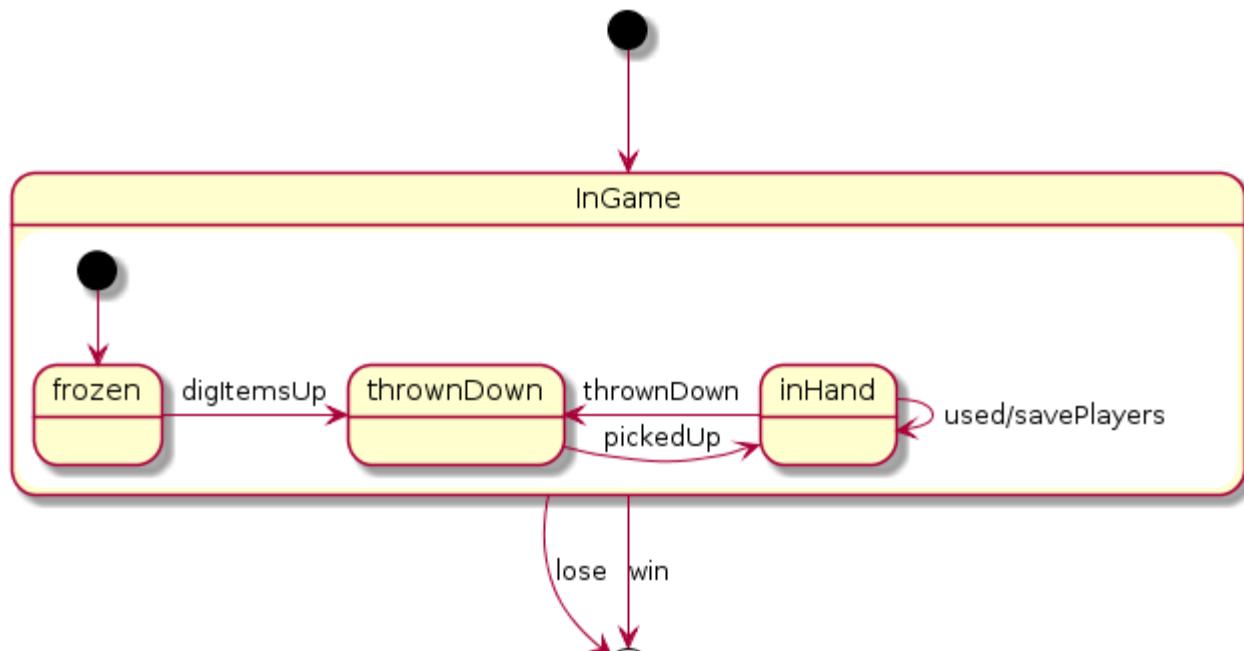




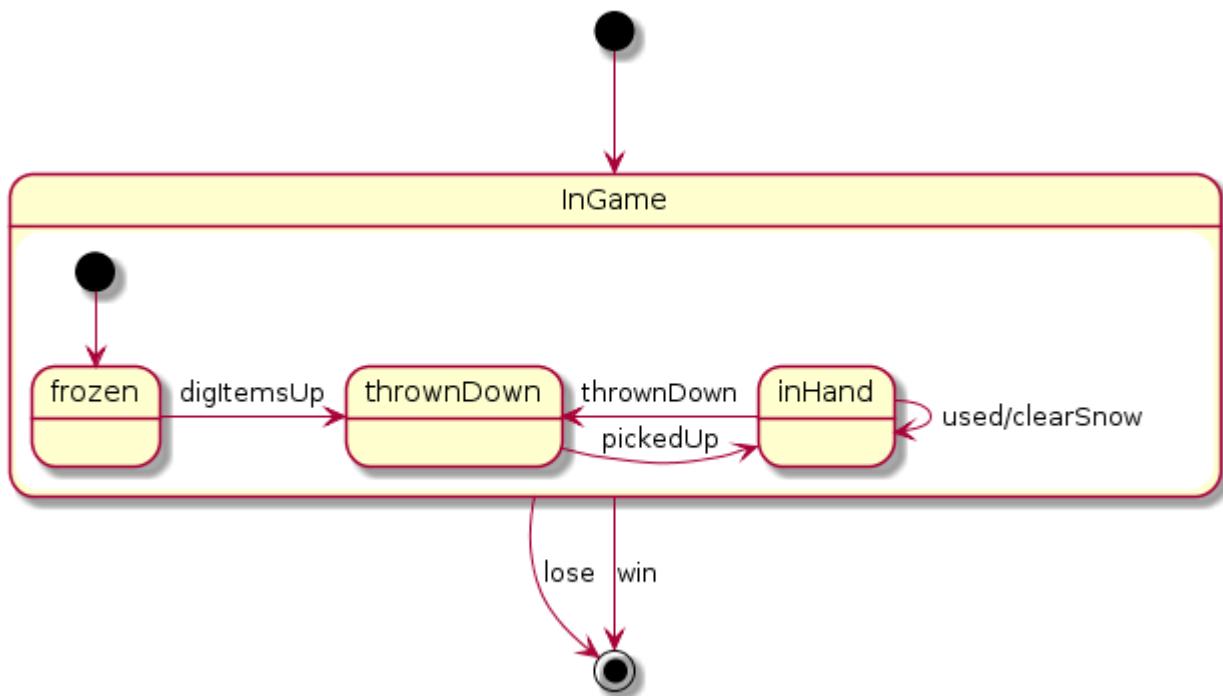
ItemStates(SignalFlarePart)



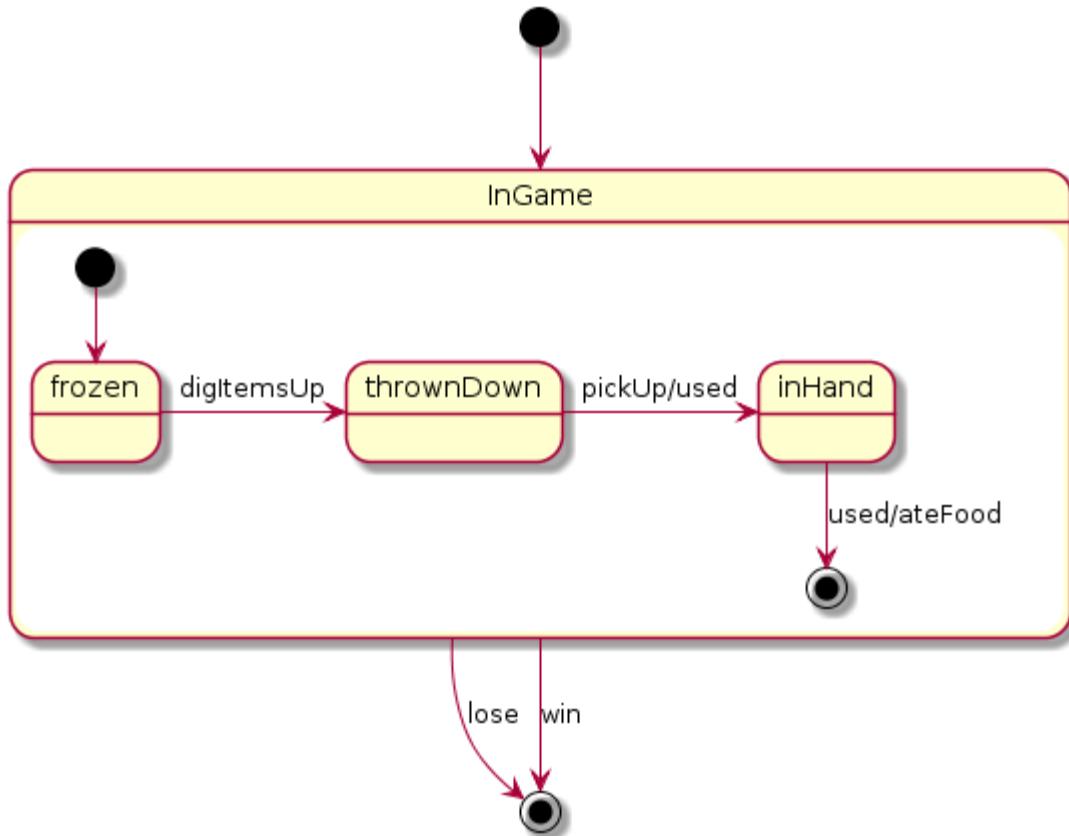
ItemStates(Rope):



ItemStates(Shovel)



ItemStates(Food)



4. Ausarbeitung von Analyse-Modell DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia
Hedrich Ádám
Pintér Balázs
Fucskár Patrícia
Tassi Timián

SOSK6A
H9HFFV
ZGY18G
XKYAOO
MYY53U

adamzsofi.mail@gmail.com
hedrichadam09@gmail.com
pinterbalazs21@gmail.com
fucskar.patricia@gmail.com
timian.tassi@gmail.com

4. März 2020

4. Entwicklung des Analysemodells

4.1 Objektkatalog

4.1.1 Players (Eskimos und Researchers)

Diese Objekte sind für die Zustände und Zustandsänderungen des Spielers verantwortlich. Sie speichern, ob die Spieler etwas im Hand haben, durch diese können die Benutzer Tätigkeiten initiieren und sie senden Signale über Sieg/Fehlschlag.

4.1.2 Platten

Sie sind für die Bewegung des Spielers verantwortlich (Sp. nehmen und geben). Sie speichern Positionen, sie können ihre Nachbarn erreichen und sie können entwerten, ob es zu viele Spieler auf sich stehen oder nicht (Kapazitäten). Es gibt 3 Typen (Loch, stabil instabil), die sich anders verhalten, wie es in Aufgabe gegeben war.

4.1.3 Gegenstände (mehrere vererbte “Typen”)

Es gibt mehrere Typen dieser Objekte. Von jedem Typen gibt es immer ein (*Wenn jemand ein Essen gegessen hat, dann wird ein neues gefrorenes Essen generiert*). Sie haben drei Zustände: gefrieren, geworfen und in der Hand, es ist in einem Enum. Diese Gegenstände erlauben verschiedenen Tätigkeiten (*falls jemand ein Essen in der Hand hat, kann er dann Essen; mit Schaufel effizienter graben; usw.*)

Gegenstandstypen:

- Essen (Food)
- Seil (Rope)
- Schaufel (Shovel)
- Taucheranzug (DivingSuit)
- SignalFackel Elemente (Signal Flare parts)

4.1.4 Schneesturm

Der Schneesturm ist verantwortlich für die Schneestürme (sehr überraschend).

Es hat immer eine Chance nach jedem Rund für ein Schneesturm, dieser Zufall wird bei dem Schneesturm “gerechnet”. Falls es kommt, dann es lost die “stürmige” Platten aus und kommuniziert mit dem Spieler und die Platten über den neuen Schnee und die Spieler Verletzungen. Es zerstört die Iglus.

4.1.5 RoundController

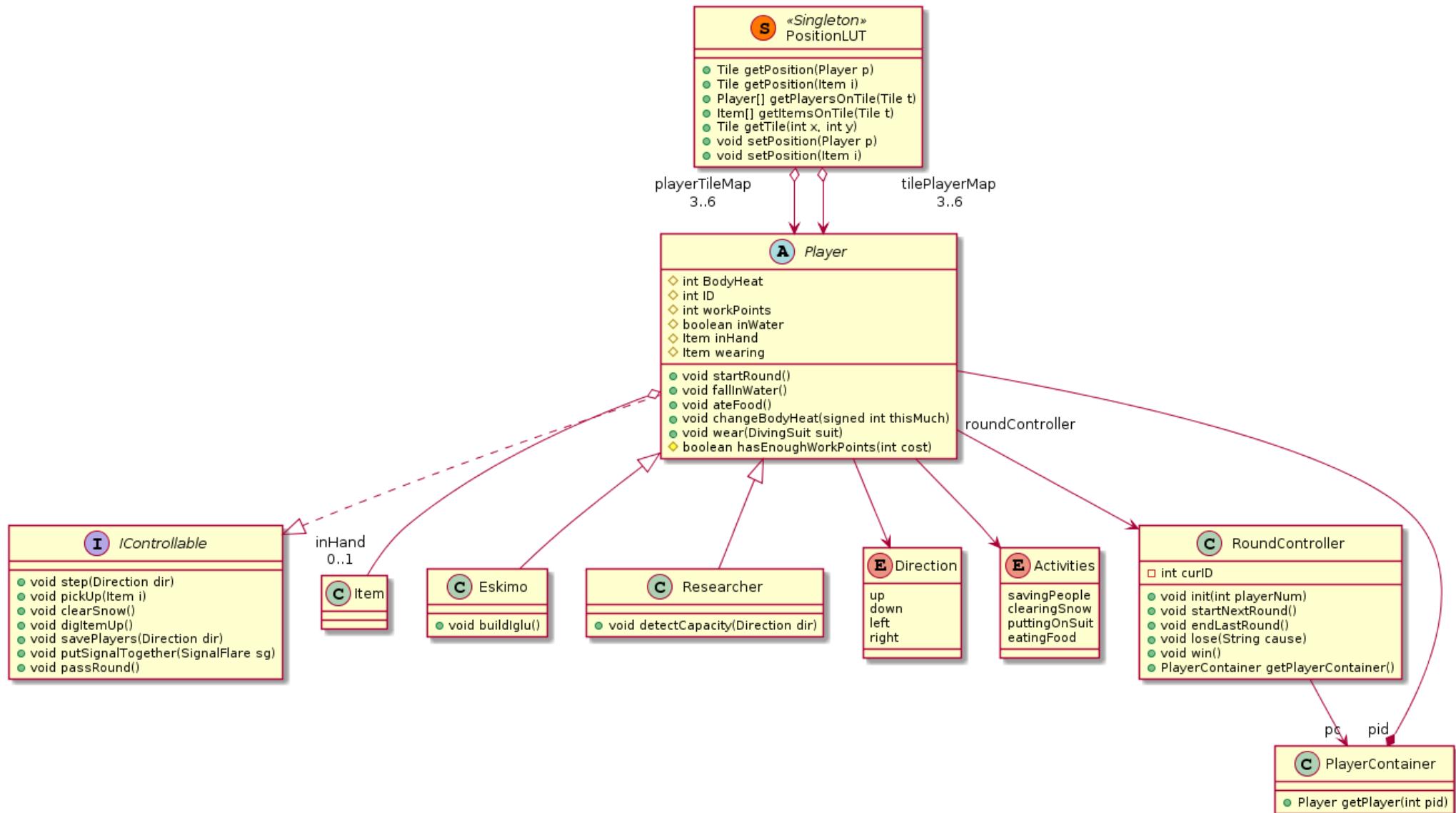
Dieses einzige Objekt wird für Initiierung und Laufen des Spiels verantwortlich. Es wird nach der Anzahl der Spieler fragen und dann die anderen Objekte initialisieren. Es macht die gebrauchte „checks“ und „cleanup“/Initiierung zwischen die Runde der Spieler und behandelt Sieg/Fehlschlag.

4.1.6 SignalFlare

Dieses Objekt speichert die 3 Signalflacke Teile (Komposition), und kann zusammengebaut werden, falls die Umstände genügend sind.

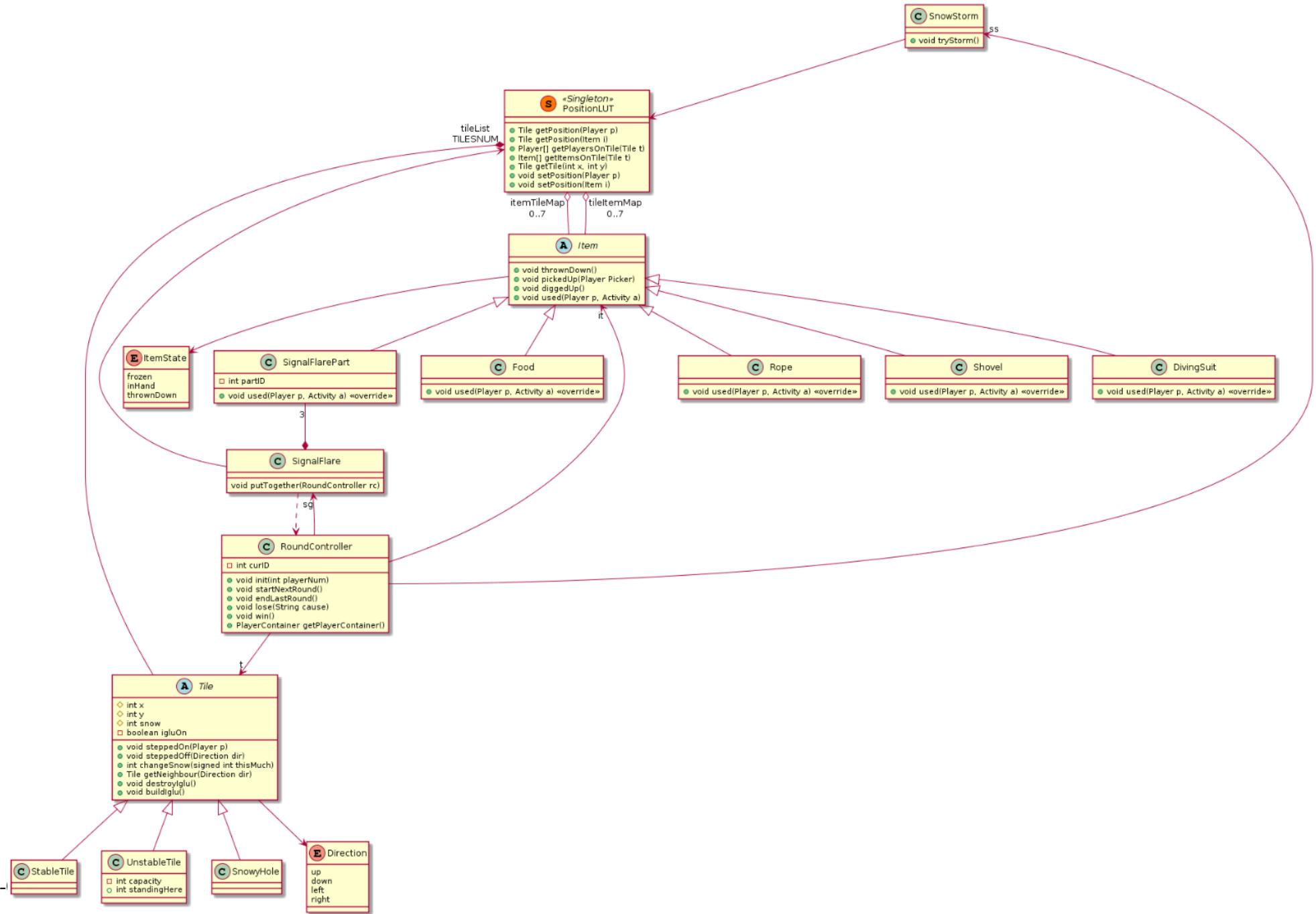
4.2 Klassendiagramme

Spieler - Class Diagram Teil 1



4. Ausarbeitung von Analyse-Modell

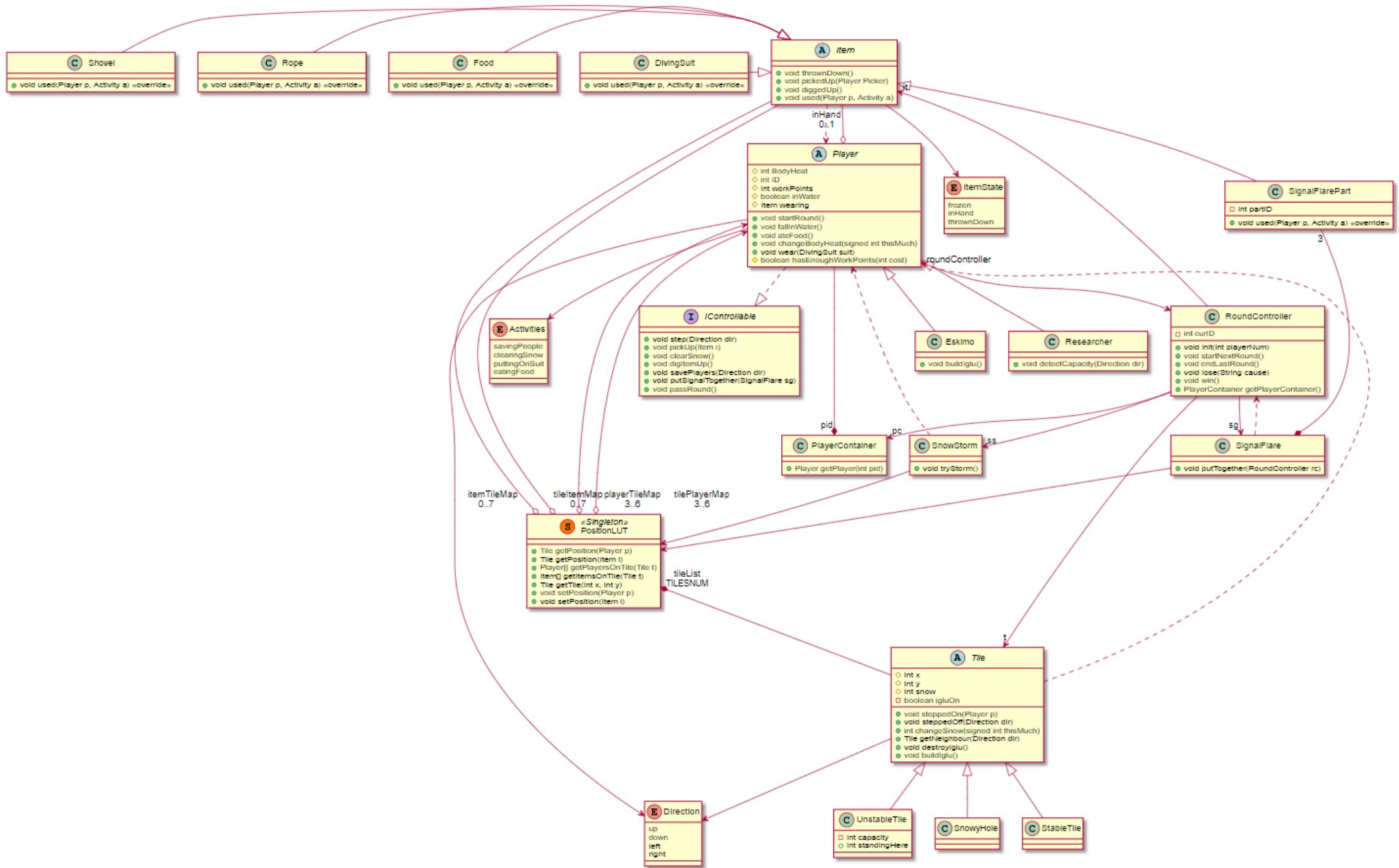
DeutschOverflow
Felder, Gegenstände, Schneesturm - Class Diagram Teil2



4. Ausarbeitung von Analyse-Modell

DeutschOverflow

Eisfeld - Class Diagram



4.3 Beschreibung der Klassen

4.3.1 Player

- Verantwortung

Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanzieren, weil ein Spieler entweder Eskimo oder Forscher ist. Diese Klasse enthält die grundsätzlichen Eigenschaften (Attributen) und die gemeinsamen Methoden von beiden Art von Spielern. Natürlich die Eskimos und Forscher Klasse spezialisiert diese Klasse, also enthält weitere Methoden.

- Schnittstellen

- IControllable

- Attributen

- #int **Bodyheat**: Die Körpertemperatur des Spielers. Es ist 5 Einheit für Eskimos und 4 Einheit für Forscher.
- #int **ID**: für eindeutige Charakterisierung des Spielers
- #int **workPoints**: Der Spieler hat so viele Arbeit (Einheit) noch.
- #Item **inHand**: Der Gegenstand, der in der Hand des Spielers sich befindet. Es kann 0 oder 1 Gegenstand in der Hand sein. (mehr als 1 nicht)
- #DivingSuit **wearing**: Es ist eigentlich der Taucheranzug, wenn der Spieler es trägt.
- #boolean **inWater**: Es zeigt, ob der Spieler im Wasser ist. (true: Ja, false: Nein)
- #RoundController **roundController**: Es ist zu dem Spieler gehörende RoundController.

- Methoden

- +void **startRound()**: Mit dieser Methode startet ein Spieler der Round.
- +void **fallInWater()**: Der Spieler fällt ins Wasser. Er wartet auf die Rettung.
- +void **changeBodyHeat(signed int thisMuch)**: Der Spieler kann sich seine Körpertemperatur erhöhen (mit dem Essen) oder es wird durch eine Schneesturm verringert.
- +void **AteFood()**: Der Spieler isst sein Essen, so seine Körpertemperatur wird sich erhöht.
- +void **wear(DivingSuit suit)**: Der Spieler trägt den Taucheranzug. Wenn er ins Wasser fällt, wird nicht gestorben.
- #boolean **hasEnoughWorkPoints(int costs)**: Diese Methode gibt zurück, ob der Spieler die bestimmten Tätigkeit durchführen kann. Es hängt von dem gebliebenen Arbeit Einheit ab. (true: ja, false: nein)

4.3.2 IControllable

- Verantwortung

Diese Klasse ist eine Schnittstelle. Einige Methoden befinden sich hier. Diese Methoden werden durch Klasse "Player" benutzt: Treten, Gegenstand aufnehmen, schneeräumen, Gegenstand ausgraben, Teile vom Signalfackel zusammenbauen, Runde passen.

- Methoden

- +void **passRound()**: Mit dieser Methode kann der Spieler passen, also das Recht für Tritt weitergeben.
- +void **digItemUp()**: Der Spieler räumt mit Hilfe von einem Gräber Schnee, wenn er ein Gräber hat.
- +void **pickUp(Item i)**: Der Spieler kann ein Gegenstand aufnehmen, wenn der Gegenstand in die Eisplatte nicht einfriert.
- +void **clearSnow()**: Der Spieler macht eine Eisplatte sauber.

4. Ausarbeitung von Analyse-Modell

DeutschOverflow

- **+void step(Direction dir):** Der Spieler kann in bestimmten Richtungen treten. (auf eine andere Eisplatte) Direction ist eine Enumeration, die die 4 Richtung beinhaltet.
- **+void putSignalTogether(Signalflare sf):** Am Ende des Spiels feuern ein Spieler die Signalfackel. Es ist möglich, wenn alle Spieler in einer Eisplatte steht und die Teile von der Signalfackel da sind. Es kostet 1 Arbeit (Einheit).
- **+void savePlayers(Direction dir):** Der Spieler rettet sein Kumpel mit Hilfe vom Seil. Man muss eingeben, auf welche Eisplatte (Direction) wird der Spieler gerettet.

4.3.3 Eskimo

- **Verantwortung**

Diese Klasse spezialisiert die “Spieler“ Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Superklasse, weil allen Attributen beiden Subklassen vorkommt, nur die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **Superklasse**

Player->Eskimo

- **Methoden**

- **+void buildIglu():** Die Eskimos können ein Iglu bauen. Es schützt von dem Schneesturm.

4.3.4 Researcher

- **Verantwortung**

Diese Klasse spezialisiert die “Spieler“ Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann, als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Vorklasse, weil allen Attributen beiden Subklassen vorkommt, nur die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **Superklasse**

Player->Researcher

- **Methoden**

- **+void detectCapacity():** Die Forscher können mit dieser Methode anschauen, ob eine Eisplatte stabil oder instabil ist, also wie viel Spieler kann auf eine bestimmten Eisplatte stehen.

4.3.5 Item

- **Verantwortung**

Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanzieren. Es ist ein Container von verschiedenen Gegenständen. Die gemeinsame Eigenschaft von verschiedenen Gegenständen ist, dass es in dem Spieler immer 1 von jedem gibt.

- **Methoden**

- **+void thrownDown()**: Der Gegenstand wird abgeworfen. In diesem Fall ist der Gegenstand in die Eisplatte nicht eingefroren und bleibt ebendorf.
- **+void pickedUp(Player Picker)**: Der Gegenstand wird durch ein bestimmten Spieler aufgenommen. Der Spieler wird im Parameter gegeben.
- **+void diggedUp()**: Mit dieser Methode können wir den Gegenstand ausgraben.
- **+void used(Player player, Activity activity)**: Die verschiedenen Gegenstände können durch einen bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten.

4.3.6 Shovel

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Schaufel Gegenstand.

- **Superklasse**

Item->Shovel

- **Methoden**

- **+void used(Player player, Activity activity)**: Die “used” Methode von der Superklasse (Item) wird überschrieben. (override)

4.3.7 Rope

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Seil Gegenstand.

- **Superklasse**

Item->Rope

- **Methoden**

- **+void used(Player player, Activity activity)**: Die “used” Methode von der Superklasse (Item) wird überschrieben. (override)

4.3.8 DivingSuit

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Taucheranzug Gegenstand.

- **Superklasse**

Item->DivingSuit

- **Methoden**

- **+void used(Player player, Activity activity):** Die “used” Methode von der Superklasse (Item) wird überschrieben. (override)

4.3.9 Food

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Essen Gegenstand. Mit diesem Gegenstand können die Spieler sich ihre Körpertemperatur erhöhen.

- **Superklasse**

Item->Food

- **Methoden**

- **+void used(Player player, Activity activity):** Die “used” Methode von der Superklasse (Item) wird überschrieben. (override)

4.3.10 ItemState

- **Verantwortung**

Diese Klasse ist eine Enumeration. In dieser Enumeration befinden sich die Zustände von einem Gegenstand. Anderer Zustand existiert nicht. In einem Augenblick kann sich zu einem Gegenstand pünktlich ein Zustand gehören.

- **Aufzählungsliterale**

- **frozen:** Wenn ein Gegenstand gefroren ist, gehört sich zu diesem Zustand.
- **inHand:** Wenn ein Gegenstand sich in der Hand von einem Spieler befindet, gehört sich zu diesem Zustand.
- **thrownDown:** Ein Gegenstand kann auf eine Eisplatte nicht in gefroren Zustand sein, also durch einen Spieler abgeworfen wird. In diesem Fall gehört sich zu diesem Zustand

4.3.11 Activities

- **Verantwortung**

Diese Klasse ist eine Enumeration. Es ist gegeben, welche Aktivitäten ein Spieler durchführen kann. Diese Enumeration speichert diese Möglichkeiten. Andere Aktivitäten können während des Spiels nicht durchgeführt werden.

- **Aufzählungsliterale**

- **savingPeople:** Ein Spieler kann einen anderen Spieler von dem Wasser retten.
- **clearingSnow:** Die Spieler können die Eisplatten saubermachen, also der Schnee wird von der Eisplatte weggeputzt.
- **eatingFood:** Wenn ein Spieler sich mit einem Essen begegnet, isst automatisch es. Danach wird ein anderes Essen irgendwo erschien. Während dieser Tätigkeit wird sich die Körpertemperatur von dem bestimmten Spieler erhöht.
- **PuttingOnSuit:** Der Spieler nimmt den Taucheranzug auf sich. Es wird automatisch durchgeführt, wenn ein Spieler einen Taucheranzug aufnehmen

4.3.12 RoundController

- **Verantwortung**

Diese Klasse steuert das Ende und der Beginn eines Spiels, also es ist ein Kontroller. Es beinhaltet ein Spieler Container. In diesem Container befinden sich den Spielern. Die Runden werden auch in dieser Klasse kontrolliert. Am Anfang des Spiels kann man durch diese Klasse den Spielern initialisieren.

- **Attributen**

- **-int curID:** Es speichert, welche Spieler kommt. Es speichert den Identifikationsnummer des Spielers.
- **+SignalFlare sg:** Es ist der Signalfackel als Attribut in dieser Klasse.
- **+Item it:** Es ist der Gegenstand als Attribut in dieser Klasse.
- **+PlayerContainer:** Es ist die Spielercontainer als Attribut in dieser Klasse.
- **+SnowStorm SS:** Es ist der Schneesturm als Attribut in dieser Klasse.
- **+Tile t:** Die Klasse Gegenstand befindet sich in dieser Klasse.

- **Methoden**

- **+void init(int playerNum):** Mit dieser Methode können wir einen Spieler initialisieren. Dazu müssen wir ein Spieler Nummer eingeben. Diese Nummern identifizieren eindeutig den Spielern, also müssen unterschiedlich sein.
- **+void startNextRound():** Am Ende der Runde wird die Nächste gerufen. Der erste Spieler der Round setzt das Spiel fort.
- **+void endLastRound():** Die jetzige Runde beenden.
- **+void lose(String cause):** Das Spiel beendet. Der Grund der Niederlage wird eingegeben.
- **+PlayerContainer getPlayerContainer():** Diese Methode gibt den Player Container zurück.

4.3.13 SignalFlare

- **Verantwortung.**

Diese Klasse ist eigentlich die Signalfackel. Dieser Gegenstand besteht aus 3 Teile: Pistole, Leuchte, Patron. Mit diesem kann die Spieler das Spiel gewinnen.

- **Attributen**

- **-SignalFlarePart sfp[3]:** Es ist ein Array und dieser Array beinhaltet die 3 Teile vom Signalfackel. Der Größe von diesem Array ist fest 3, weil es 3 Teile vom Signalfackel gibt und kein Teil kann wegnehmen.

- **Methoden**

- **+void putTogether(RoundController rc):** Mit dieser Methode können die Spieler das Spiel beenden. Wenn alle Spieler in derselbe Eisplatte stehen, dann es kostet eine Arbeit diese Methode zu rufen und das Spiel zu beenden.

4.3.14 PlayerContainer

- **Verantwortung**

Diese Klasse beinhaltet die Spieler. Die Spieler werden durch ihren Identifikationsnummer eindeutig identifiziert.

- **Attributen**

- **-Player[] pid:** Alle Player wird in einer Container (*PlayerContainer*) auch speichert.

- **Methoden**

- **+Player getPlayer(int pid):** Diese Methode gibt ein Spieler zurück. Wir müssen den Identifikationsnummer dem bestimmten Spieler eingeben. Die verschiedenen Nummern werden zu verschiedenen Spielern zugeordnet.

4.3.15 SnowStorm

- **Verantwortung**

Diese Klasse erzeugt den Schneesturm. Dieser Sturm verringert die Körpertemperatur des Spielers, fall er sich nicht in einem Iglu und erhöht sich die Dicke der Schneeschichten auf die verschiedenen Eisplatten.

- **Methoden**

- **+void tryStorm():** Diese Methode erzeugt eigentlich den Schneesturm.

4.3.16 SignalFlarePart

- **Verantwortung**

Diese Klasse speichert ein Teil von Signalfackel. Es gibt 3 Teile und das Ziel des Spiels ist, diese 3 einzubauen und abzuschließen. Die verschiedenen Teile sind verschiedene Instanz von dieser Klasse.

- **Superklasse**

Item->SignalFlarePart

- **Attributen**

- **-int partID:** Die Identifikationsnummer von dem Teil. Es muss eindeutig sein.

- **Methode:**

- **+void used(Player player, Activity activity):** Die “used” Methode von der Superklasse (Item) wird überschrieben. (override)

4.3.16.1 PositionLUT

- **Verantwortung**

Diese Klasse handelt sich die Bewegungen von den Spielern. Den Spielern können mit Hilfe von dieser Klasse von einer Platte auf eine andere Platte springen. Diese Klasse ist ein so genannt “Singleton”, also in dem Spieler befindet sich nur eine von dieser Klasse. Des Weiteren können wir die Position von verschiedenen Spielern abfragen oder welchen Gegenständen befindet sich in einer bestimmten Eisplatte.

- **Attributen**

- **-Item[] itemTileMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays)
- **-Item[] tileItemMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays)
- **-Player[] playerTileMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays) Es gibt mindestens 3 Spieler (Es ist eine Anforderung, damit man das Spiel beginnen können), und maximum 7.
- **-Player[] tilePlayerMap:** Es kann zwischen 0 und 7 sein. (Die Größe des Arrays) Es gibt mindestens 3 Spieler (Es ist eine Anforderung, damit man das Spiel beginnen können), und maximum 7.
- **-Tile tileList[TILENUM]:** Die Eisplatten (alle) werden in dieser Klasse (*PositionLUT*) in diesem Array gespeichert. Es ist TILENUM, weil es fest TILENUM Eisplatte gibt.

- **Methoden**

- **+Tile getPosition(Player p):** Mit dieser Methode können wir die Position des bestimmten Spieler abfragen. Wir müssen einen Spieler eingeben und es wird zurückgegeben, auf welche Eisplatte steht er.
- **+Tile getPosition(Item i):** Mit dieser Methode können wir die Position des bestimmten Gegenstand abfragen. Wir müssen einen Gegenstand eingeben und es wird zurückgegeben, auf welche Eisplatte befindet sich es.
- **+Player[] getPlayersOnTile(Tile t):** Diese Methode ergibt eine Spieler Array. In diesem Array befindet sich ein Spieler, falls er auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.

- **+Item[] getItemOnTile(Tile t):** Diese Methode ergibt eine Gegenstand Array. In diesem Array befindet sich ein Gegenstand, falls es auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.
- **+void setPosition(Player p):** Mit dieser Methode können wir die Position von dem eingegebenen Spieler einstellen.
- **+void setPosition(Item i):** Mit dieser Methode können wir die Position von dem eingegebenen Gegenstand einstellen. Es kann vorkommen, wenn zum Beispiel ein Spieler ein Essen isst. Dann wird das Essen auf eine zufällige Eisplatte generiert.
- **+Tile getTile(int x, int y):** Diese Methode gibt eine bestimmten Gegenstand zurück. Dieser Gegenstand wird als Parameter eingegeben.

4.3.17 Tile

- **Verantwortung**

Diese Klasse ist ein Container, also zusammenfasst alle Art von den Eisplatten. Die Klasse ist abstrakt, man kann es nicht instanzieren. Es gibt 3 Arten von den Eisplatten. Es kann stabil, instabil oder ein schneebedecktes Loch sein. Diese Arten sind verschiedene Subklassen, vererben diese abstrakten Klasse.

- **Attributen**

- **#int x:** Es ist die x Koordinate (horizontale) von der bestimmten Eisplatte.
- **#int y:** Es ist die y Koordinate (vertikale) von der bestimmten Eisplatte.
- **#int snow:** Diese Attribute speichert, wie viel Schnee (wie viel Einheit) sich auf die Eisplatte befindet.
- **-boolean igluOn:** Diese Attribute bestimmt, ob ein Iglu sich auf die Eisplatte befindet. (true: Ja, false: Nein)

- **Methoden**

- **+void steppedOn(Player p):** Ein Spieler tritt auf die Eisplatte. Wir müssen den Spieler eingeben.
- **+void steppedOff(Direction dir):** Ein Spieler tritt auf eine andere Eisplatte. Wir müssen die Richtung eingeben. Direction ist eine Enumeration, also von bestimmten Richtungen (4 Himmelsrichtung) können wir wählen.
- **+int changeSnow(signed int thisMuch):** Die Höhe des Schneeschicht verändert sich. Es wird eingegeben, mit wie vielen. Es kann höher oder kleiner sein. Zum Beispiel höher, falls ein Schneesturm kommt.
- **+Tile getNeighbour(Direction dir):** Mit dieser Methode können wir die benachbarten Eisplatten von einer Platte abfragen. Wir müssen die Richtung auswählen und eingeben.
- **+void destroyIglu():** Diese Methode baut den Iglu, der auf diese Eisplatte steht, ab.
- **+void buildIglu():** Eskimos können auf eine bestimmten Platte ein Iglu bauen.

4.3.18 SnowyHole

- **Verantwortung**

Diese Klasse spezialisiert die "Tile" Klasse. Diese Klasse verwirklicht das schneebedeckte Loch. Diese Eisplatten siehe so aus, wie die andere schneebedecktes Platte, aber wenn ein Spieler auf diese tritt, fällt ins Wasser. Auf diesen Platten können 0 Spieler stehen.

- **Superklasse**

Tile-> SnowHole

4.3.19 StableTile

- **Verantwortung**

Diese Klasse spezialisiert die “Tile“ Klasse. Diese Klasse verwirklicht die stabile Eisplatte. Auf diesen Platten können unendlich viel Spieler stehen. Auf diesen Platten kann sich natürlich auch Schnee befinden.

- **Superklasse**

Tile-> SnowHole

4.3.20 UnstableTile

- **Verantwortung**

Diese Klasse spezialisiert die “Tile“ Klasse. Diese Klasse verwirklicht die instabile Eisplatte. Auf diesen Platten können nur begrenzt viel Spieler stehen. Die Anzahl den Spielern ist nicht sichtbar, aber die Forscher können es anschauen.

- **Superklasse**

Tile-> SnowHole

- **Attributen**

- **-int capacity:** So viele Spieler können auf diese Platte stehen. Dieser Wert ist größer gleich null und wird zufällig eingestellt.
- **+int standingHere:** Die Anzahl der Spieler, die auf diese Platte stehen. Es kann nicht größer oder gleich als “capacity” sein.

4.3.21 Direction

- **Verantwortung**

Diese Klasse ist eine Enumeration und beinhaltet die verschiedenen Richtungen. Wenn ein Spieler treten möchte, sollte er von diesen Richtungen wählen. Es ist eigentlich die 4 Himmelrichtungen. Andere Richtung existiert in diesem Spiel nicht, also können die Spieler diagonale weise nicht treten.

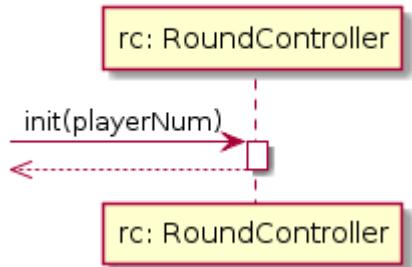
- **Aufzählungsliterale**

- **up:** Auf die nördliche Platte treten. (Nord)
- **down:** Auf die untere Platte treten. (Süd)
- **left:** Auf die linke Platte treten. (West)
- **right:** Auf die rechte Platte treten. (Ost)

4.4 Sequenz Diagramme

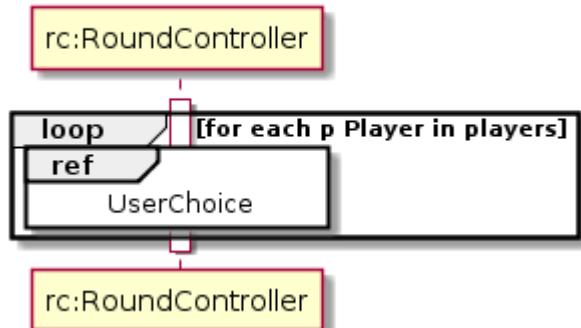
4.4.1 Round control initialization

Round control initialisation

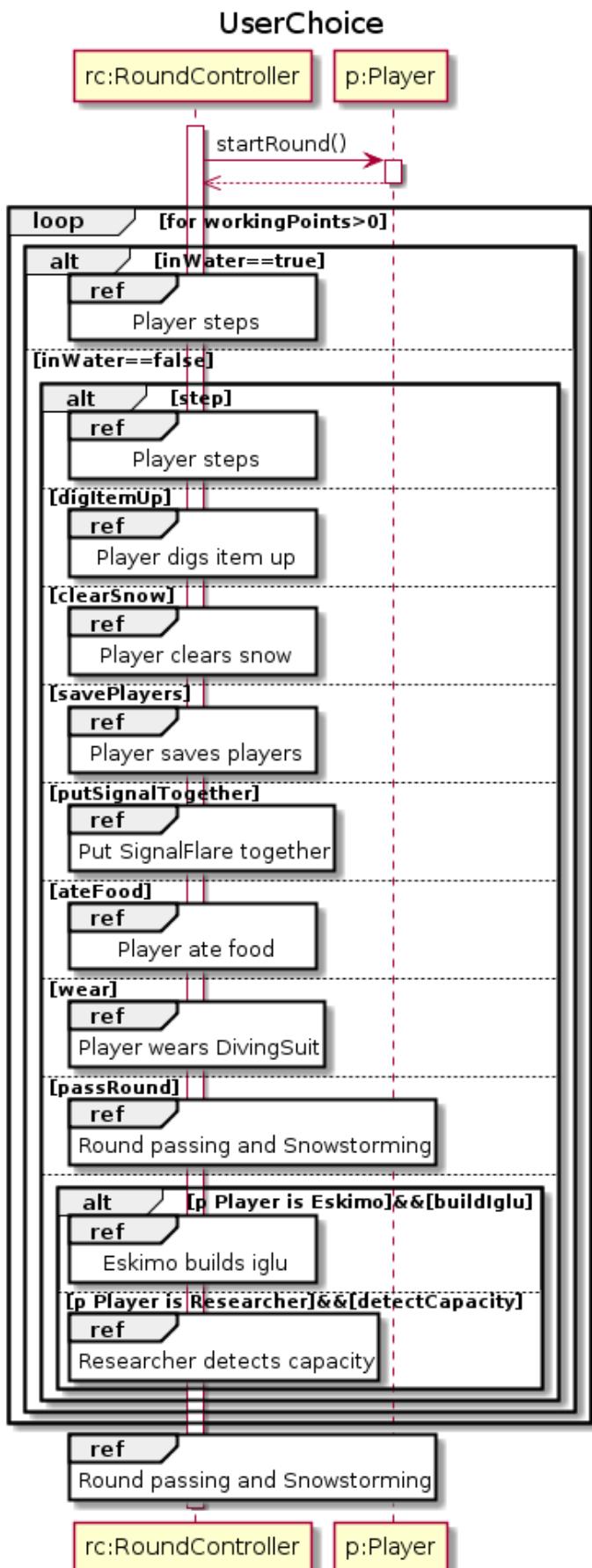


4.4.2 Round control

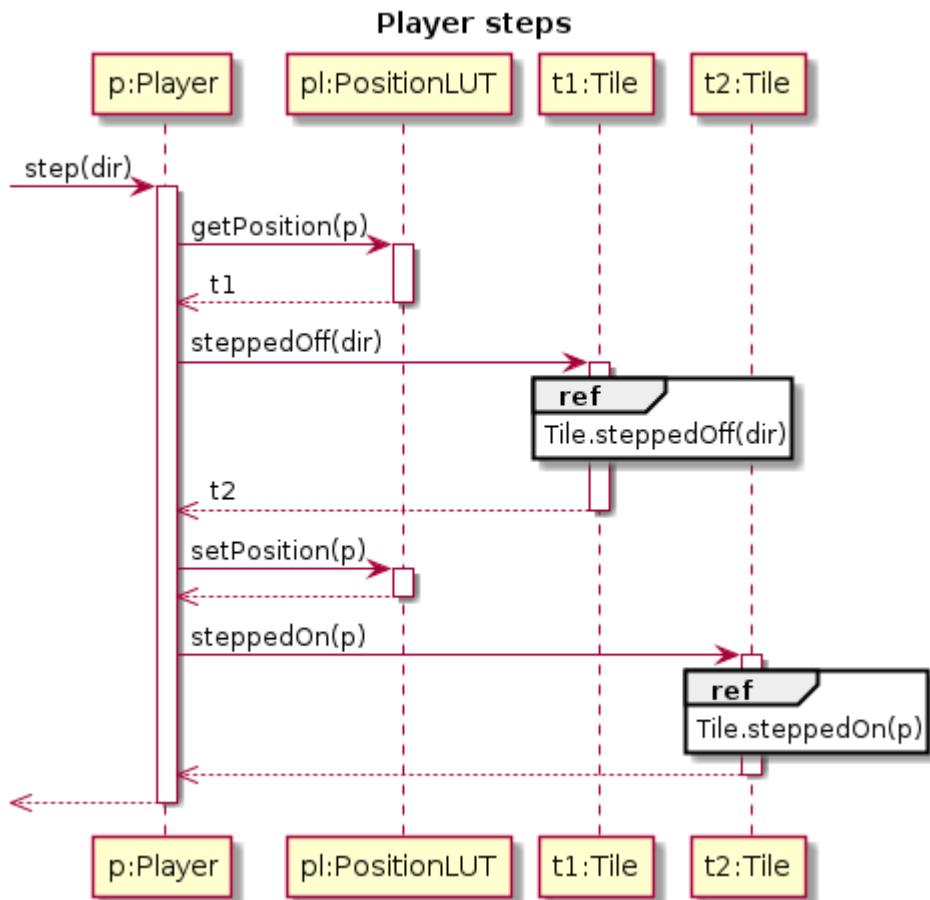
Round control



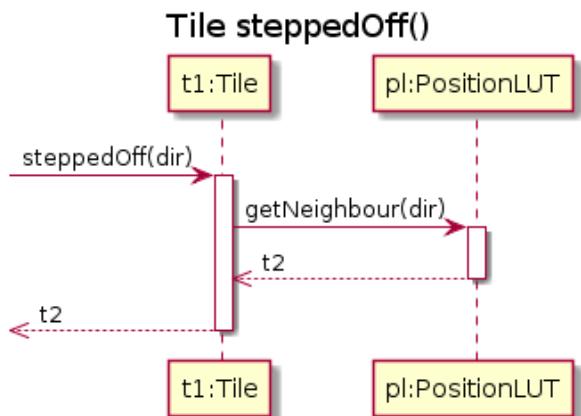
4.4.3 UserChoice



4.4.4 Player steps

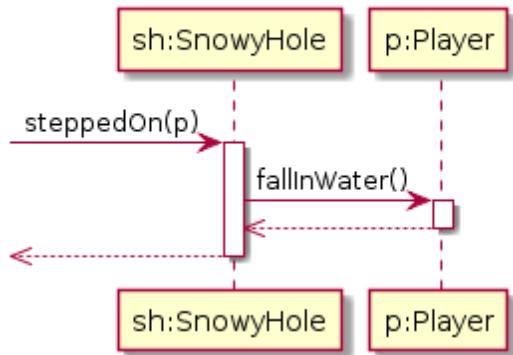


4.4.5 Tile steppedOff



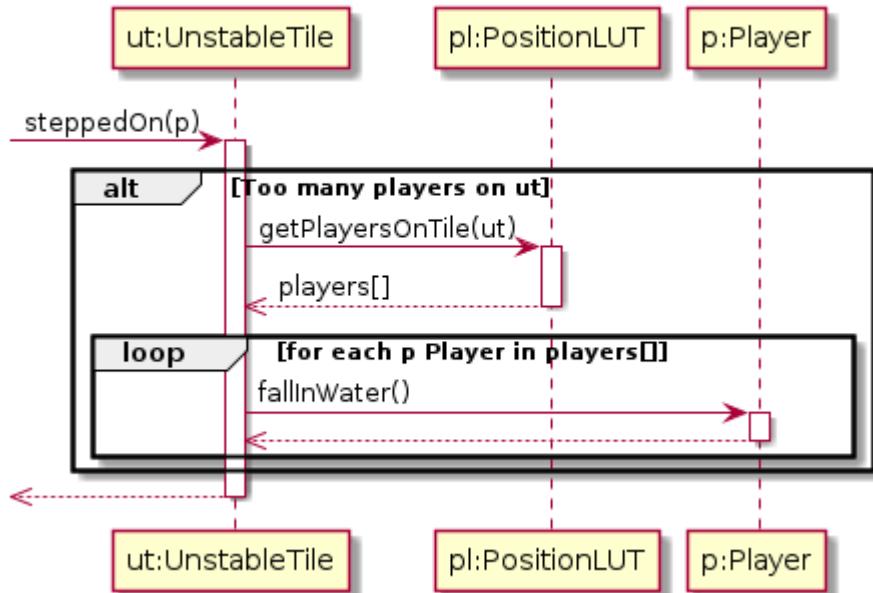
4.4.6 SnowyHole steppedOn

SnowyHole steppedOn()

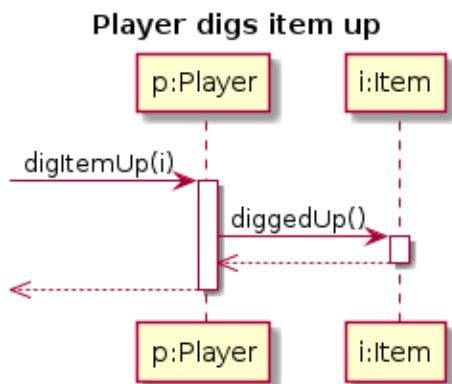


4.4.7 UnstableTile steppedOn

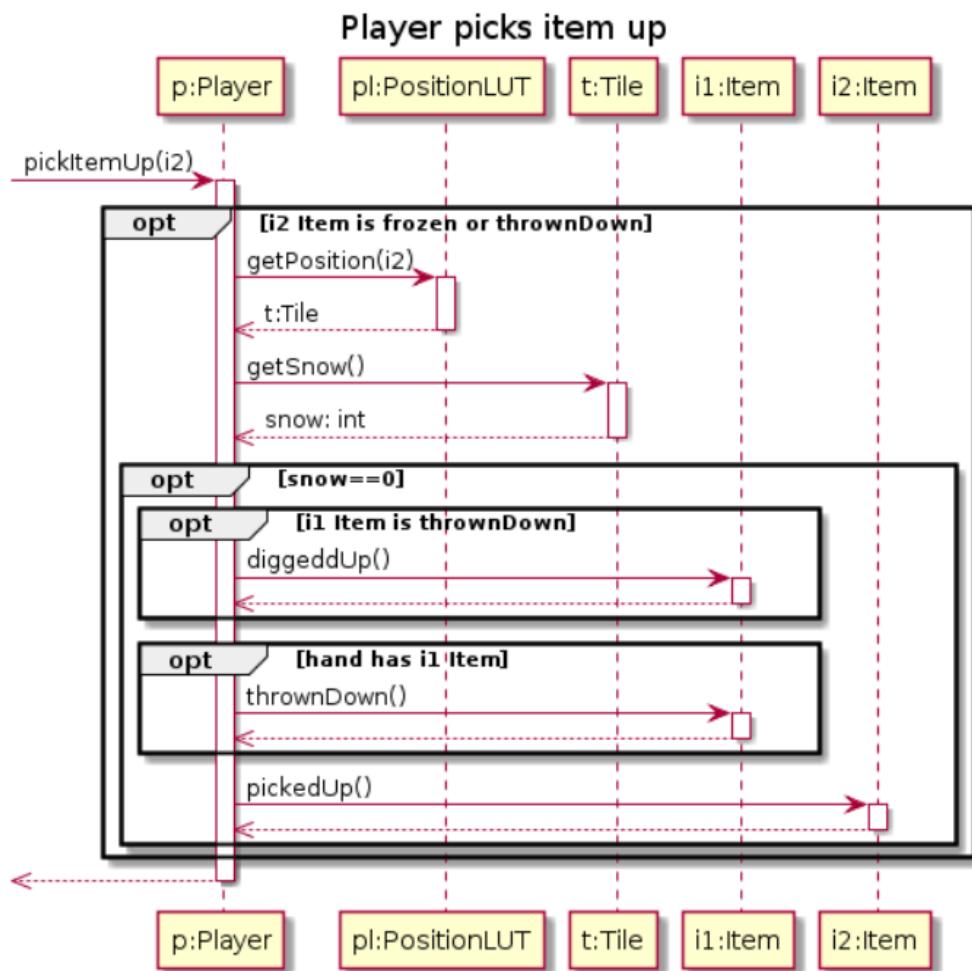
UnstableTile steppedOn()



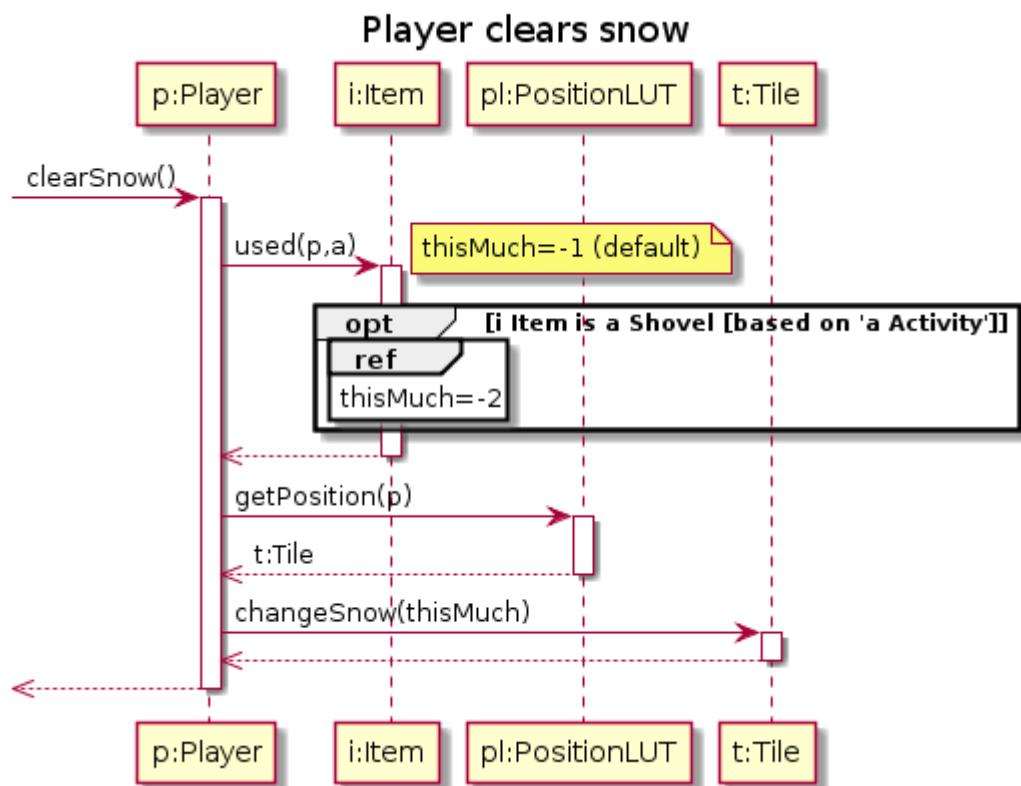
4.4.8 Player digs item up



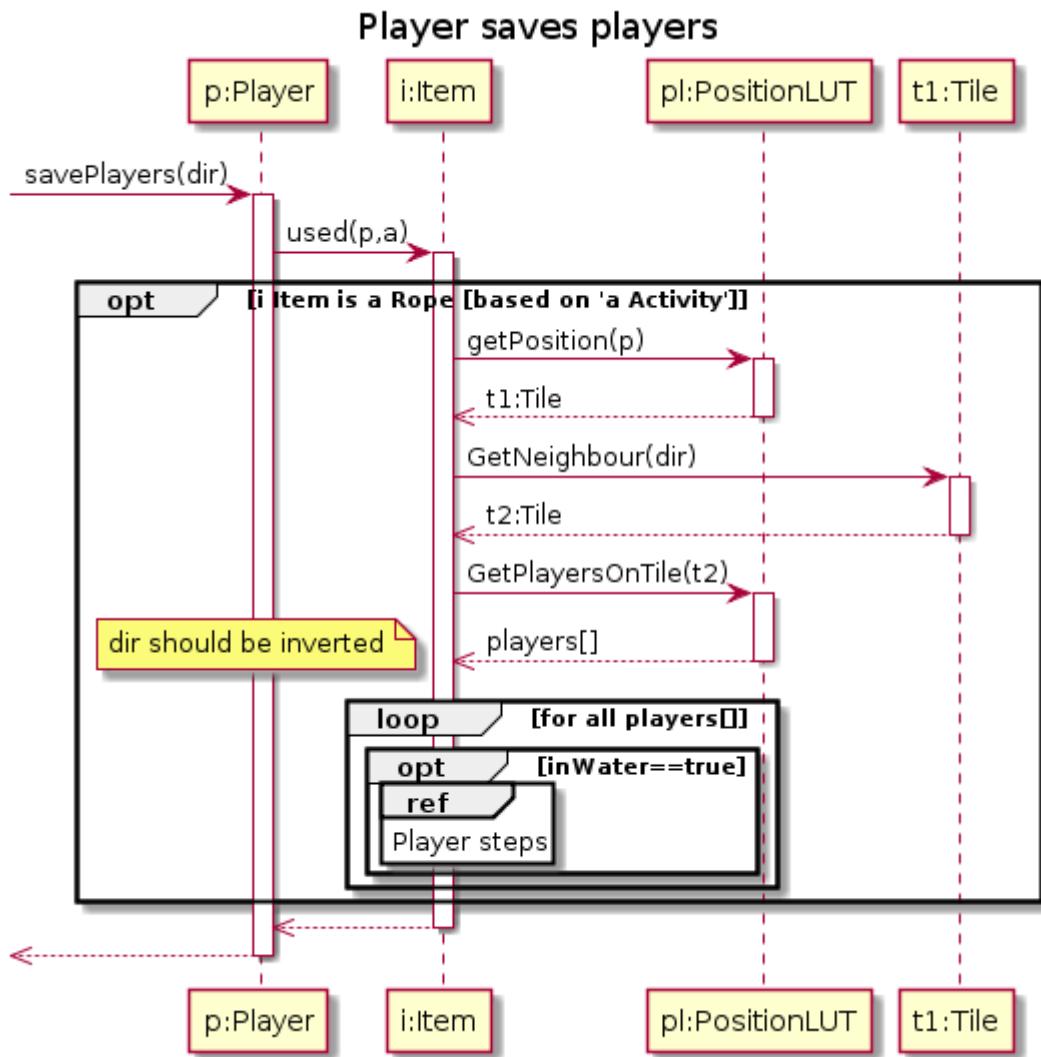
4.4.9 Player picks item up



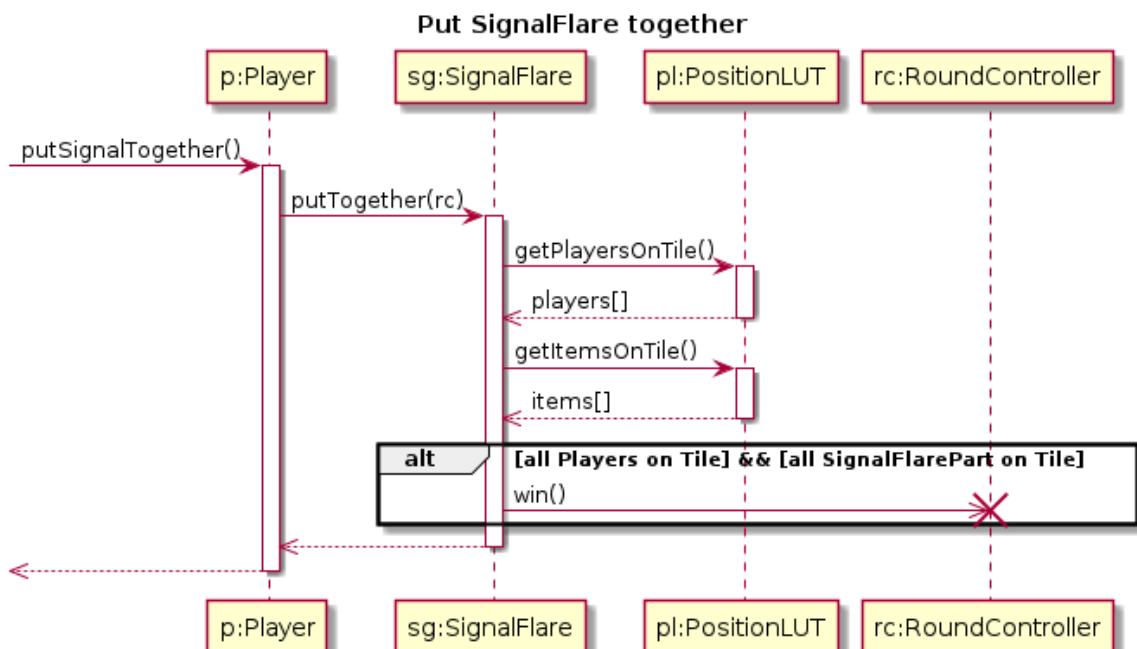
4.4.10 Player clears snow



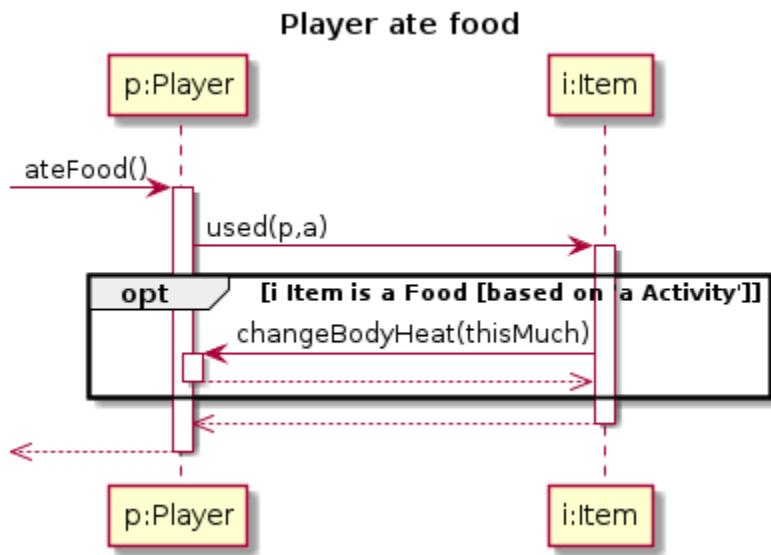
4.4.11 Player saves players



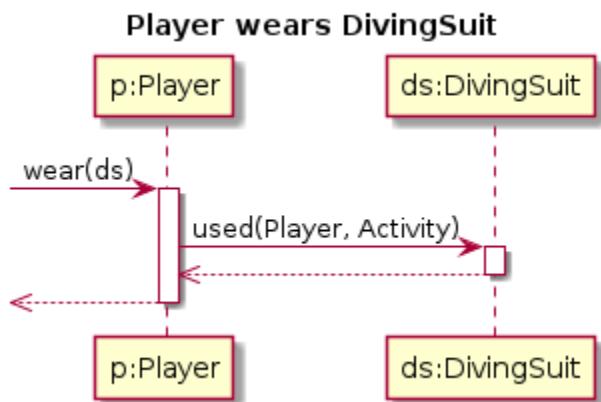
4.4.12 Put SignalFlare together



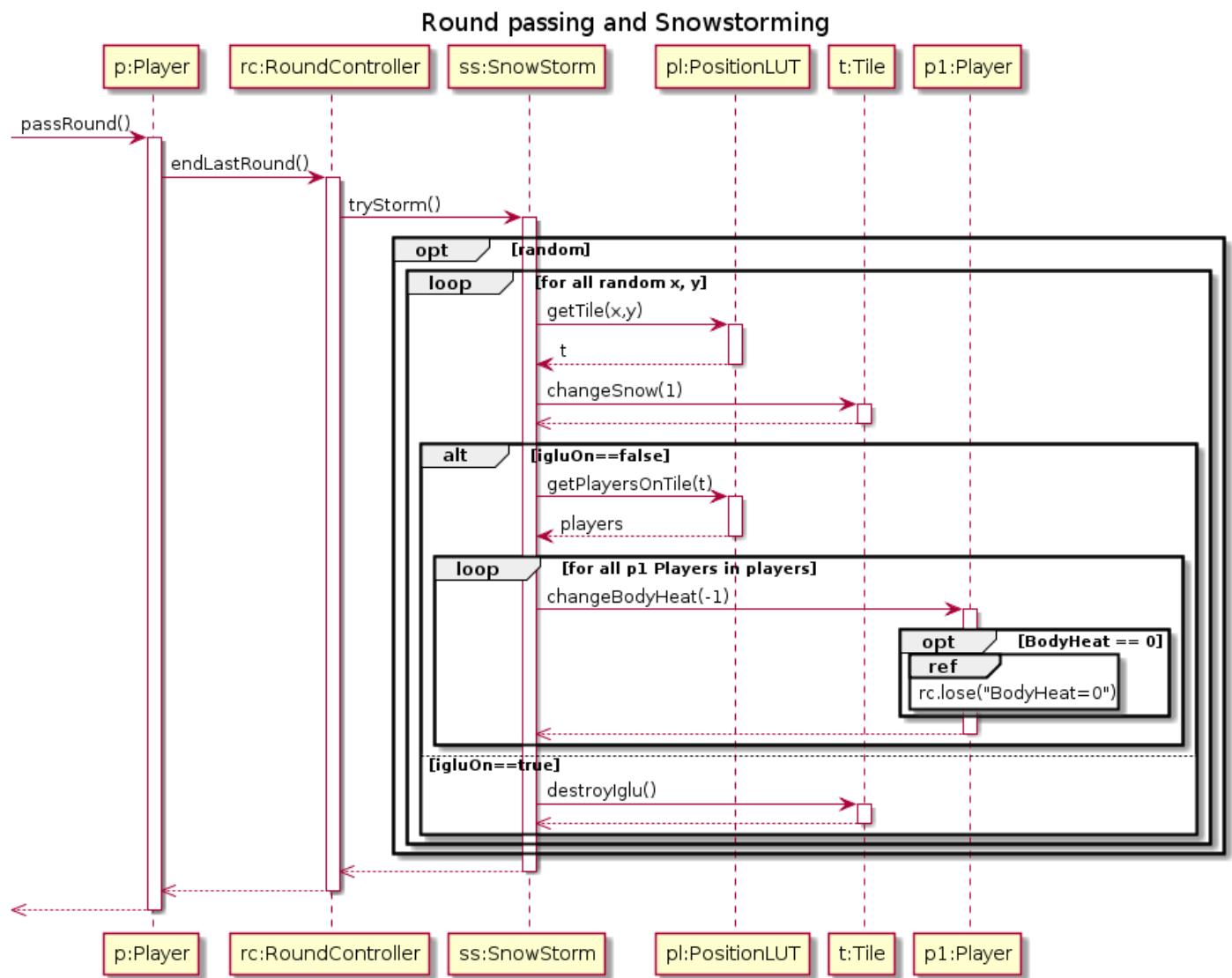
4.4.13 Player ate food



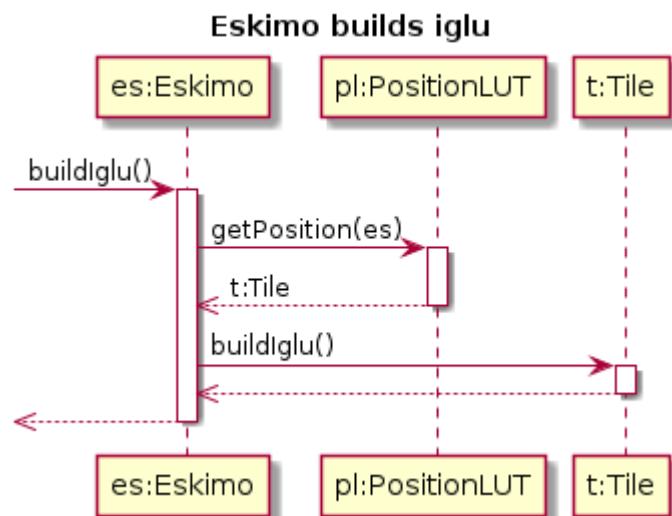
4.4.14 Player wears DivingSuit

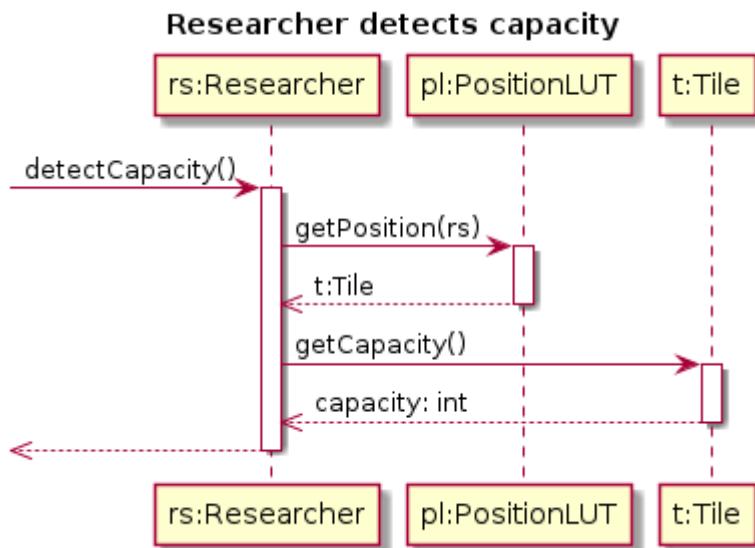


4.4.15 Round passing and Snowstorming



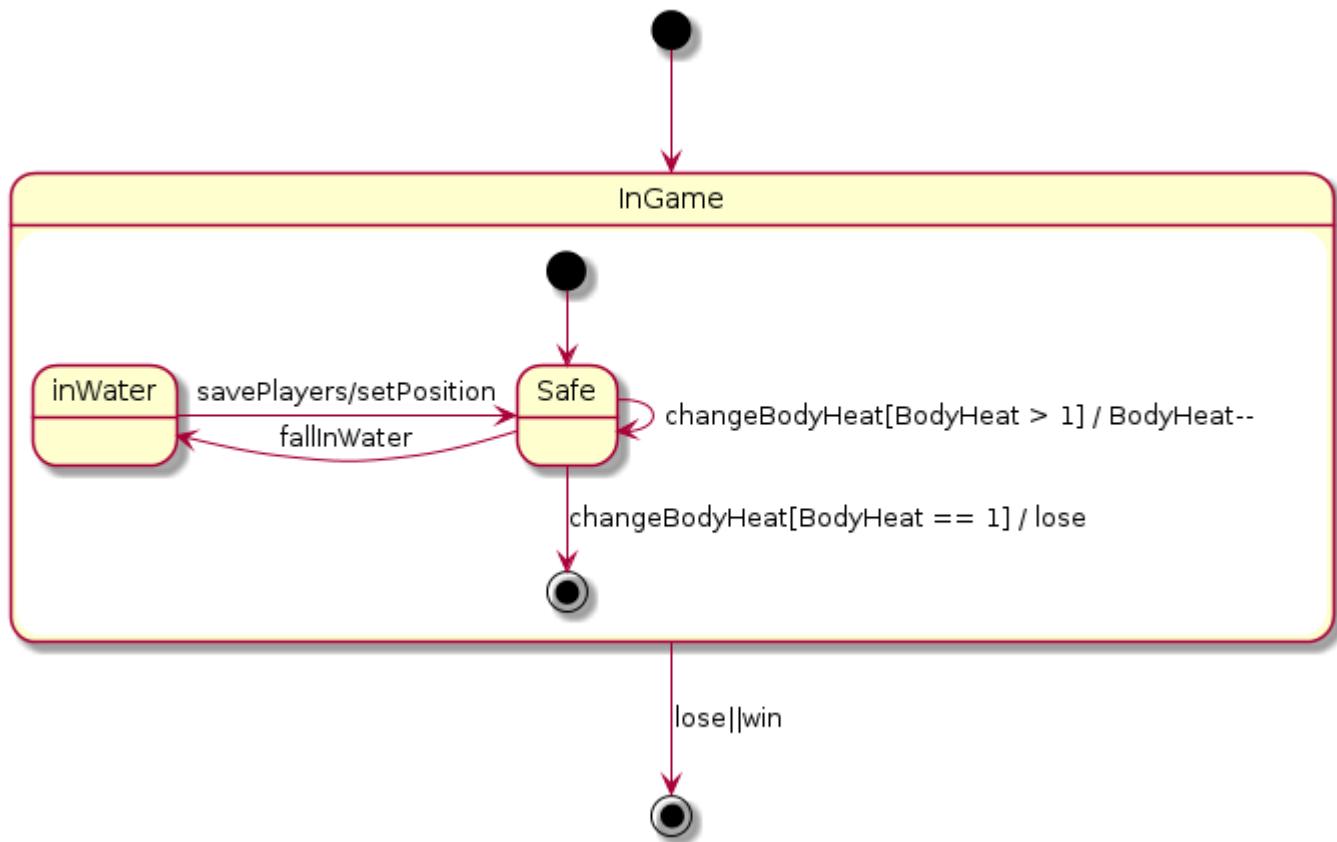
4.4.16 Eskimo builds iglu



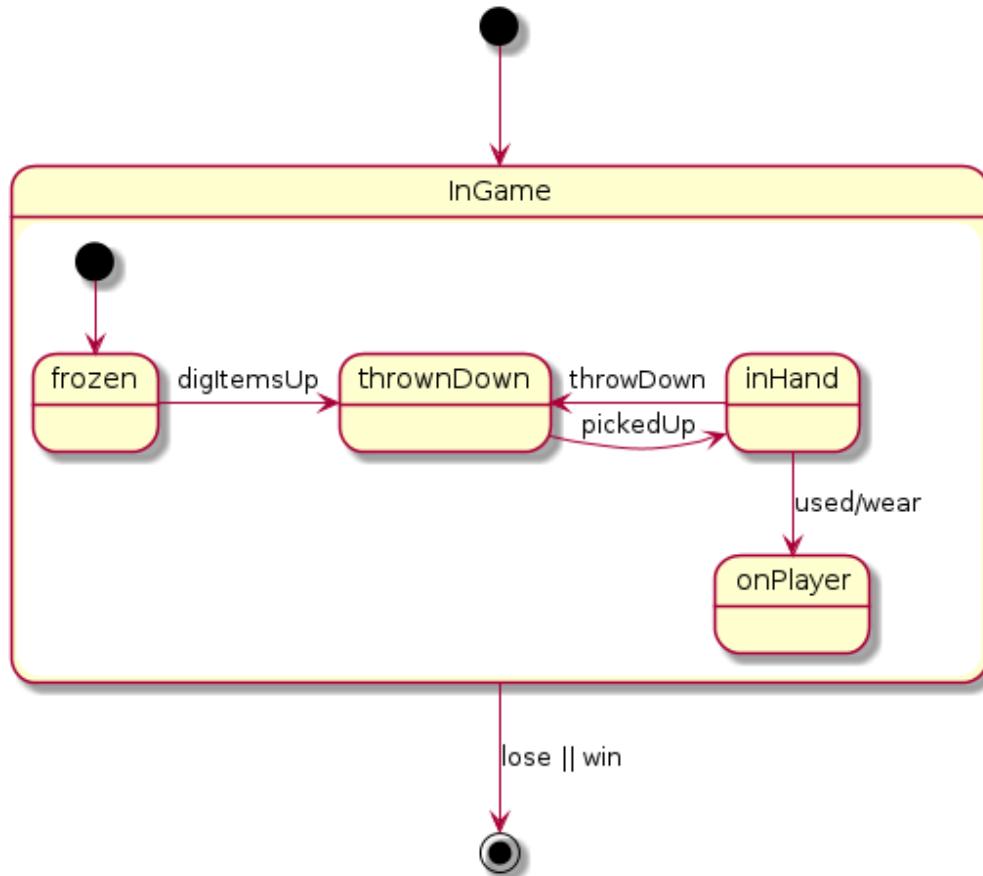


4.5 Zustandsdiagramme (State-charts)

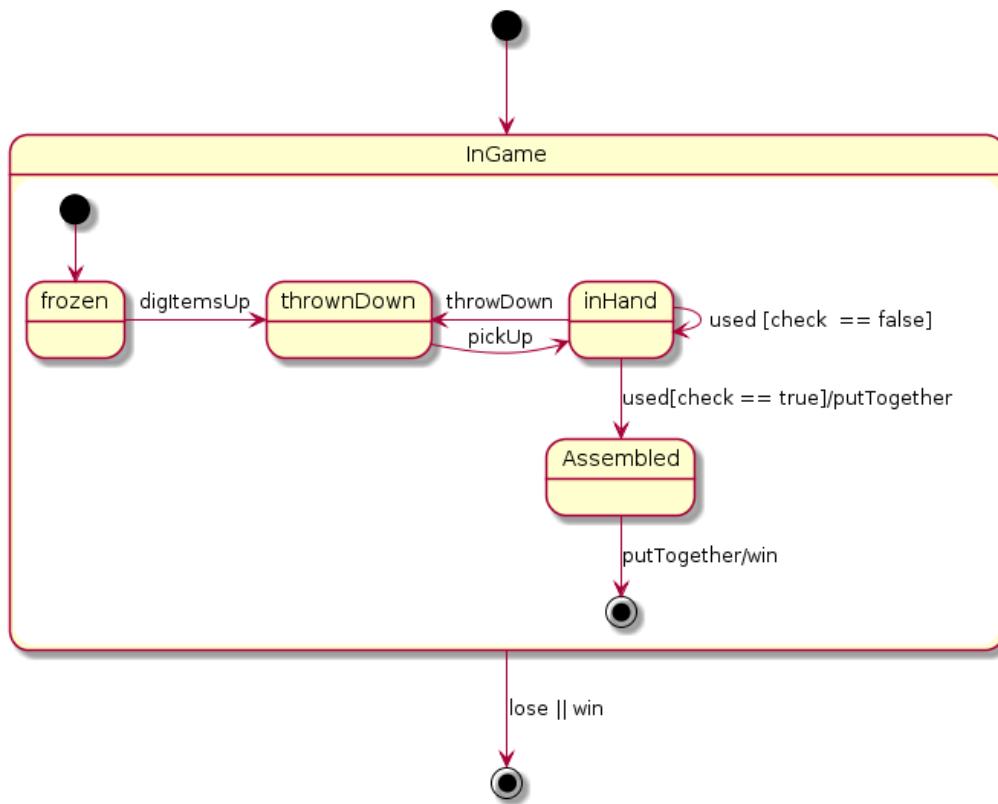
4.5.1 PlayerStates:



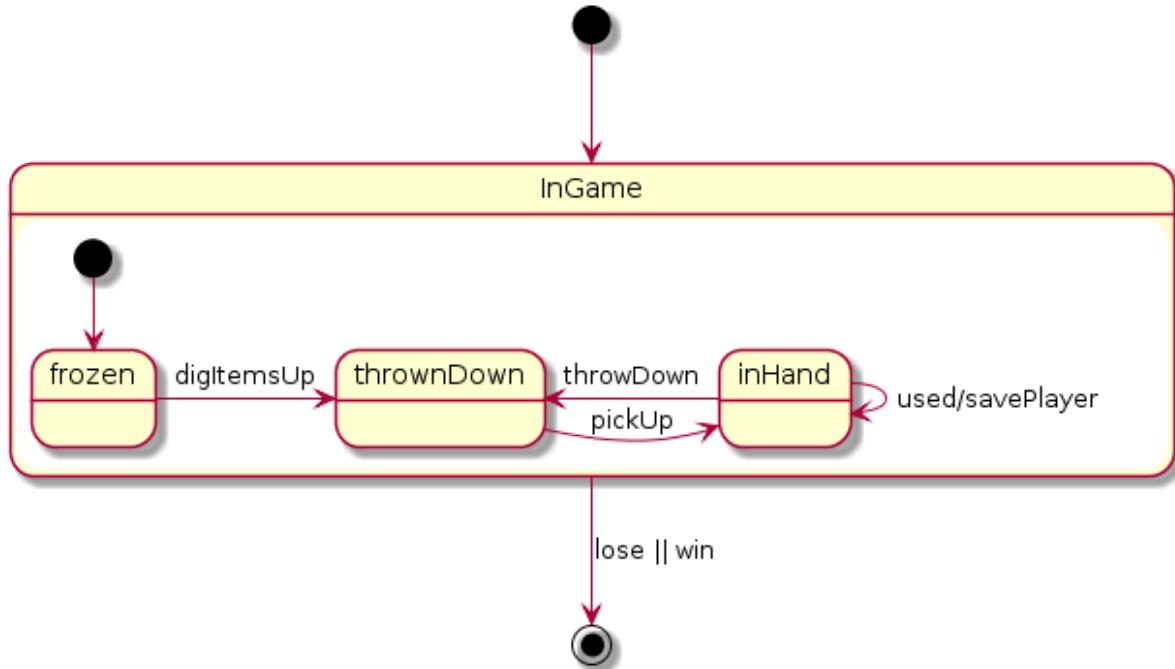
4.5.2 ItemStates(DivingSuit):



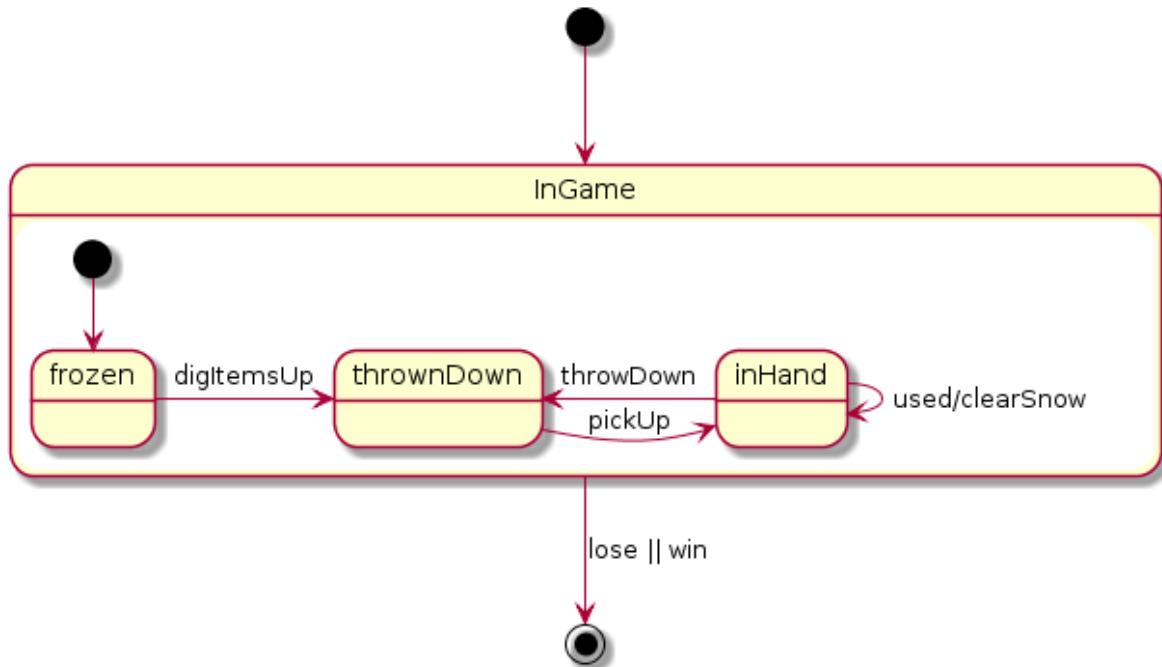
4.5.3 ItemStates(SignalFlarePart)



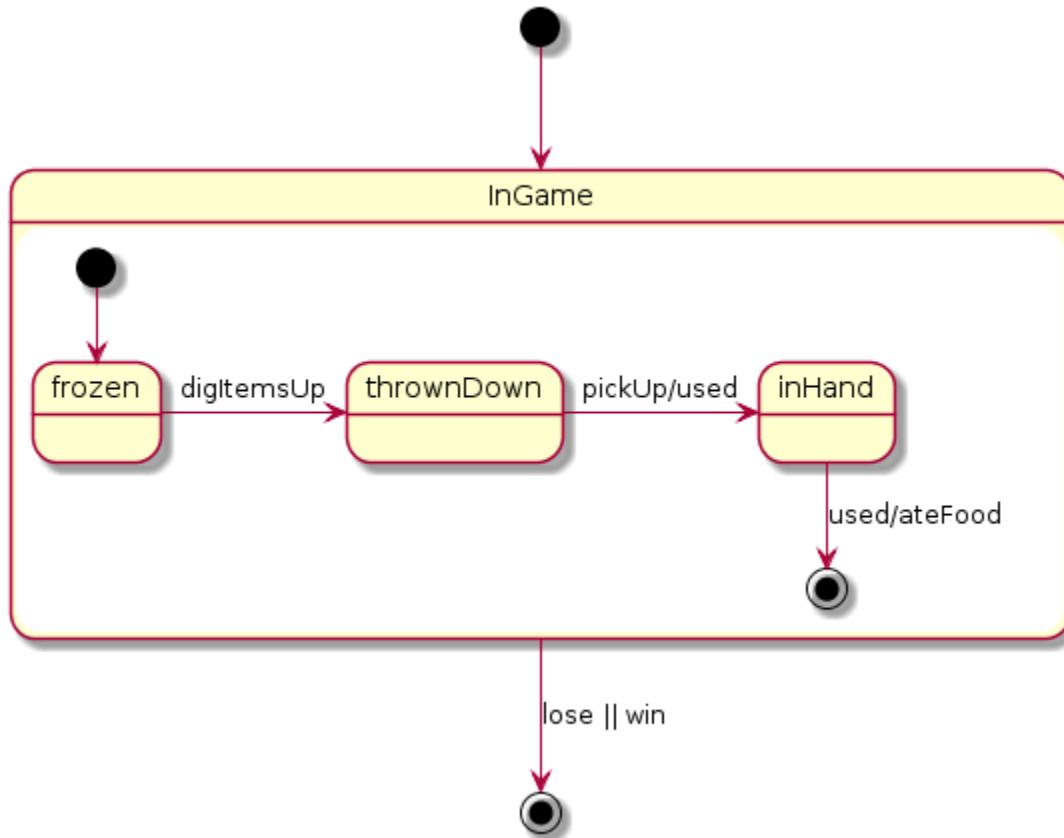
4.5.4 ItemStates(Rope):



4.5.5 ItemStates(Shovel):



4.5.6 ItemStates(Food)



5. Planung des Skeletons

DeutschOverflow

Supervisor:
Kovács Márton

Members:

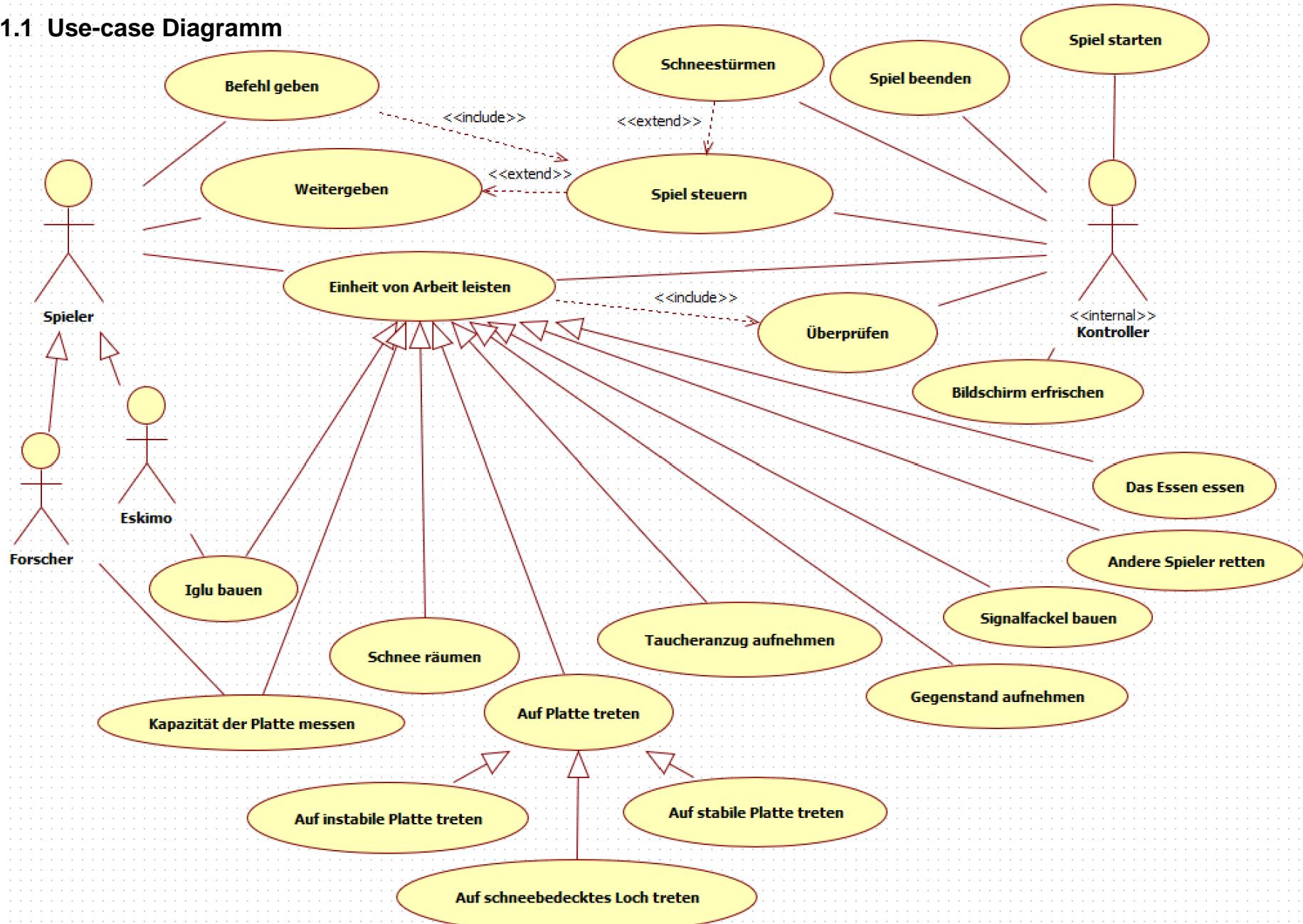
Ádám Zsófia	SOSK6A	adamzsolfi.mail@gmail.com
Hedrich Ádám	H9HFFV	hedrichadam09@gmail.com
Pintér Balázs	ZGY18G	pinterbalazs21@gmail.com
Fucskár Patrícia	XKYAOO	fucskar.patricia@gmail.com
Tassi Timián	MYY53U	timian.tassi@gmail.com

1. April 2020

5. Planung des Skeletons

5.1 Die reale Use-cases der Skeleton Modell

5.1.1 Use-case Diagramm



5.1.2 Use-case Beschreibungen

5.1.3 Use-case Beschreibungen

Name von Use-case	Spiel starten
Kurze Beschreibung	Kontroller startet das Spiel
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller initiiert das Spiel 2. Kontroller zeigt das Spiel an Spielern

Name von Use-case	Spiel beenden
Kurze Beschreibung	Kontroller beendet das Spiel
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller schließt das Spiel ab 2. Kontroller schließt das Fenster des Spieles

Name von Use-case	Spiel steuern
Kurze Beschreibung	Kontroller steuert & koordiniert das Spiel
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Das System wählt den neuen Spieler aus 2. Das System zeigt den aktiven Spieler
Alternative Tätigkeit	1.A.1. Das System generiert einen Schneesturm

Name von Use-case	Befehl geben
Kurze Beschreibung	Spieler gibt einen Befehl
Akteur	Spieler
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler gibt einen Befehl 2. Das System speichert den Befehl des Spielers

Name von Use-case	Bildschirm erfrischen
Kurze Beschreibung	Kontroller erfrischt den Benutzeroberflächenbildschirm
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Das System stellt den Bildschirm zusammen 2. Das System zeigt den Bildschirm an Spieler

Name von Use-case	Einheit von Arbeit leisten
Kurze Beschreibung	Spieler leistet eine Einheit von Arbeit
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler wählt den Typ von Arbeit: eine Einheit Schnee von Platte entfernen, auf eine benachbarte Eisplatte treten, einen Gegenstand von Platte aufnehmen, Verwendung einer Spezialfähigkeit 3. Das System speichert die Auswahl 4. Das System führt den gewählten Arbeitsprozess aus

Spezifizierungen von Einheit von Arbeit leisten:

Name von Use-case	Auf Platte treten
Kurze Beschreibung	Spieler tritt von seiner Platte auf eine Nachbarplatte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt nach einer Richtung, um Nachbarplatte treten 2. Das System fragt die Position von Player ab 3. Das System entfernt den Spieler von Platte 4. Das System stellt die neue Position von Spieler 5. Das System ordnet den Spieler zur Platte 6. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 1.A.1. Der Spieler zeigt nach einer Richtung, wo es keine Platte mehr gibt (Randplatte) 1.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit

Name von Use-case	Auf instabile Platte treten
Kurze Beschreibung	Spieler tritt auf instabile Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt nach einer Richtung, um Nachbarplatte treten 2. Das System fragt die Position von Player ab 3. Das System entfernt den Spieler von Platte 4. Das System stellt die neue Position von Spieler 5. Das System ordnet den Spieler „zur Platte“ 6. Das System misst die Kapazität der Platte 7. Das System hat sichergestellt, dass die Platte so viele Spieler tragen kann 8. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 6.A.1. Das System hat die Kapazität ist nicht genug für so viele Spieler 6.A.2. Das System stellt für alle Spieler, die sich „in Platte befinden“ „inWasser“ Parameter ein 6.A.3. Das System erniedrigt Gesamteinheit von Arbeit der aktuellen Spieler mit 1 Arbeitseinheit

Name von Use-case	Auf schneebedecktes Loch treten
Kurze Beschreibung	Spieler tritt auf schneebedecktes Loch
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt nach einer Richtung, um Nachbarplatte treten 2. Das System fragt die Position von Player ab 3. Das System entfernt den Spieler von Platte 4. Das System stellt die neue Position von Spieler 5. Das System ordnet den Spieler „zur Platte“ 6. Das System stellt für den Spieler „inWasser“ Parameter ein 7. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit

Name von Use-case	Auf stabile Platte treten
Kurze Beschreibung	Spieler tritt auf stabile Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt nach einer Richtung, um Nachbarplatte treten 2. Das System fragt die Position von Player ab 3. Das System entfernt den Spieler von Platte 4. Das System stellt die neue Position von Spieler 5. Das System ordnet den Spieler „zur Platte“ 6. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit

Name von Use-case	Gegenstand aufnehmen
Kurze Beschreibung	Spieler nimmt einen Gegenstand auf
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt einen Gegenstand, -der auf Platte liegt-, aufzunehmen 2. Das System stellt sicher der Spieler keinen Gegenstand in Hände hat 3. Das System ordnet den Gegenstand zum Spieler 4. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 2.A.1. Der Spieler hat schon etwas in seiner Hand 2.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit

Name von Use-case	Schnee räumen
Kurze Beschreibung	Spieler räumt Schnee von Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: Schnee von Platte entfernen 2. Das System fragt die Position von Player ab 3. Das System erniedrigt Schnee der Platte mit einer Schneeeinheit 4. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 1.A.1. Der Spieler zeigt an Schaufel an seiner Hand, um Schnee von Platte entfernen 1.A.2. Das System fragt die Position von Player ab 1.A.3. Das System erniedrigt Schnee der Platte mit 2 Schneeeinheit 1.A.4. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit

Name von Use-case	Andere Spieler retten
Kurze Beschreibung	Spieler rettet andere Spieler
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler zeigt an seinem Seil 2. Der Spieler gibt eine Richtung an 3. Das System fragt die Position von Player ab 4. Das System fragt „die Nachbarplatte“ nach gegebener Richtung ab 5. Das System fragt die Spieler - die im Wasser an der Platte sind – ab 6. Das System stellt die Player auf die Platte des Helfers 7. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 4.A.1. Das System hat keine Spieler im Wasser gefunden 4.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit

Name von Use-case	Das Essen essen
Kurze Beschreibung	Spieler isst sein Essen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: er möchte seinen Essen von seiner Hand essen 2. Das System erhöht die Körpertemperatur vom Spieler mit einer Körpertemperatureinheit 3. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit

Name von Use-case	Signalfackel bauen
Kurze Beschreibung	Spieler(n) baut das Signalfackel zusammen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: Signalfackel bauen 2. Das System fragt die Spieler der Platte ab 3. Das System fragt die Gegenstände – die auf die Platte gelegt sind- ab 4. Das System wertet die Bedingung [alle Spieler sind in einer Platte und alle Bestandteile der Signalfackel sind auf dieser Platte vorhanden] aus 5. Das System alles richtig gefunden, benachrichtigt die Spieler über den Gewinn 6. Das System schließt das Spiel ab
Alternative Tätigkeit	<p>4.A.1. Das System hat die Bedingung nicht richtig gefunden</p> <p>4.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit</p>

Name von Use-case	Taucheranzug aufnehmen
Kurze Beschreibung	Spieler kleidet sich in Taucheranzug
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: den Taucheranzug aufnehmen 2. Das System ordnet zum Spieler den Taucheranzug 3. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit

Name von Use-case	Iglu bauen
Kurze Beschreibung	Spieler baut ein Iglu auf die Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: Iglu bauen 2. Das System fragt die Position des Spielers ab 3. Das System stellt sicher, dass der Spieler nicht im Wasser steht 4. Das System stellt ein Iglu auf die Platte 5. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<p>3.A.1. Das System bringt ans Licht, dass der Spieler steht im Wasser steht</p> <p>3.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit</p>

Name von Use-case	Kapazität der Platte messen
Kurze Beschreibung	Spieler misst die Kapazität der Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler signalisiert seine Absicht: Kapazität der Platte messen 2. Das System fragt die Position des Spielers ab 3. Das System stellt sicher, dass der Spieler nicht im Wasser steht 4. Das System gibt die Kapazität von Platte an 5. Das System erniedrigt Gesamteinheit von Arbeit der Spieler mit 1 Arbeitseinheit
Alternative Tätigkeit	<ol style="list-style-type: none"> 3.A.1. Das System bringt ans Licht, dass der Spieler steht im Wasser steht 3.A.2. Das System warnt den Spieler über den Misserfolg der Tätigkeit

Name von Use-case	Weitergeben
Kurze Beschreibung	Spieler gibt das Spiel weiter
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Der Spieler benutzt vollständig seinen Gesamtarbeit (4 Einheit) 2. Das System speichert, dass der Spieler seine Arbeit geleistet hat
Alternative Tätigkeit	<ol style="list-style-type: none"> 1.A.1. Der Spieler gibt das Spiel weiter (Arbeit übrig), sagt „ÜBERGEBEN“ 1.A.2. Das System speichert, dass der Spieler seine Arbeit geleistet hat

Name von Use-case	Überprüfen
Kurze Beschreibung	Kontroller überprüft das Spiel
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Das System hat überprüft die Bedingungen: [Signalfackel nicht zusammengebaut wurde], [keine der Spieler hat Körpertemperatur 0], [keine der Spieler stirbt im Wasser] 2. Das System hat alles richtig gefunden
Alternative Tätigkeit	<ol style="list-style-type: none"> 1.A.1. Team hat die Signalfackel aufgebaut 1.A.2. Das System benachrichtigt die Spieler über den Gewinn des Spieles 1.A.3. Das System schließt das Spiel ab 1.B.1. Jemand abkühlt, also Körpertemperatur eines Spielers ist 0. 1.B.2. Das System benachrichtigt die Spieler über den Verlust des Spieles 1.B.3. Das System schließt das Spiel ab 1.C.1. Jemand stirbt im Wasser 1.C.2. Das System benachrichtigt die Spieler über den Verlust des Spieles 1.C.3. Das System schließt das Spiel ab

Name von Use-case	Schneestürmen
Kurze Beschreibung	Kontroller generiert einen Schneesturm
Akteur	Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Das System wählt zufällige Platten aus 2. Das System inkrementiert die Menge der Schnee auf generierten Platten 3. Das System fragt ab, ob ein Iglu auf die einzelnen Platten steht 4. Das System hat kein Iglu auf die einzelnen Platten gefunden 5. Das System fragt die Spieler ab, die in den generierten Platten liegen 6. Das System dekrementiert die Körpertemperatur der Spieler 7. Das System stellt sicher, ob jemanden abkühlt ist 8. Das System hat keine Spieler gefunden, die abkühlt sind
Alternative Tätigkeit	<p>3.A.1. Das System hat Iglu auf der einzelnen Platte gefunden</p> <p>7.A.1. Das System hat jemandem gefunden, der abkühlt ist</p> <p>7.A.2. Das System informiert die Spieler über den Verlust des Spieles</p> <p>7.A.3. Das System schließt das Spiel ab</p>

5.2 Der Plan vom Bedienfeld des Skeletons, Dialogen

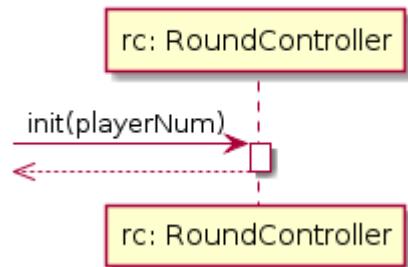
Eine Konsole UI wird zum testen benutzt. Deshalb schreibt dieses Testprogramm ein hierarchisches Menu aus und mit die Auswahl des entsprechendes Zahlen lässt es die verschiedenen Tests laufen und schreibt die ausgerufte Funktionen und das Entwertung aus. Dieses Menu wird so aussehen:

- **Hauptmenu**
 1. Spieler Tätigkeiten
 2. Sturm
- **Möglichkeiten (Testfälle) in ‘1. Spieler Tätigkeiten’**
 - (1.) 1 Schritt
 - Auf stabilen Eisplatte
 - Auf instabilen Eisplatte (Kapazität ok/zu viel)
 - Auf Loch (ins Wasser fallen)
 - (2.) Schnee aufräumen (*mit/ohne Schaufel*)
 - (3.) ein Gegenstand ausgraben
 - (4.) seinen Kumpel retten
 - (5.) Speziale Fähigkeit benutzen (*Eskimo/Forscher*)
 - (6.) Gegenstand aufnehmen (*mit verschiedene Gegenstände*)
 - (7.) Signalfackel zusammenbauen
 - (8.) Sterben (*im Wasser/wegen Verletzung*)
- **Möglichkeiten (Testfälle) in ‘2. Sturm’ - in jede Testfälle benutzen wir eine bestimmte Tile mit die gebrauchte Elemente/objekte darauf**
 - (1.) Iglu wird zerstört
 - (2.) Spieler wird verletzt
 - (3.) frisches Schnee wird auf Feld hingelegt(2.)

5.3 Sequenzdiagramme für den inneren Lauf

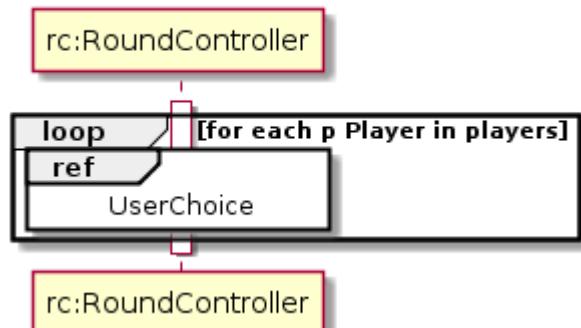
5.3.1 Round control initialization

Round control initialisation

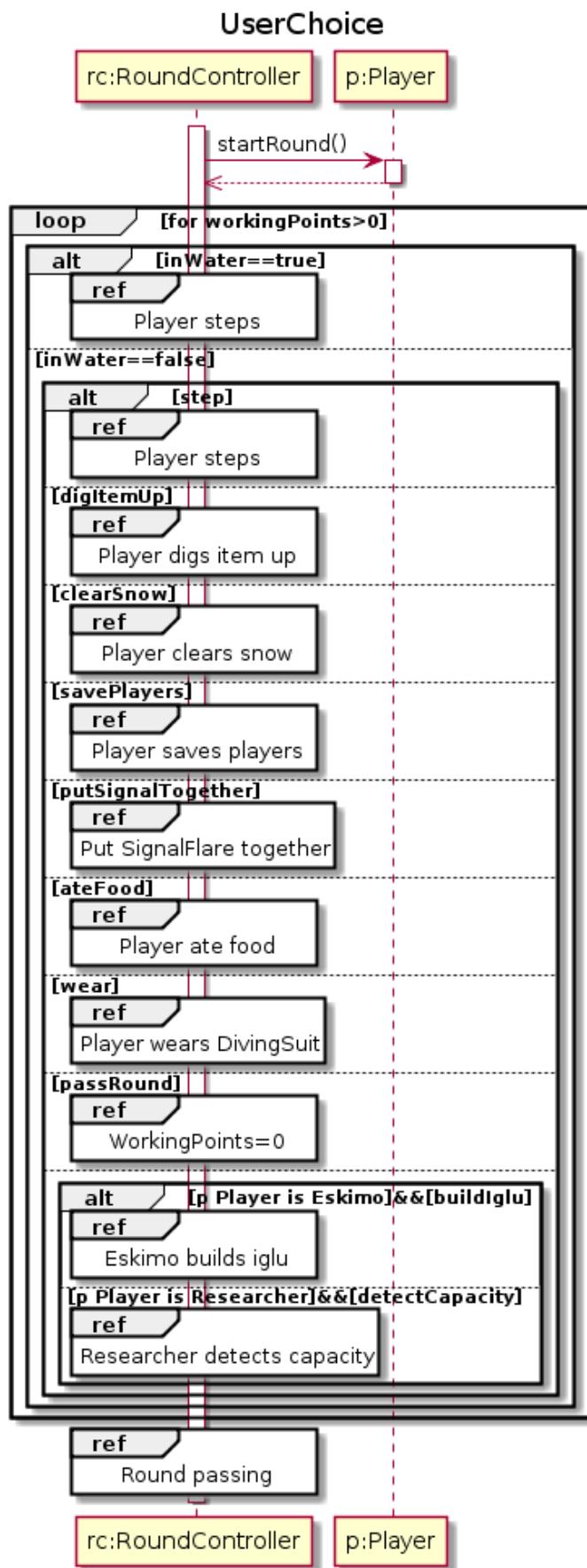


5.3.2 Round control

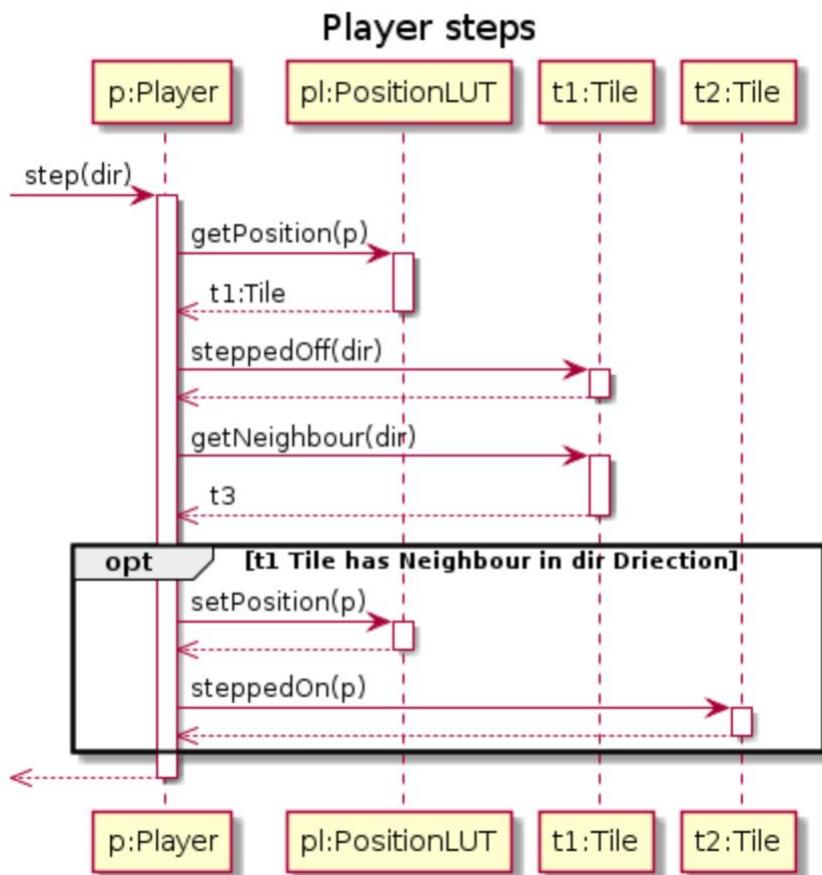
Round control



5.3.3 UserChoice

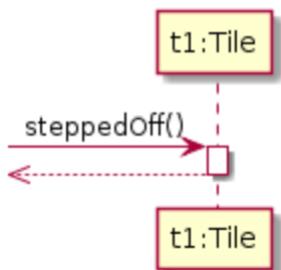


5.3.4 Player steps



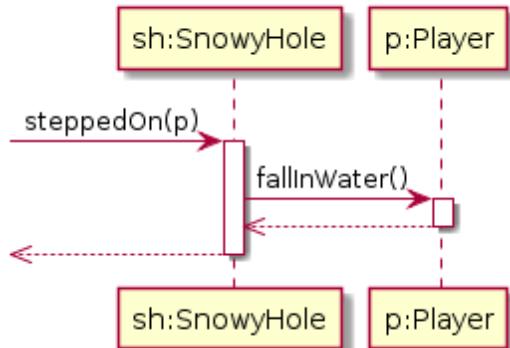
5.3.5 Tile steppedOff

Tile steppedOff



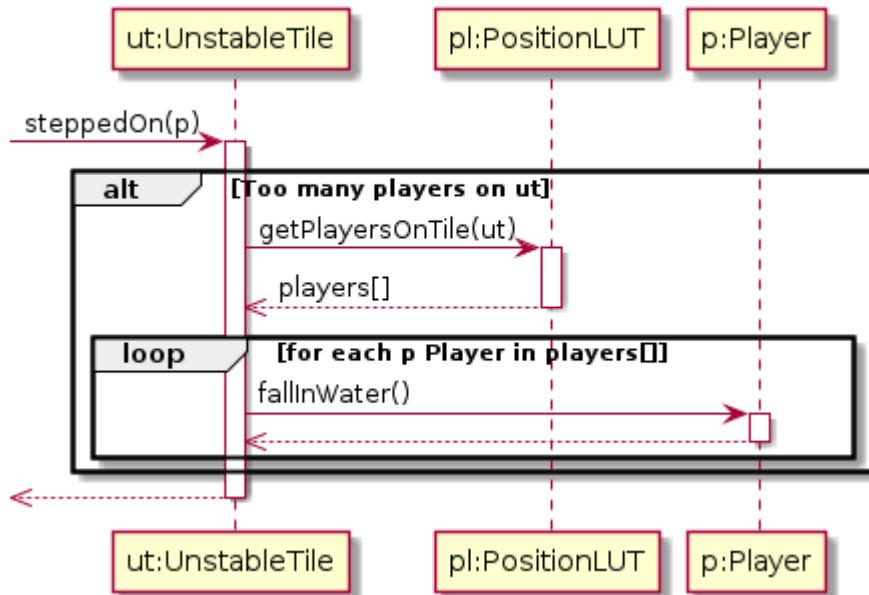
5.3.6 SnowyHole steppedOn

SnowyHole steppedOn()

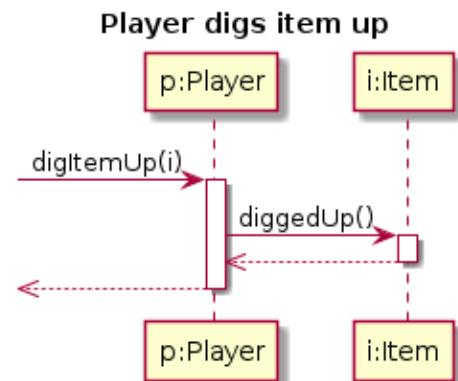


5.3.7 UnstableTile steppedOn

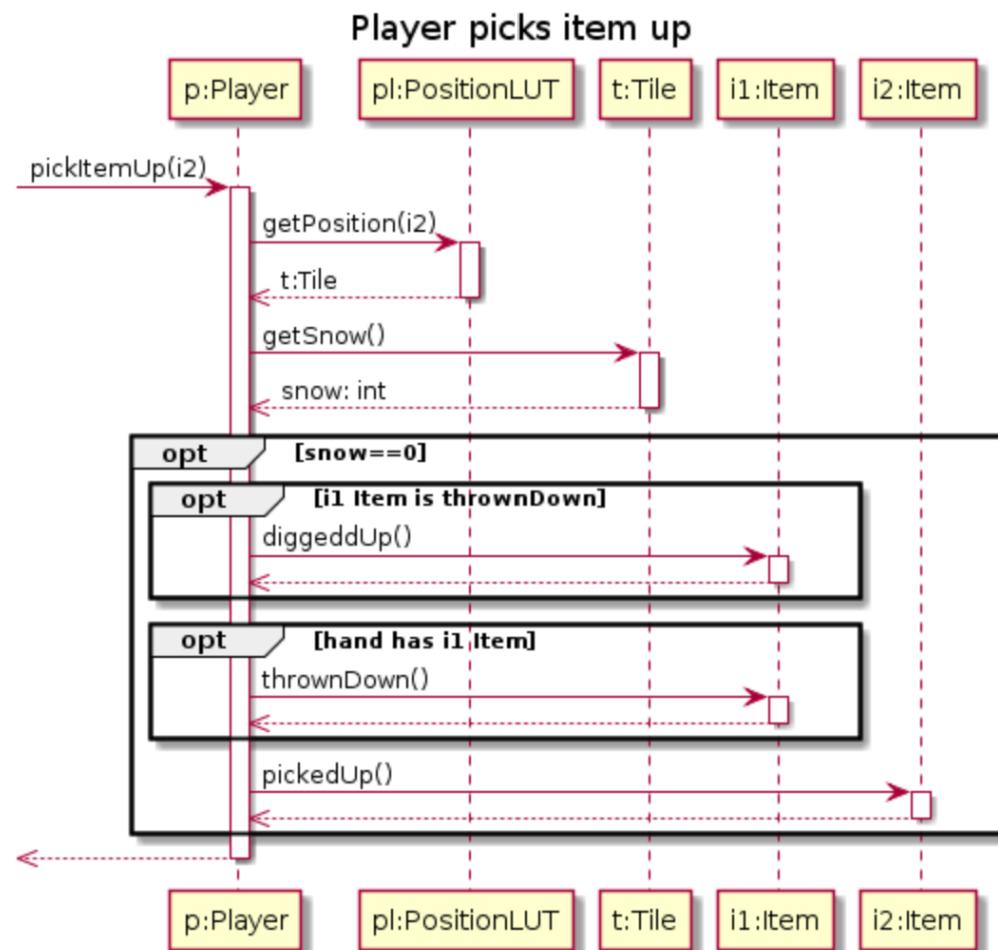
UnstableTile steppedOn()



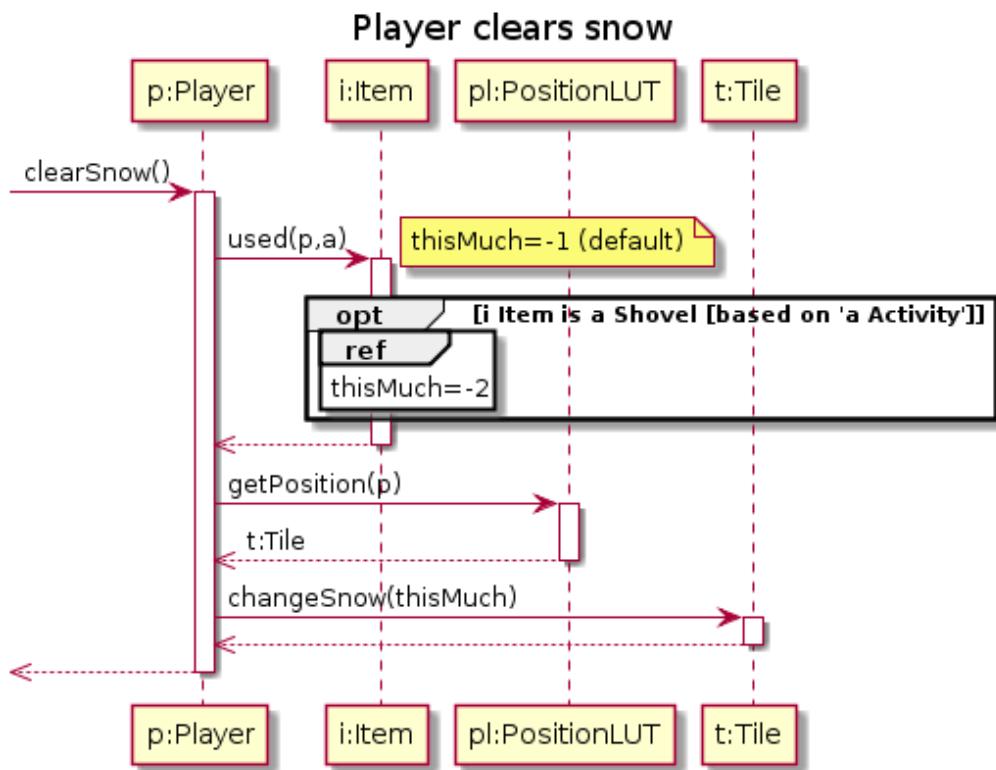
5.3.8 Player digs item up



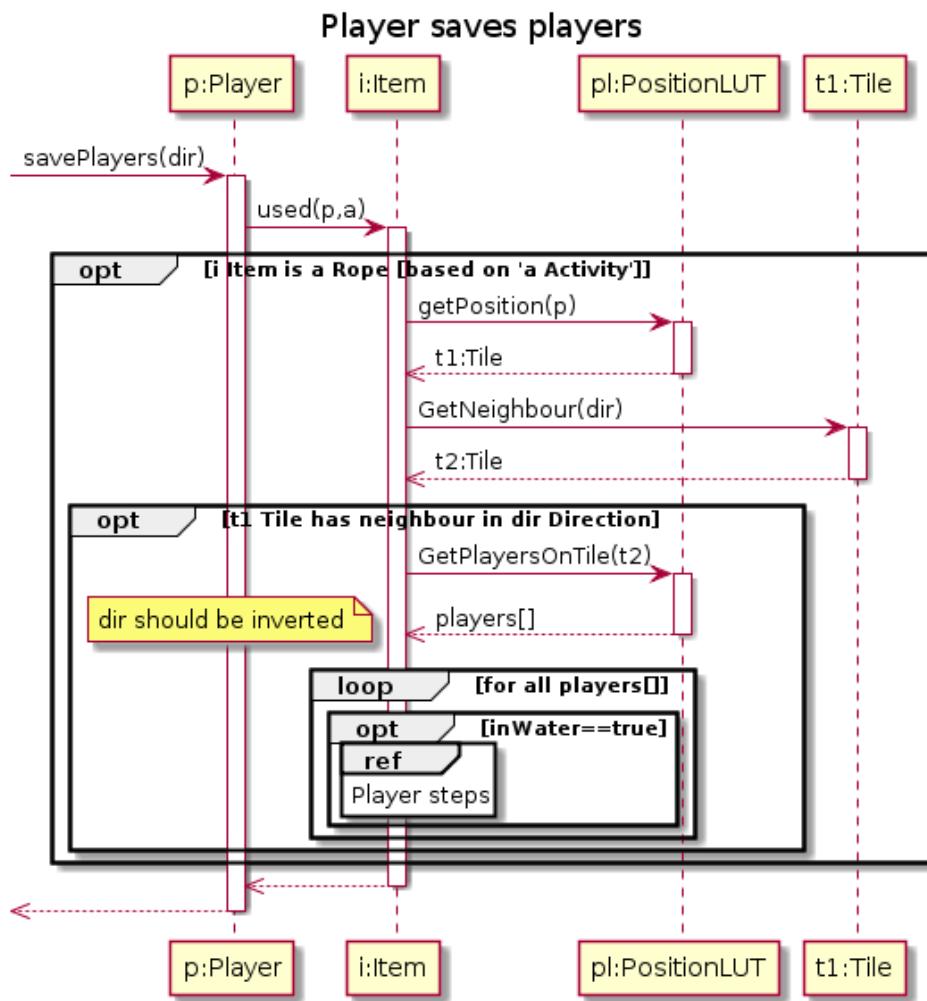
5.3.9 Player picks item up



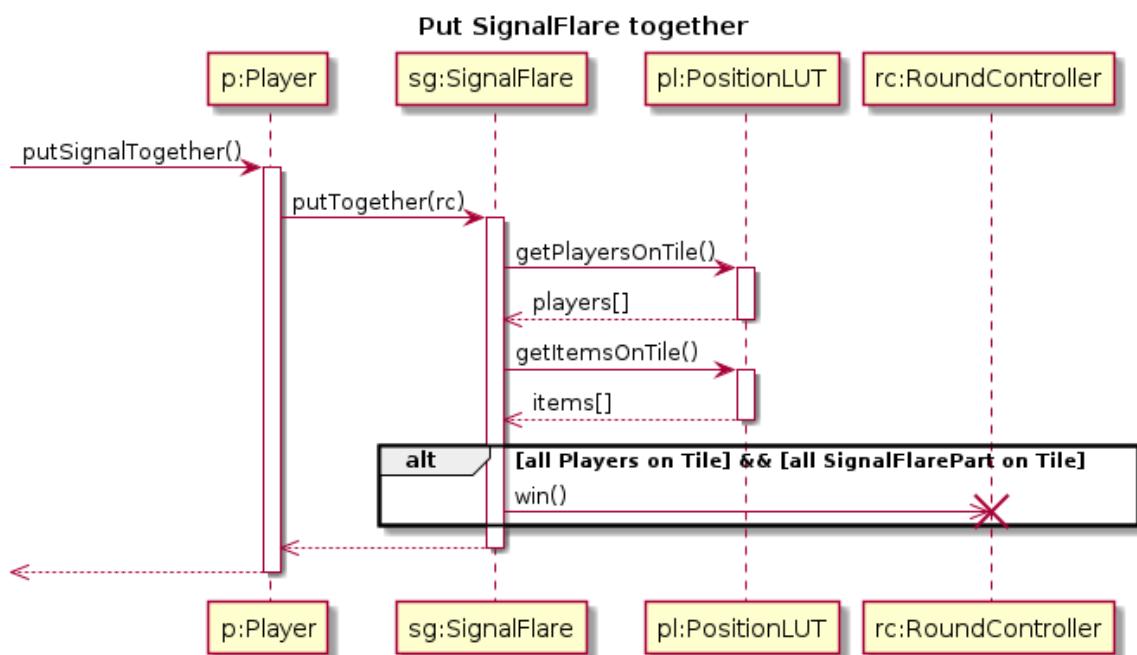
5.3.10 Player clears snow



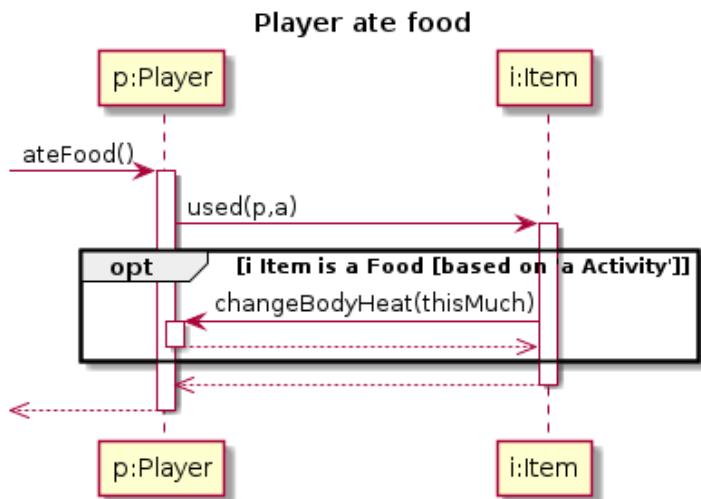
5.3.11 Player saves players



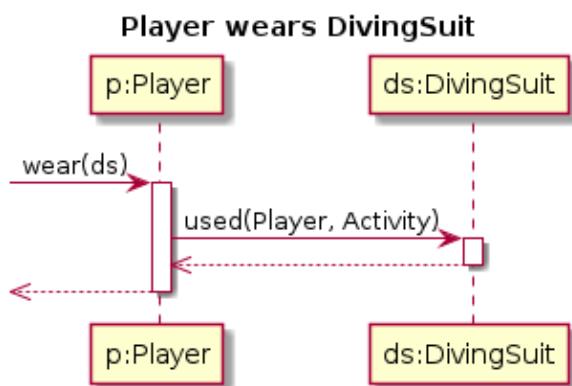
5.3.12 Put SignalFlare together



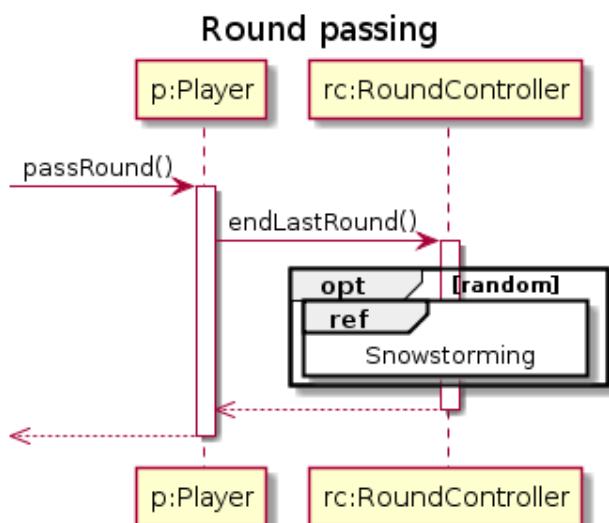
5.3.13 Player ate food



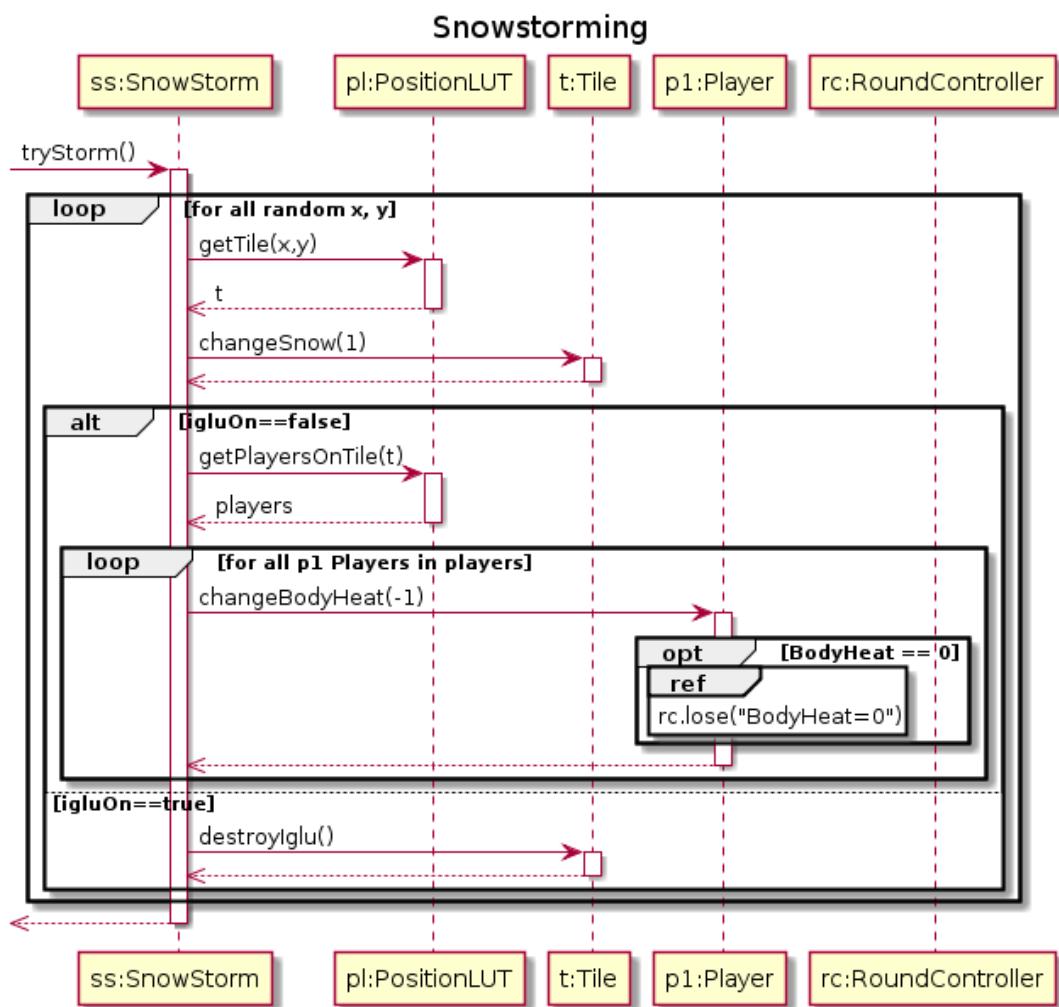
5.3.14 Player wears DivingSuit



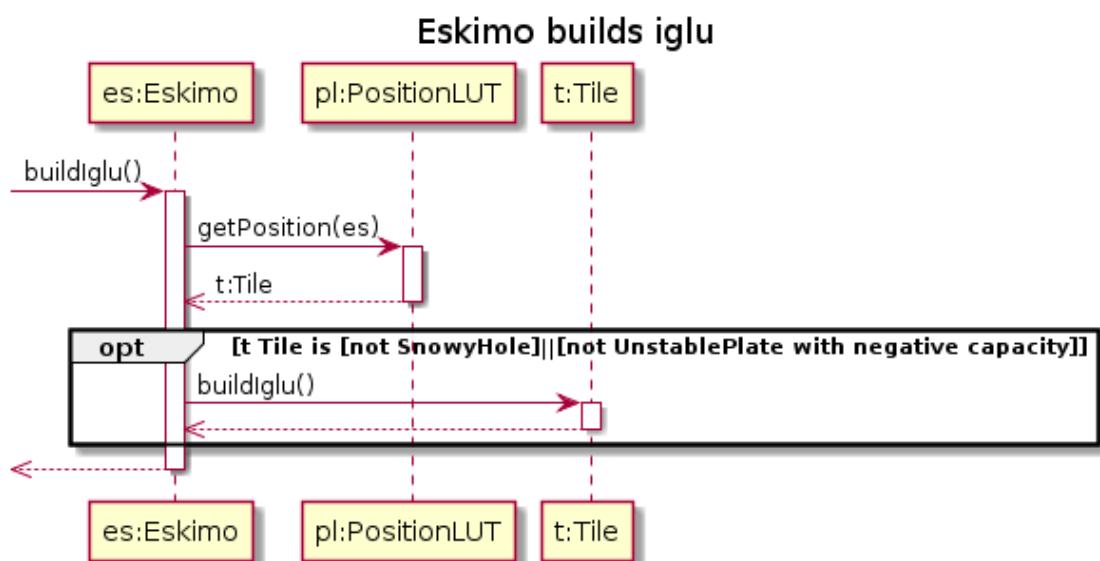
5.3.15 Round passing



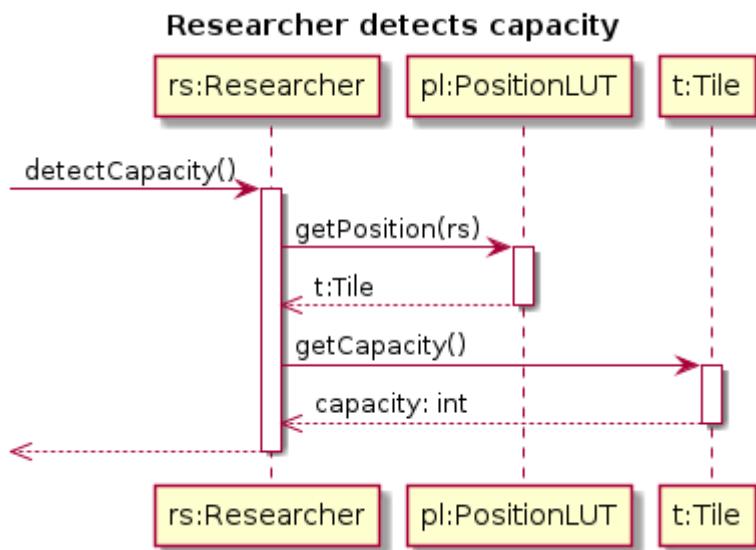
5.3.16 Snowstorming



5.3.17 Eskimo builds iglu



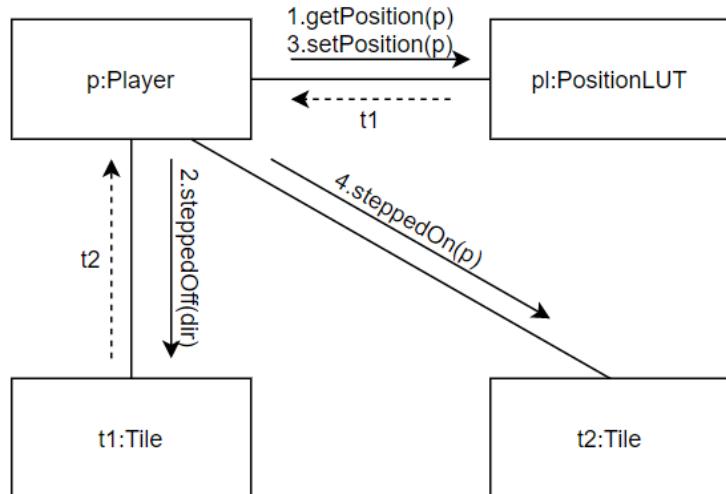
5.3.18 Researcher detects capacity



5.4 Kommunikationsdiagramme

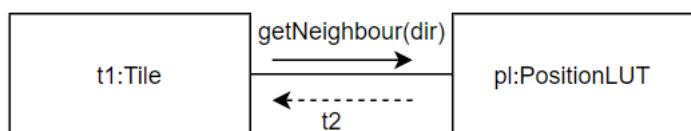
5.4.1 Player Steps

Player Steps



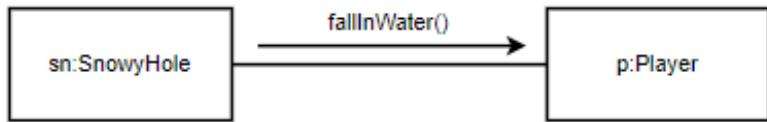
5.4.2 Tile steppedOff

Tile steppedOff



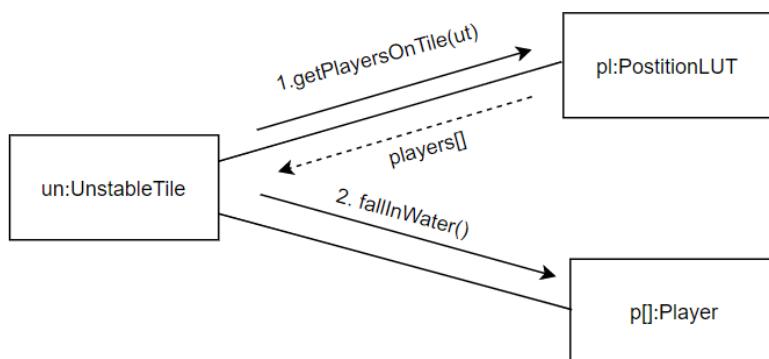
5.4.3 SnowyHole steppedOn

SnowyHole steppedOn



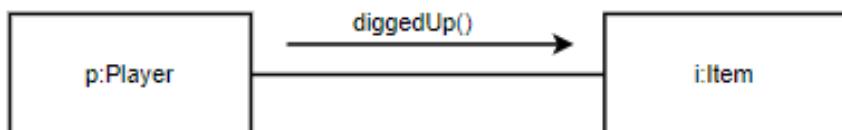
5.4.4 UnstableTile steppedOn()

UnstableTile steppedOn()



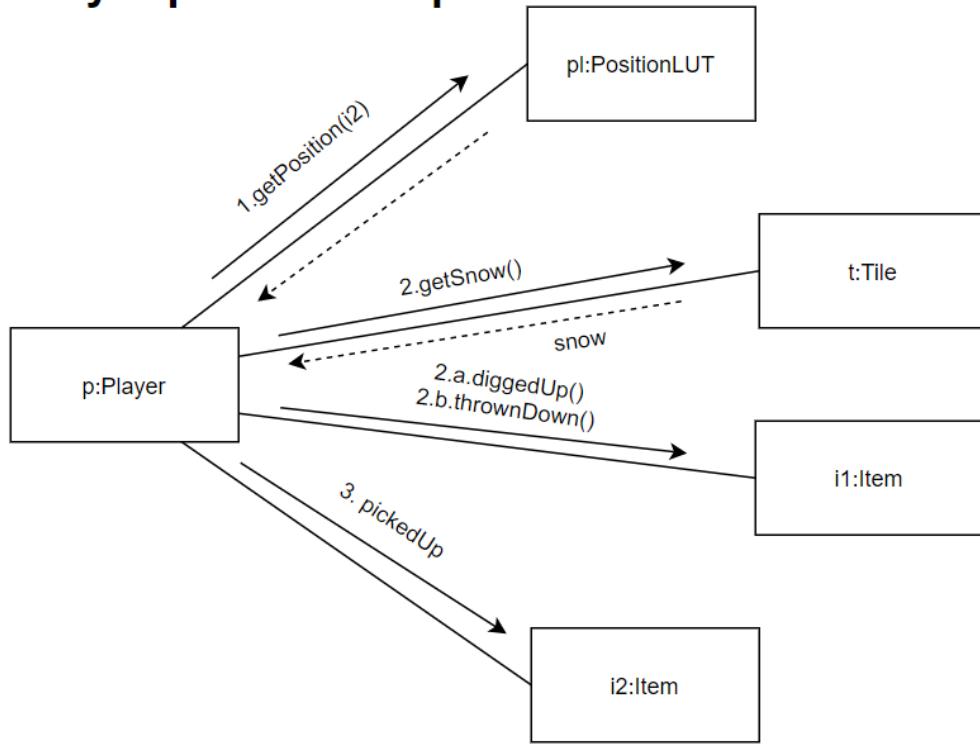
5.4.5 Player digs item up

Player digs item up



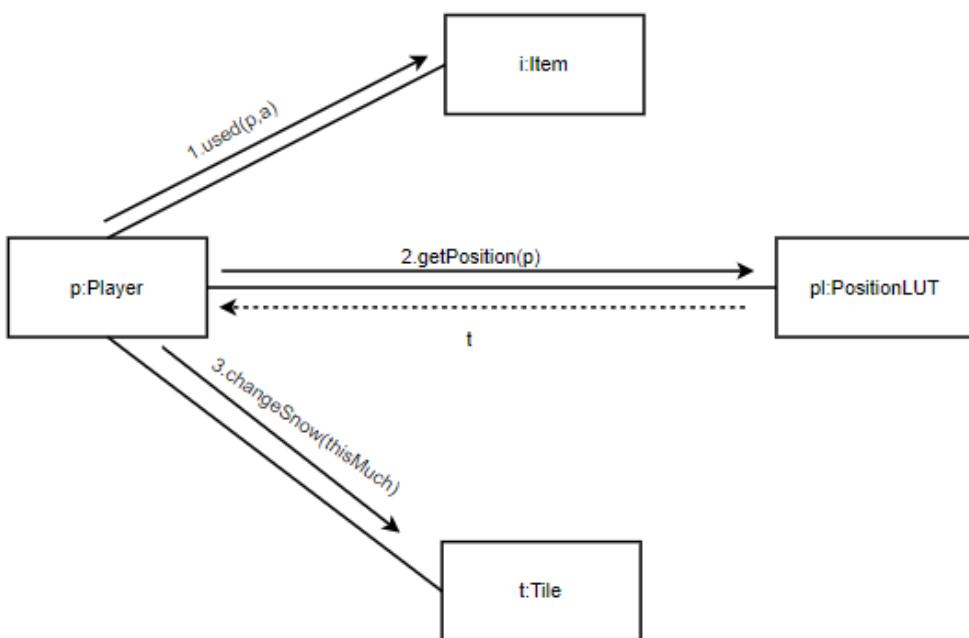
5.4.6 Player picks item up

Player picks item up



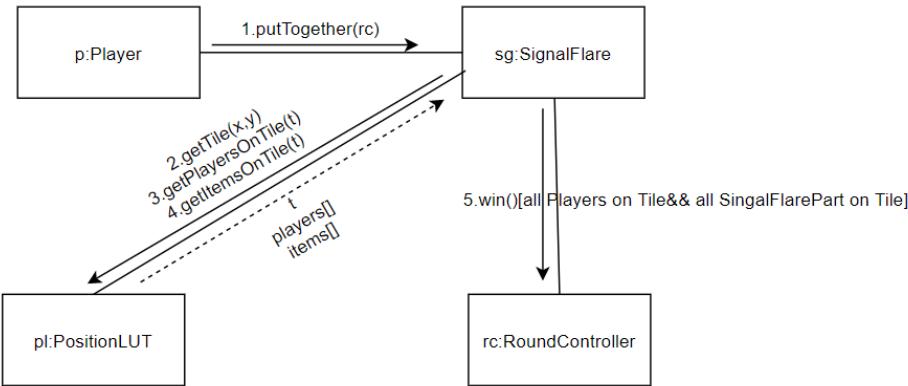
5.4.7 Player clears snow

Player clears snow



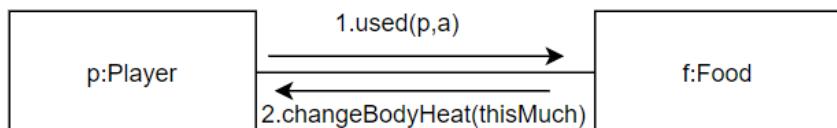
5.4.8 Put SignalFlare together

Put SignalFlare together



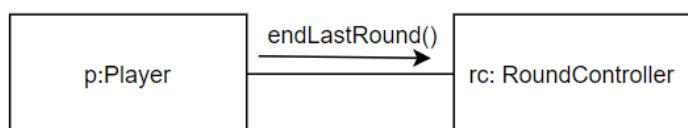
5.4.9 Player ate food

Player ate food



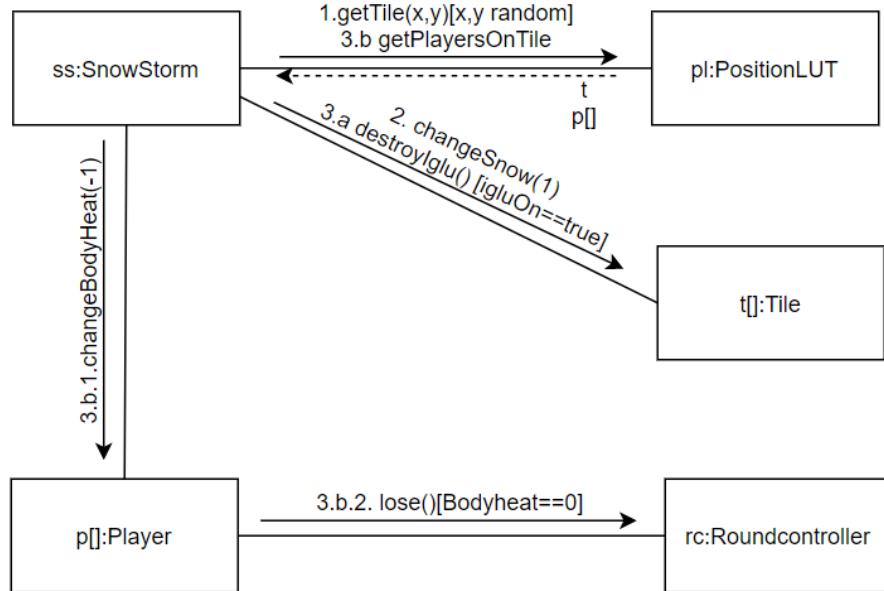
5.4.10 RoundPassing

Round passing

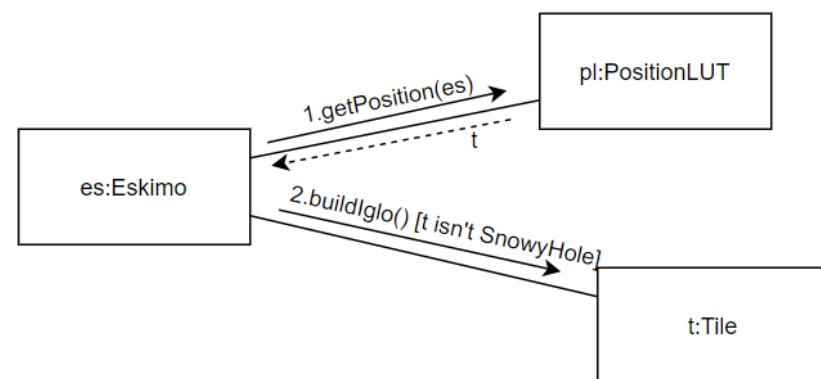


5.4.11 SnowStorming

Snowstorming



5.4.12 Eskimo builds iglu



6. Skeleton Eingabe

DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia
Hedrich Ádám
Pintér Balázs
Fucskár Patrícia
Tassi Timián

SOSK6A
H9HFFV
ZGY18G
XKYAOO
MYY53U

adamzsolfi.mail@gmail.com
hedrichadam09@gmail.com
pinterbalazs21@gmail.com
fucskar.patricia@gmail.com
timian.tassi@gmail.com

31. März 2020

6. Skeleton Eingabe

6.1 Übersetzung und Ausführung Anleitung

6.1.1 DateiListe

Dateiname	Grösse	Erstelldatum	Inhalt
GlobalControllers			
PositionLUT.java	7 kb	2020.03.31.	
RoundController.java	3 kb	2020.03.31	
ItemClasses			
Activity.java	1 kb	2020.03.31.	
DivingSuit.java	1 kb	2020.03.31	
Food.java	1 kb	2020.03.31.	
Item.java	1 kb	2020.03.31	
ItemState.java	1 kb	2020.03.31.	
Rope.java	2 kb	2020.03.31	
Shovel.java	1 kb	2020.03.31.	
SignalFlare.java	1 kb	2020.03.31	
SignalFlarePart.java	1 kb	2020.03.31.	
Main			
SkeletonMain.java	13 kb	2020.03.31.	
PlayerClasses			
Eskimo.java	1 kb		
IControllable.java	1 kb	2020.03.31	
Player.java	7 kb	2020.03.31.	
PlayerContainer.java	2 kb	2020.03.31	
Researcher.java	1 kb	2020.03.31.	
SnowStorm			
SnowStorm.java	2 kb	2020.03.31	
TileClasses			
Direction.java	1 kb	2020.03.31	
SnowyHole.java	1 kb	2020.03.31	
StableTile.java	1 kb	2020.03.31	
Tile.java	4 kb	2020.03.31	
UnstableTile.java	1 kb	2020.03.31	

Hinweis!

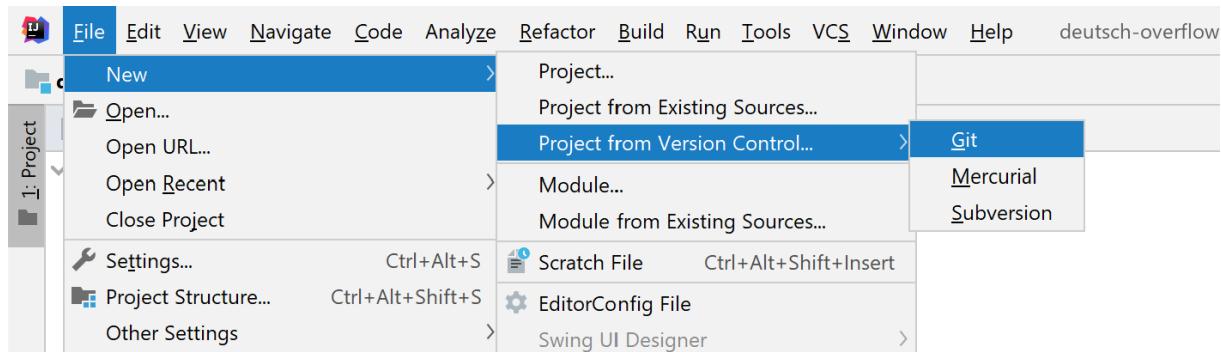
Die Dateien enthalten alle Methoden und zur Sequenzen benötigten Befehle.

6.1.2 – 6.1.3 Übersetzung & Ausführung

Die Übersetzung wird mithilfe von Github durch IntelliJ IDEA durchgeführt:

1. Öffnen Sie die Anwendung: [IntelliJ IDEA](#)

2. Klicken Sie auf [File>New>Project with Version Control>Git](#)



1. Abbildung: Öffnen des Projektes

3. Kopieren Sie der folgende Link ins Fenster:

<https://github.com/AdamZsofi/deutsch-overflow.git>

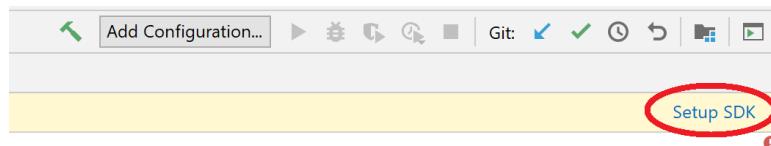
4. Klicken Sie auf „Clone“



2. Abbildung: Öffnen des Projektes

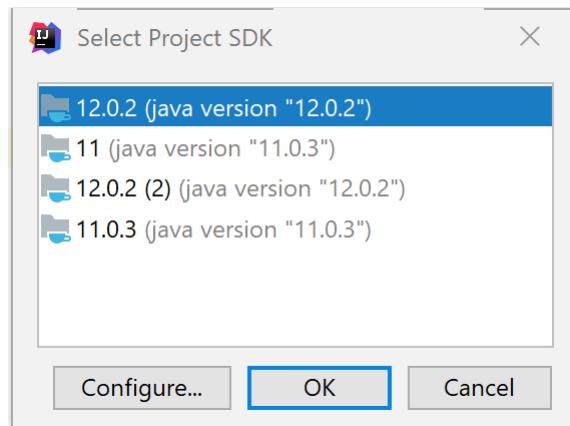
5. Navigieren Sie auf der linken Seite des Fensters im Baumstruktur [deutsch-overflow>src>SkeletonMain](#)

6. Wenn die Java SDK nicht eingestellt ist, dann klicken Sie an [Setup SDK](#)



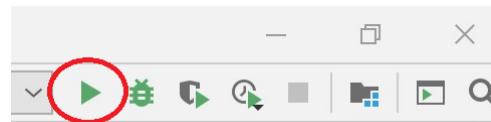
3. Abbildung

7. Stellen Sie die neueste Java Umgebung ein, dann klicken Sie auf „OK“



4. Abbildung

8. Klicken Sie auf „Ausführen“(Play)



5. Abbildung

Hinweis!

Später können Sie das Programm (.exe) auch mit Java Runtime Environment ausführen
(von der „bin“ Ordner des generierten Projektes)

6.4 Bewertung

Tag neve	Tag neptun	Munka százalékban
Ádám Zsófia	SOSK6A	20%
Hedrich Ádám	H9HFFV	20%
Pintér Balázs	ZGY18G	20%
Fucskár Patrícia	XKYAAO	20%
Tassi Timián	MYY53U	20%

How to import project in Eclipse from github

File>Import

Git>Projects from Git:Next

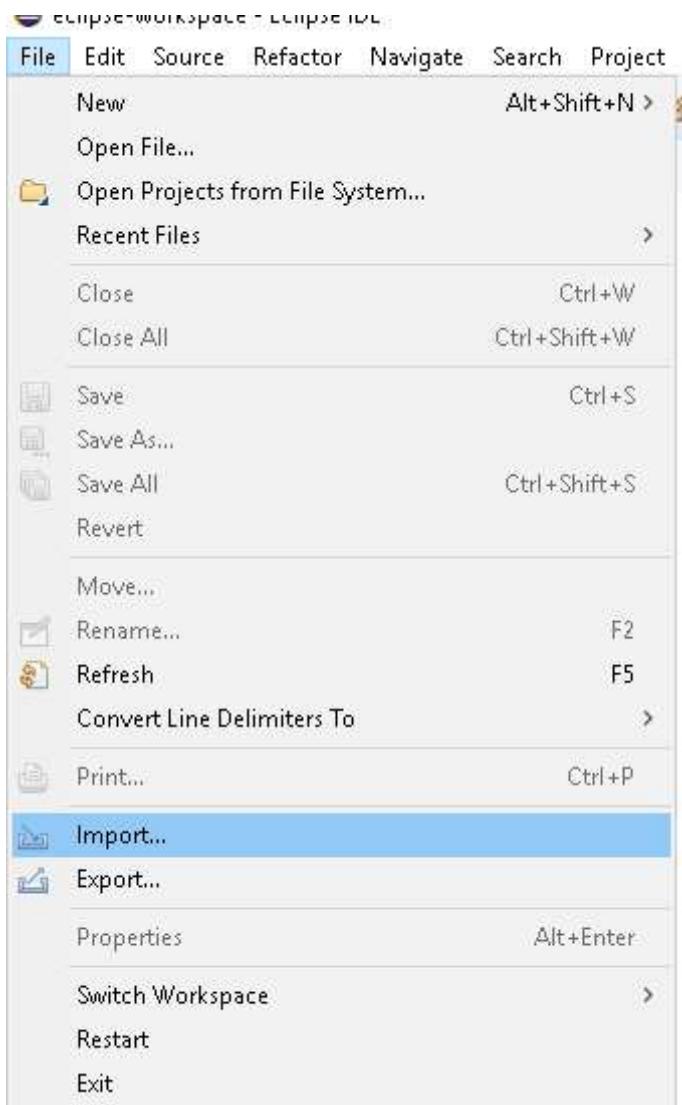
Clone URI:Next

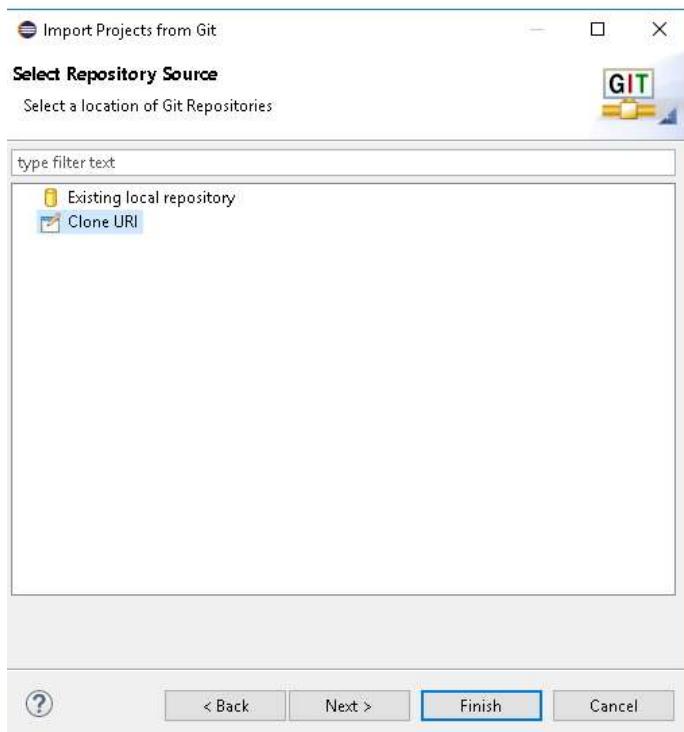
URI : <https://github.com/AdamZsofi/deutsch-overflow.git>: Next

Select Master branch: Next

Choose Destination Directory : Next

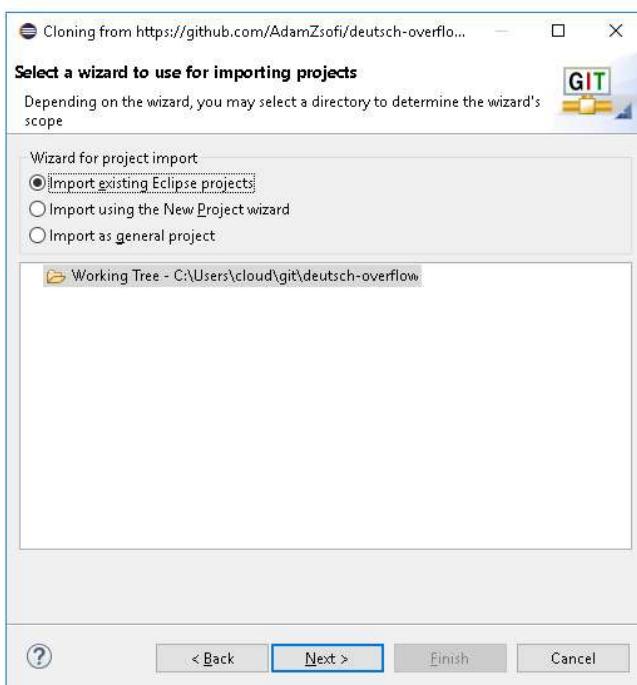
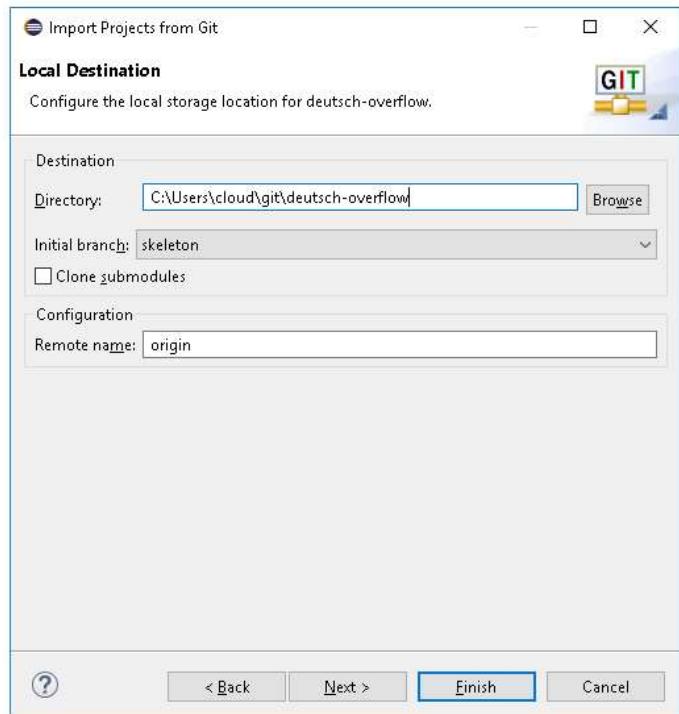
Import existing Eclipse project: Finish





The main dialog box is titled "Import Projects from Git" and "Source Git Repository". It prompts the user to "Enter the location of the source repository". The "Location" section contains fields for "URI" (set to <https://github.com/AdamZsofi/deutsch-overflow.git>), "Host" (set to github.com), and "Repository path" (set to /AdamZsofi/deutsch-overflow.git). The "Connection" section includes "Protocol" (set to https) and "Port" (empty). The "Authentication" section includes "User" (empty) and "Password" (empty), with a checked "Store in Secure Store" checkbox. At the bottom are buttons for "?", "< Back", "Next >", "Finish" (highlighted in blue), and "Cancel".

A secondary window titled "Branch Selection" is overlaid on the main dialog. It shows a list of branches from the remote repository: "master" and "deutsch". It includes "Select All" and "Deselect All" buttons at the bottom.



7. Konzeption von Prototyp

DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia
Hedrich Ádám
Pintér Balázs
Fucskár Patrícia
Tassi Timián

SOSK6A
H9HFFV
ZGY18G
XKYAOO
MYY53U

adamzsofi.mail@gmail.com
hedrichadam09@gmail.com
pinterbalazs21@gmail.com
fucskar.patricia@gmail.com
timian.tassi@gmail.com

8. April 2020

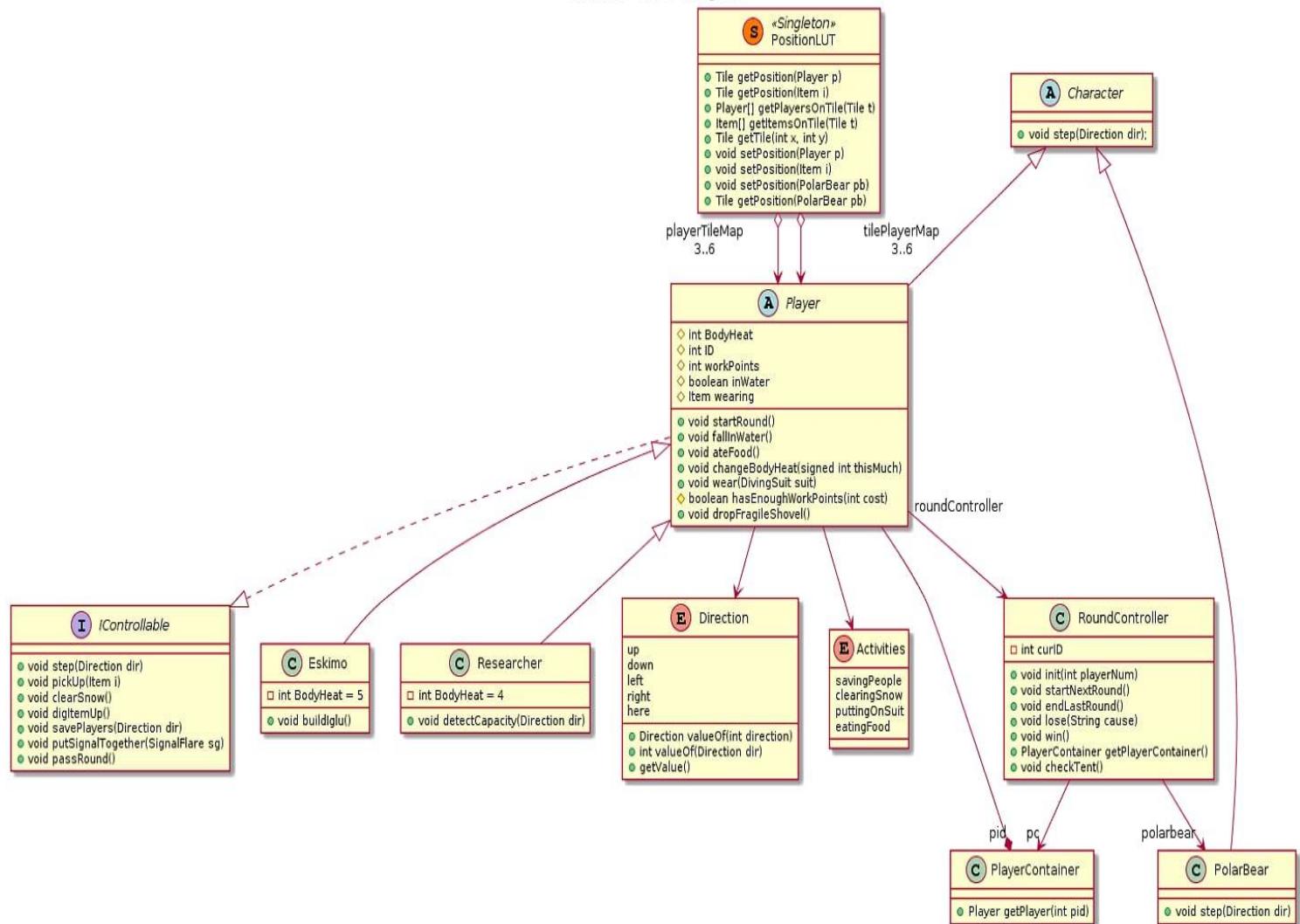
7. Konzeption von Prototyp

7.0 Wirkung der Änderung auf das Modell

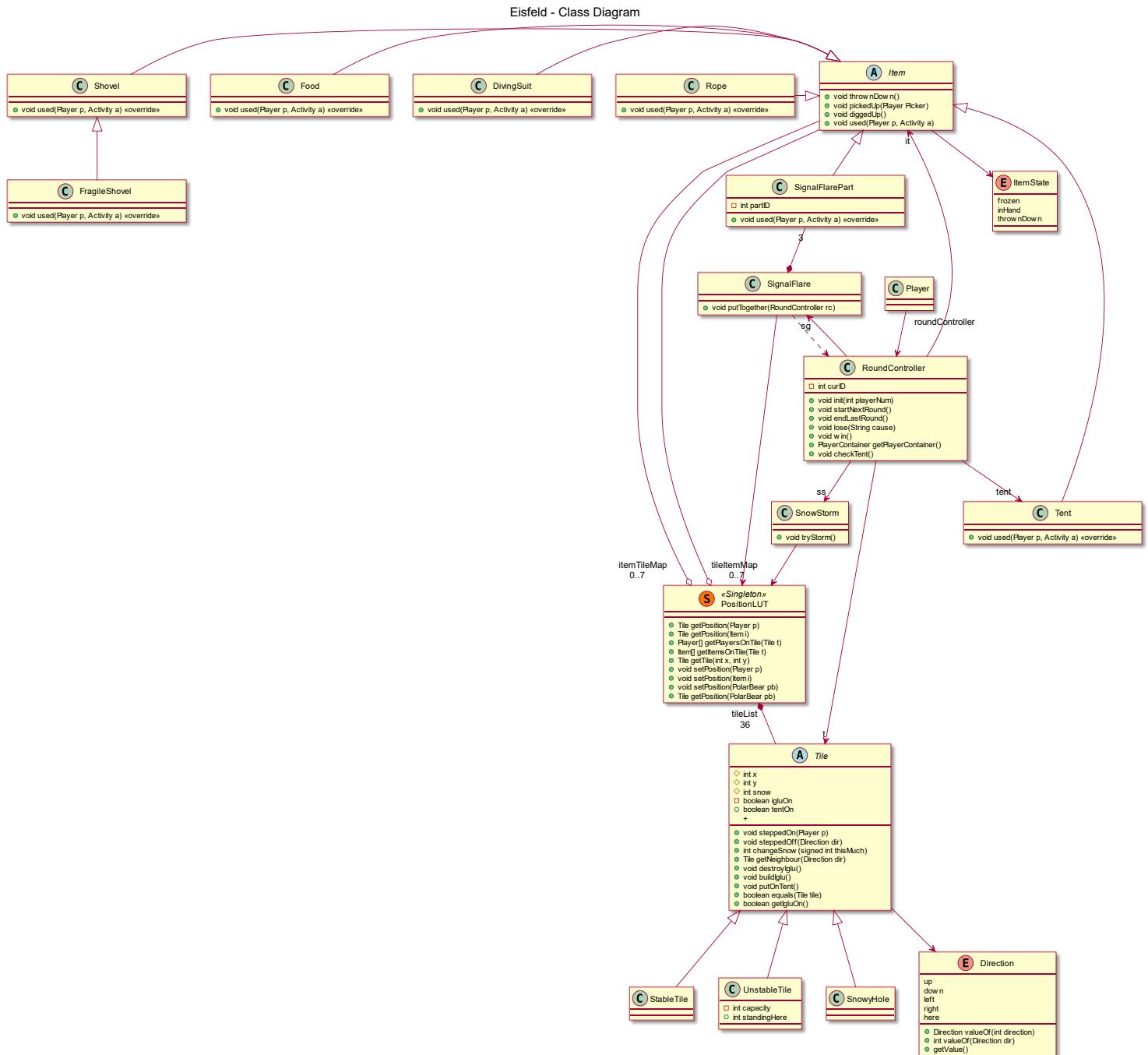
7.0.1 Verändertes Klassendiagramm

- Part1:

Eisfeld - Class Diagram

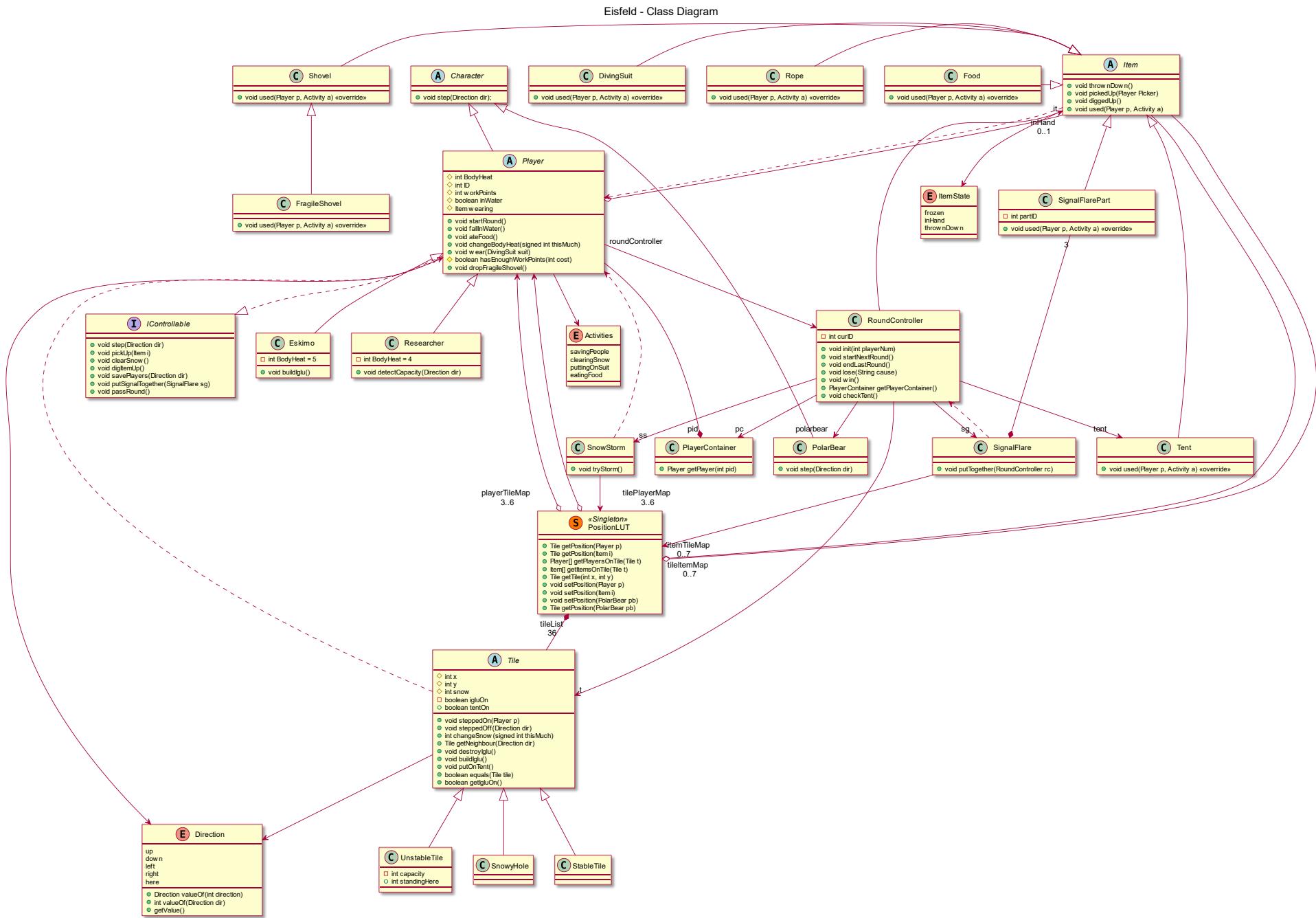


- Part2:**



7. Konzeption von Prototyp

DeutschOverflow



7.0.2 Neue oder veränderte Methoden

Neue Klassen:

1. FragileShovel:

- **Verantwortung:** Diese Klasse verwirklicht die zerbrechliche Schaufel. Es vererbt die Klasse Item.
- **Schnittstellen:** -
- **Attributen:** -
- **Methoden:**
 - **+void used(Player p, Activity a):** Diese Methode schreibt die “used” Methode vom Gegenstand über.

2. Character:

- **Verantwortung:** Diese Klasse ist eine Superklasse. Es ist eigentlich ein Container für alle Charakter, die in dem Spiel vorkommen kann. Es gibt eine gemeinsame Eigenschaft zwischen allen Charakter: der Schritt.
- **Schnittstellen:** -
- **Attributen:** -
- **Methoden:**
 - **+void step(Direction dir):** Diese Methode wird gerufen, wenn ein Charakter treten möchte. Wir müssen die Richtung eingeben.

3. PolarBear:

- **Verantwortung:** Diese Klasse verwirklicht der Eisbär. Es vererbt die Superklasse Character.
- **Schnittstellen:** -
- **Attributen:** -
- **Methoden:**
 - **+void step(Direction dir):** Die “Step” Methode von Character wird überschreibt.

4. Tent:

- **Verantwortung:** Diese Klasse ist verwirklicht das Zelt. Wir haben es als Gegenstand behandelt, aber diesem Gegenstand hat eine Zeitgrenze.
- **Schnittstellen:** -
- **Attributen:** -
- **Methoden:**
 - **+void used(Player p, Activity a):** Diese Methode schreibt die “used” Methode vom Gegenstand über.

Veränderte Methode oder Attribute:

- Player:

Methode:

- **+dropFragileShovel():** Diese Methode wirft die zerbrechliche Schaufel runter. Diese Methode dient, dass das “inHand” Attribut von außen nicht modifizierbar sein werden können.

- RoundController:

Attribute:

- **PolarBear polarbear:** Der einzige Eisbär wird hier gespeichert.
- **Tent tent:** Das einzige Zelt wird hier gespeichert.

Methode:

- **+void dropFragileShovel():** Diese Methode wirft die zerbrechliche Schaufel runter. Diese Methode dient, dass das “inHand” Attribut von außen nicht modifizierbar sein werden können.

- **PositionLUT:**

Methode:

- **+void setPosition(PolarBear polarbear):** Wir können die Position des Eisbär einstellen.
- **+Tile getPosition(PolarBear polarbear):** Wir können die Position des Eisbärs abfragen.

- **Direction(Enumeration):**

Attribute:

- **Here:** Die Position des Charakters verändert sich nicht.

Methode:

- **+Direction valueOf(int direction):** Es wandelt die ganze Zahl zur Richtung um.
- **+int valueOf(Direction dir):** Es wandelt die Richtung zur ganzen Zahl.
- **+getValue():** Es liefert den Wert zurück.

- **Tile:**

Attribute:

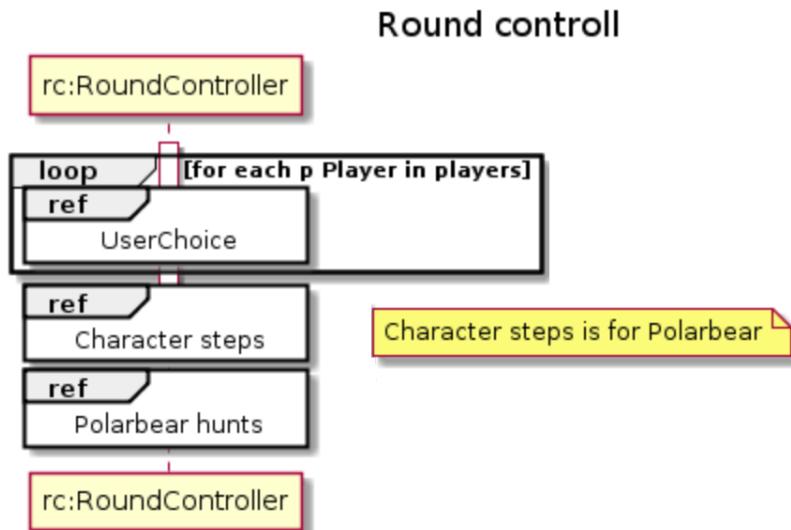
- **+boolean tentOn:** Es zeigt, ob ein Zelt sich auf dieser Platte befindet. True: Ja, False: Nein.

Methode:

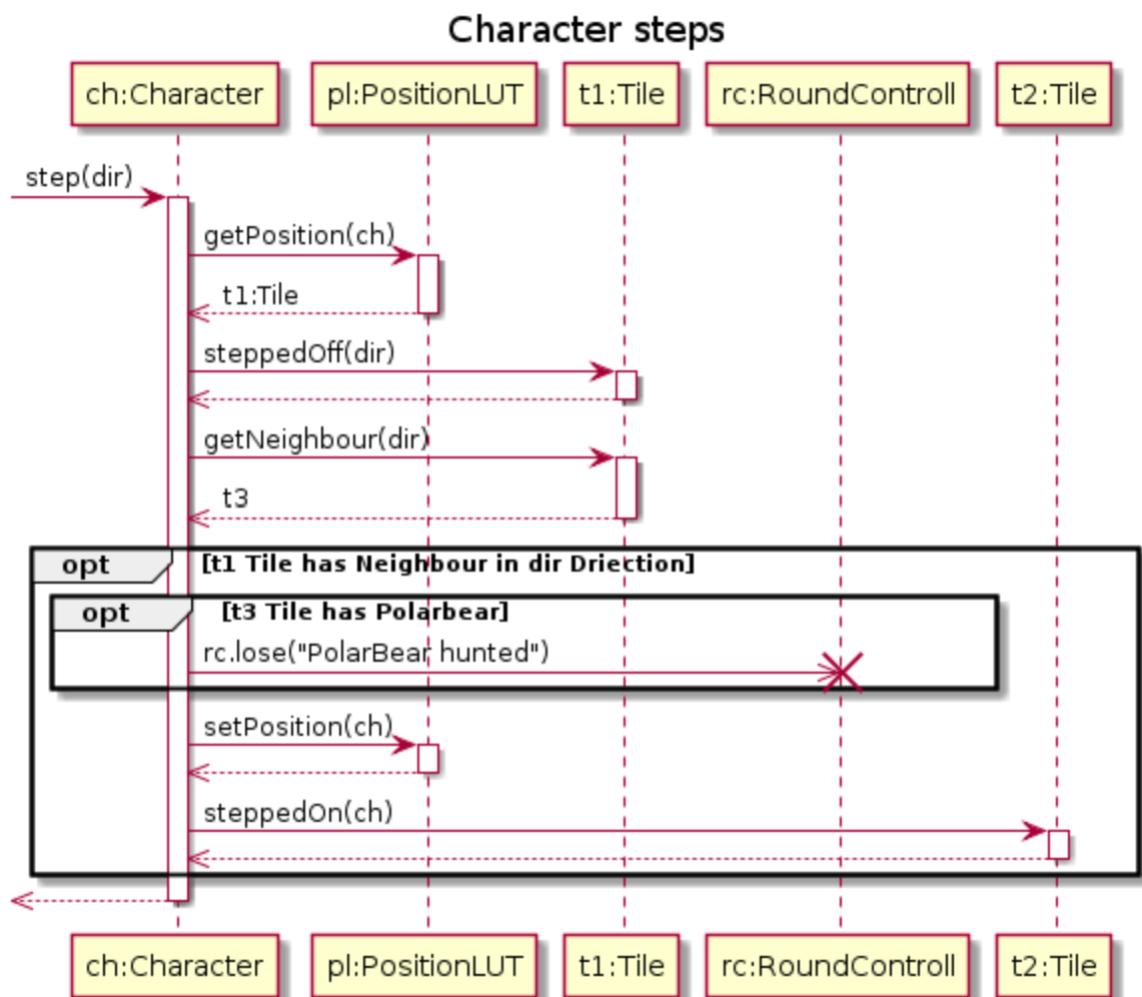
- **+boolean equals(Tile tile):** Es zeigt, ob die Position von zwei Platte gleich sind.
- **+boolean getIgluOn():** Es liefert den Wert von “igluOn” Attribut zurück.
- **+void putOnTent():** Diese Methode baut ein Zelt auf diese Platte auf.

7.0.3 Sequenzdiagramme

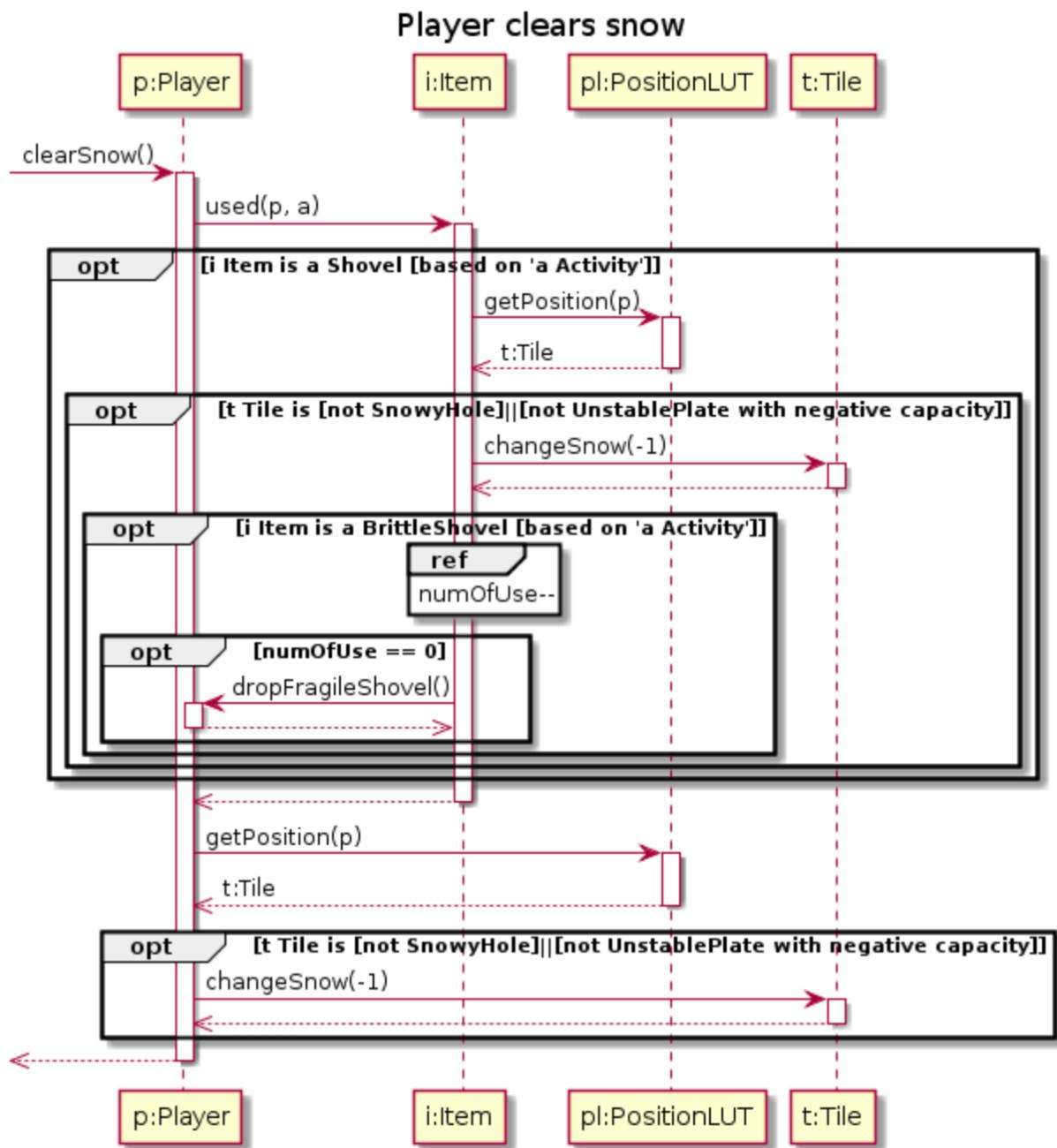
7.0.3.1 Round controll



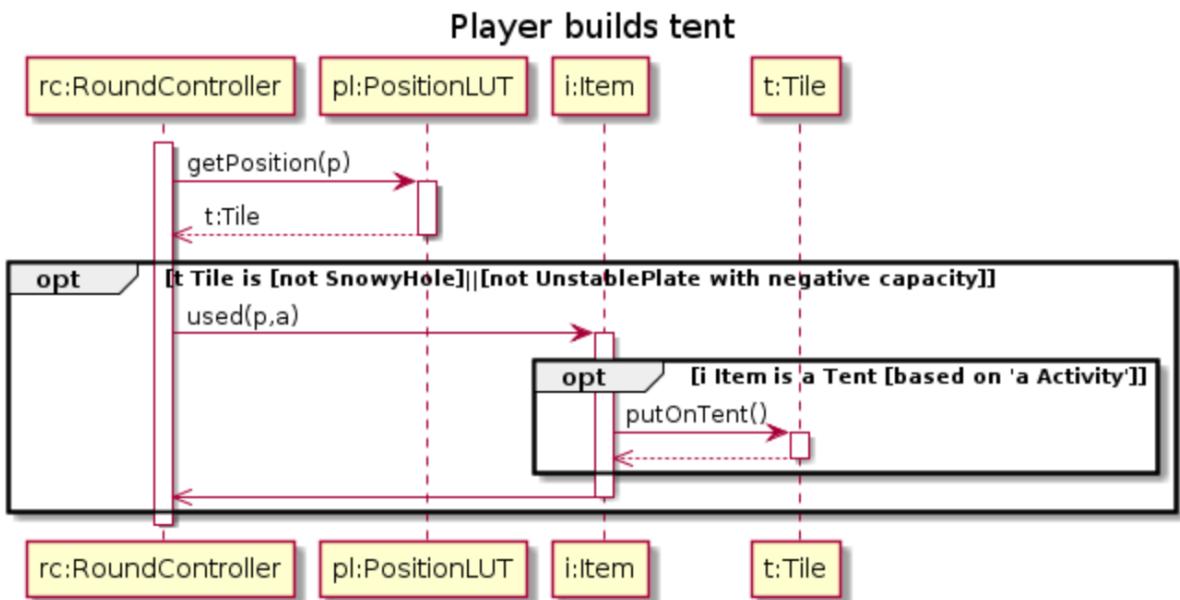
7.0.3.2 Character steps



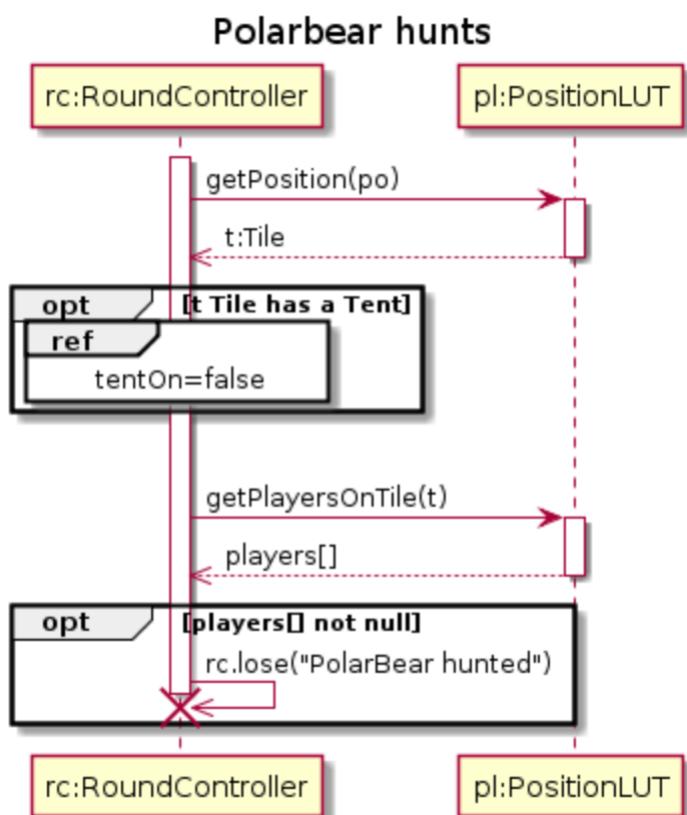
7.0.3.3 Player clears snow



7.0.3.4 Player builds tent



7.0.4 Polarbear hunts



7.1 Definition von der Schnittstelle des Prototyps

7.1.1 Allgemeine Beschreibung der Schnittstelle

7.1.2 Eingangssprache

Die Befehle der Eingangssprache sind hier gegeben. Falls es mögliche Optionen gibt, dann fragt das Spiel nach diese! (z.B.: PrintTile <enter> "x?" 2 <enter> "y?" 3 <enter> ...) (Zwischen die Befehle/Option gibt es immer ein new line)

Andere Bemerkung: die Printer Befehle schreiben solche Informationen auch aus, die die Spieler werden normalerweise nicht sehen. Es ist so geplant, weil testen ist so durchschaubarer und einfacher.

Allgemeine und “Printer” Befehle:

StartGame

Beschreibung: Startet das Spiel. Zufallsartigkeit kann mit dem Option “deterministic” ausgeschaltet werden.

Optionen: deterministic

PrintCharacterMap

Beschreibung: Schreibt die Landkarte der Spieler aus. Über das präzise Format, siehe unten.

Optionen: -

PrintItemMap

Beschreibung: Schreibt die Landkarte der Gegenstände aus. Über das präzise Format, siehe unten.

Optionen: -

PrintHeimMap

Beschreibung: Schreibt die Landkarte der Iglus und Zelter aus. Über das präzise Format, siehe unten.

Optionen: -

PrintSnowTileMap

Beschreibung: Schreibt die Landkarte über Schneeeinheiten aus. Über das präzise Format, siehe unten.

Optionen: -

PrintTile

Beschreibung: Schreibt die Eigenschaften (Schnee, Kapazität und wie viele Spieler gibt darauf) der gegebenen Tile aus.

Optionen: Koordinaten der Tile (x,y)

PrintItem

Beschreibung: Schreibt die Eigenschaften (Typ, Zustand) der gegebene Gegenstand aus.

Optionen: ID der Gegenstand

PrintPlayer

Beschreibung: Schreibt die Eigenschaften (inHand, wearing, Temperatur ...) der gegebenen Spieler aus.

Optionen: ID von Player

Spielerbefehle (“inGame”):

Während die Runde der Spieler können wir die obene Befehle (außer StartGame) auch benutzen, aber um das Spiel zu spielen benutzen wir diese Befehle.

Funktion	Befehl	Beschreibung	Optionen
Treten	Step	<i>Als Befehl:</i> Tritt das Spieler in die gegebene Richtung (falls möglich) <i>Als Option:</i> Bedeuten diese Tasten verschiedene Richtungen	Richtung (a/w/s/d)
Gegenstand - aufnehmen - ausgraben	PickUp DigItemUp	Eindeutig nach “Funktion”	-
Spezialfähigkeit	UseSkill	Benutzt das Spieler ihre spezielle Fähigkeit	Richtung (a/w/s/d), falls es ein Forscher ist (<i>Die Forscher können auch das Feld unten ihr analysieren – dazu müssen wir das Option leer lassen</i>)
Schnee räumen	ClearSnow	(als in Clear) Das Spieler Räumt 1 oder 2 Schnee	-
Retten	SavePlayers	(als in retten) Retten andere Spieler, falls möglich.	Richtung (a/w/s/d)
Zelt bilden	BuildTent	Falls das Spieler ein Zelt im Hand hat, bildet es.	-
Signalfackel aufbauen	PutSignalTogether		-
Spieler Runde beenden	PassRound		-

7.1.3 Ausgangssprache

Ausgang der “Printer” Befehle:

StartGame

Ausgang: Schreibt “Games started” aus und schreibt der Ausgang ein gestartetes Rund aus.

PrintCharacterMap

Ausgang: Schreibt die PlayerList von jedem Eisfeld aus (Jeder Teil als [E<PlayerID>, R<PlayerID>], wo R bedeutet Forscher und E bedeutet Eskimo und B bedeutet Eisbär)

Beispiel mit ein 3x3 Spielfeld:

[E1, R2][][]

[][R3][B]

[][][E4]

PrintItemMap

Ausgang: Ähnlich, wie beim PrintCharacterMap.

Wir schreiben der Abkürzung der englische Name aus (D, F, R, S, SFP, Z).

Die Gegenstände, die in der Hand eines Spielers sind, sind hier nicht ausgeschrieben.

Wir schreiben ihren Zustand auch aus. F bedeutet gefrieren und T bedeutet “auf den Boden”

Beispiel Feld: [F(T)] (Hier liegt ein Essen auf den Boden)

PrintHeimMap

Ausgang: Schreibt eine ähnliche Landkarte wie oben aus, aber mit T für Zelt und I für Iglo.

PrintSnowTileMap

Ausgang: Landkarte mit Anzahl der Schnee Einheiten für jeder Feld.

3x3 Bsp.:

[3][0][1]

[4][3][2]

[1][1][0]

PrintTile

Ausgang: Schreibt die (heimliche) Information über Feld (x,y) aus. (Kapazität, wie viel Spieler auf Feld)

Ausgang der Spielerbefehle:***Step***

Ausgang:

“Step not successful” /

“Step successful, stepped from (x,y) to (x,y)”

“Player falled in water”

PickUp

Ausgang:

“Nothing to pick up” /

“Picked up <Item> (optionally:, <Other Item> thrown down)”

(Und im Fall von Essen/Taucheranzug automatisch: “gegessen”, “aufgenommen”)

DigItemUp

Ausgang:

“No frozen item here” /

“Digged up <Item>.”

UseSkill

Ausgang:

“Can’t build igloo here” /

“Successfully built an igloo” /

“Capacity of Tile (x, y) is <Capacity>”

ClearSnow

Ausgang:

“Cleared <1 oder 2> Snow off from Tile.”

SavePlayers

Ausgang:

“I have no Rope” /

“No one to save here” /

“Saved players: <Player IDs>”

BuildTent**Ausgang:**

“Can’t build tent here” /
“I have no tent in my hand” /
“Successfully built a tent”

PutSignalTogether**Ausgang:**

“The signal flare is done!” /
“We don’t have all the 3 parts” /
“All players should stand here to do that”

PassRound**Ausgang:**

“Player <ID> passed”

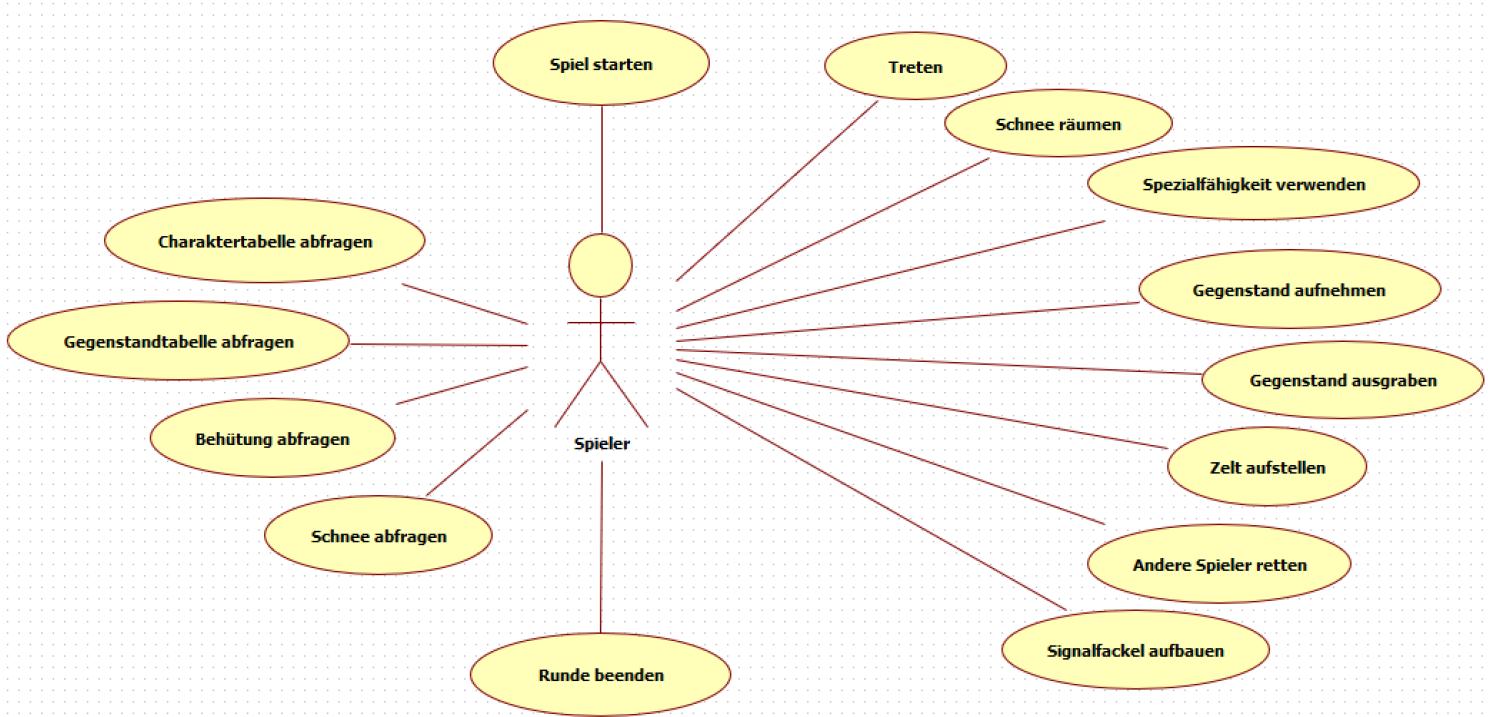
Andere, allgemeine Ausgänge:

Diese Ausgänge werden nicht immer als den Ausgang der verschiedenen Befehle ausgeschrieben, sondern wenn etwas sich verändert.

Wir geben solche aus, wenn...

- Die Runde eines Spielers endet (nach PassRound oder automatisch falls das Spieler kann nichts anderes tun)
- Die Betätigungen der nicht kontrollierbare Elemente – Sturm, Eisbär, Zelt aufhören
- Falls das Spiel gewonnen/verloren ist

7.2 Alle detaillierte use-case



Name von Use-case	Spiel starten
Kurze Beschreibung	Spieler startet das Spiel
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'StartGame ' (Spiel starten) 3. Kontroller speichert die Anfrage von Spieler

Name von Use-case	Charaktertabelle abfragen
Kurze Beschreibung	Spieler möchte Charaktertabelle ansehen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'PrintCharacterMap ' (Charaktertabelle anzeigen) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller zeigt die Charaktertabelle an

Name von Use-case	Gegenstandtabelle abfragen
Kurze Beschreibung	Spieler möchte Gegenstandtabelle ansehen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ‘PrintItemMap‘ (Gegenstandtabelle) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller zeigt die Gegenstandtabelle an

Name von Use-case	Behütung abfragen
Kurze Beschreibung	Spieler möchte die Behütung sehen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ‘PrintHeimMap‘ (Behütung) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller zeigt die Behütung an

Name von Use-case	Schnee abfragen
Kurze Beschreibung	Spieler möchte Schnee von einzelnen Tiles sehen
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ‘PrintSnowTileMap‘ (Schnee anzeigen) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller zeigt den Schnee an

Name von Use-case	Information über einem bestimmten Tile abfragen
Kurze Beschreibung	Spieler fragt die Informationen von bestimmten Tile
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ‘PrintTile‘ (Info über einem Tile) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller fragt welche Tile angezeigt werden soll 5. Spieler gibt Tilenummer an 6. Kontroller zeigt die Infos über Tile
Alternative Tätigkeit	<ol style="list-style-type: none"> 1.A.5. Der Spieler hat nicht eine richtige Tilenummer gegeben 1.A.6. Kontroller warnt über Misserfolg der Angabe

Name von Use-case	Treten
Kurze Beschreibung	Spieler tritt an eine andere Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Benutzer gibt 'Step' an 3. Kontroller speichert die Angabe 4. Spieler gibt eine Richtung ein, mithilfe von 'a/w/s/d' Tasten (links/oben/rechts/links) realisiert 5. Kontroller speichert die angegebene Richtung
Alternative Tätigkeit	1.A.4. Es gibt keine Platte in der gegebenen Richtung von Spieler 1.A.5. Kontroller warnt über Misserfolg der Angabe

Name von Use-case	Gegenstand ausgraben
Kurze Beschreibung	Spieler gräbt einen Gegenstand aus
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'DigItemUp' (Gegenstand ausgraben) 3. Kontroller speichert die Anfrage von Spieler

Name von Use-case	Gegenstand aufnehmen
Kurze Beschreibung	Spieler nimmt einen Gegenstand auf
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'PickUp' (Gegenstand aufnehmen) 3. Kontroller speichert die Anfrage von Spieler

Bemerkung: Beim Gegenstand ist gemeint Essen/ Iglu bauen oder Kapazität der Platte messen, natürlich es hängt von Charakter des Spielers (im Spiel) ab.

Name von Use-case	Schnee räumen
Kurze Beschreibung	Spieler entfernt Schnee von Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'ClearSnow' (Schnee räumen) 3. Kontroller speichert die Anfrage von Spieler

Name von Use-case	Spezialfähigkeit verwenden
Kurze Beschreibung	Spieler verwendet eine Spezialfähigkeit
Akteur	Spieler, Kontroller
Haupttätigkeit	<ul style="list-style-type: none"> 5. Kontroller bittet den Spieler einen Befehl anzugeben 6. Spieler drückt: 'UseSkill' (Spezialfähigkeit verwenden) 7. Kontroller speichert die Anfrage von Spieler

Bemerkung: Bei Spezialfähigkeit ist gemeint messen der Kapazität der Platte oder Iglo bauen, es hängt von Character des Spielers.

Name von Use-case	Zelt aufstellen
Kurze Beschreibung	Spieler stellt Zelt auf
Akteur	Spieler, Kontroller
Haupttätigkeit	<ul style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'BuildTent' (Zelt aufstellen) 3. Kontroller speichert die Anfrage von Spieler

Use-case neve	Andere Spieler retten
Rövid leírás	Spieler rettet andre Spieler aus dem Wasser
Aktorok	Spieler, Kontroller
Forgatókönyv	<ul style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'SavePlayers' (Andere Spieler retten) 3. Kontroller speichert die Anfrage von Spieler 4. Kontroller bittet den Spieler eine Richtung anzugeben 5. Spieler gibt eine Richtung ein, mithilfe von 'a/w/s/d' Tasten (links/oben/rechts/links) realisiert 6. Kontroller speichert die Richtung von Spieler

Use-case neve	Signalfackel aufbauen
Rövid leírás	Spieler baut den Signalfackel auf
Aktorok	Spieler, Kontroller
Forgatókönyv	<ul style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: 'PutSignalTogether' (Signalfackel aufbauen) 3. Kontroller speichert die Anfrage von Spieler

Use-case neve	Runde beenden
Rövid leírás	Spieler möchte die Runde beenden
Aktorok	Spieler, Kontroller
Forgatókönyv	<ol style="list-style-type: none"> 1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ' PassRound' (Spieler Runde beenden) 3. Kontroller speichert die Anfrage von Spieler

7.3 Test Plan

Test case	Used test (Food)
Beschreibung	Manual test. Wir nehmen ein Player mit ein Essen in Hand, und sehen wir das Körpertemperatur mit PrintPlayer Command, dann benutzen wir das used() Method und dann wieder PrintPlayer kommand. Wenn das Temperatur erhöht ist, ist das Method gut.
Ziel	Wir testen, ob used Method mit gutem Parameter das Körpertemperatur von einem Spieler wirklich erhöht.
Test case	Used test (DivingSuit)
Beschreibung	Manual test. Wir nehmen ein Player mit ein DivingSuit, und sehen wir das Wearing attribut mit PrintPlayer Command, dann benutzen wir das used() Method und dann wieder PrintPlayer kommand. Wenn das Wearing ein DivingSuit ist, ist das Method gut.
Ziel	Wir testen, ob used Method mit gutem Parameter das Wearing attribut von einem Spieler wirklich erhöht.
Test case	Used test (Rope)
Beschreibung	Manual test. Wir nehmen ein Player mit ein Rope und mindestens ein Player in einem benachbarten Feld, in dem Wasser. Erst sehen wir das CharacterMap mit PrintCharacterMap Command. Dann rufen wir das used Method von dem Rope und beenden wir die ganze Prozedur – wir wählen die Richtung von dem Spieler in Wasser, dann treten wir mit dem Spieler aus dem Wasser. Wenn wir das PrintCharacterMap Command wieder benutzen, sehen wir, ob die Rettung erfolgreich war. Bemerkung: es testet auch getDir() Hilfsmethod.
Ziel	Wir testen, ob used Method von Rope mit gutem Parameter anderen Spielern wirklich retten kann.

Test case	Used test (Shovel)
Beschreibung	Manual test. Wir nehmen ein Player mit ein Shovel. Erst sehen wir mit PrintSnowTileMap den Schnee, dann rufen wir used Method mit ClearSnow Command. Wir können den Schnee wieder mit PrintSnowTileMap sehen. Wenn der Wert originell 2 oder mehr war, sollte es mit originellwert-2 sein, sonst 0. Bemerkung: Wir können fragile Shovel auch testen: wenn wir es dreimal rufen, und am Ende sehen wir das inHand mit PrintPlayer. Wenn es leer ist, dann ist fragile Shovel gut.
Ziel	Wir testen, ob used Method von (fragile) Shovel mit gutem Parameter gut funktioniert.
Test case	State check test (Item)
Beschreibung	JUnit test. Wir nehmen ein Item, rufen wir die Methoden, die zu Zustandsänderungen führen (thrownDown, pickedUp, diggedUp), und testen, ob die Zustandsand (ItemState) des Items gut ist nach diesen Methoden.
Ziel	Wir testen, ob die Zustandsänderungen gut sind.
Test case	PutTogether Test
Beschreibung	Manual test. Wir sehen in ItemMap mit PrintItemMap, und PlayerMap mit PrintCharacterMap. Wenn alle sind auf einem feld, und alle SignalFlarePart ist dort, dann sollte auf dem Bildschirm: “The signal flare is done!” Sont entweder: “We don’t have all the 3 parts”oder “All players should stand here to do that”
Ziel	Test von Signalflare.
Test case	Player Test
Beschreibung	Manual integrationstest. Spieler Klasse hat ziemlich komplexe Methoden (abgesehen von Item-benutzung), das können wir manuell testen, wir haben das Ein- und Ausgangssprache, mit Kommanden und erwartete Antworten können wir es am besten überprüfen. Bemerkung: es testet IControllable auch (Player implementiert IControllable).
Ziel	Test von Player Klasse (auch IControllable).
Test case	Researcher Test
Beschreibung	Manual integrationstest. Wir testen, ob detectCapacity Method gut funktioniert. Es kann man mit der Vergleichung der erwarteten und wirklichen Ausgänge.
Ziel	Test von Researcher Klasse.
Test case	Eskimo Test
Beschreibung	Manual integrationstest. Wir testen, ob buildIgloo Method gut funktioniert. Es kann man auch mit der Vergleichung der erwarteten und wirklichen Ausgänge.
Ziel	Test von Eskimo Klasse.

Test case	PlayerContainer Test
Beschreibung	JUnit test. Unit-test von getPlayer method. Hier vergleichen wir das rückgabewert, oder das mögliche exception
Ziel	Test von getPlayer(int pid) Method.
Test case	SnowStorm Test
Beschreibung	Manual integraionstest. Es hat random Verhalten (in proto-Version gibt es auch ein deterministic Version, das testen wir auch), deswegen sollten wir es mehrmals testen mit PrintSnowTileMap Kommand. So können wir sehen, welche Tiles haben mehrere Schnee als früher. Wir können auch Zelten, Igloos und körpertemperatur ähnlicherweise beobachten.
Ziel	Sehen, dass tryStorm() funktion gut funktioniert.
Test case	Tile Test (Auch für abgeleitete Klassen: StableTile, UnstableTile, SnowyHole)
Beschreibung	Maunal integraionstest. Wir testen getNeighbour() Method, und die steppedOff und steppedOn funktionen von abgeleiteten Klassen (StableTile, UnstableTile, SnowyHole) in CLI.
Ziel	In diesem Test testen wir die komplexere Methoden von Tile (auch in abgeleiteten Klassen nach override) Klasse.
Test case	RoundController test
Beschreibung	Manual integraionstest. Test von komplexen Methoden in CLI.
Ziel	Test von startNextRound und endLastRound.
Test case	PositonLUT test
Beschreibung	JUnit test. Unit test von Containermanipulationsmethoden. In diesem Test prüfen wir, ob die Methode die Containern gut modifizieren. Wir sehen hier, dass das Method zu Container etwas addiert, wegnimmt und/oder etwas als Rückgabe gibt. Wir sehen auch nach exceptions.
Ziel	Unit Test für alle Methode, die ein Container modifizieren oder benutzen. Methoden: setPosition mit verschiedenem Parameter, getTile, getItemOnTile, getPlayersOnTile, getPosition mit verschiedenem Parameter.
Test case	PolarBear test
Beschreibung	Manual integraionstest. Normaleweise tritt das PolarBear zufällig, man braucht mehrere schritte(ähnlich zu SnowStorm test) um das sehen, ob es gut ist. In proto Version wird auch ein deterministic Drehbuch, um zu testen und demonstrieren.

Ziel	Test von step Method von PolarBear Klasse.
Test case	Used Test (Tent)
Beschreibung	Manual integraionstest. Wir nehmen ein Player mit ein Tent in Hand, und sehen wir was passiert, wenn er es benutzt.
Ziel	Test von used Method von Tent Klasse.

7.4 Spezifikation der Test unterstützenden Hilfs- und Übersetzungsprogramme

Wir werden JUnit benutzen um Unit-testen zu machen. Es hat built-in support in IntelliJ IDEA. Mit JUnit können wir testen das erwartete Ergebnis von einem Method mit konkreten Parametern gut ist, wir können auch erwartete exceptions prüfen.

8. Detaillierte Pläne

DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia	SOSK6A	adamzsofi.mail@gmail.com
Hedrich Ádám	H9HFFV	hedrichadam09@gmail.com
Pintér Balázs	ZGY18G	pinterbalazs21@gmail.com
Fucskár Patrícia	XKYAOO	fucskar.patricia@gmail.com
Tassi Timián	MYY53U	timian.tassi@gmail.com

15. April 2020

8. Detaillierte Pläne

8.1 Pläne von Klassen und Methoden.

8.1.1 Player

- **Verantwortung**

Diese Klasse ist eine abstrakte Klasse, weil ein Spieler entweder Eskimo oder Forscher ist. Diese Klasse enthält die grundsätzlichen Eigenschaften (Attributen) und die gemeinsamen Methoden beider Arten von Spielern. Natürlich die Eskimos und Forscher Klasse spezialisiert diese Klasse, also enthält weitere Methoden.

- **Schnittstellen**

- **IControllable**

- **SuperKlassen**

Character->Player

- **Attributen**

- **#int Bodyheat:** Die Körpertemperatur des Spielers. Es ist 5 Einheit für Eskimos und 4 Einheit für Forscher.
- **#int ID:** für eindeutige Charakterisierung des Spielers
- **#int workPoints:** Der Spieler hat so viele Arbeit (Einheit) noch.
- **#Item inHand:** Der Gegenstand, der in der Hand des Spielers sich befindet. Es kann 0 oder 1 Gegenstand in der Hand sein. (mehr als 1 nicht)
- **#DivingSuit wearing:** Es ist eigentlich der Taucheranzug, wenn der Spieler es trägt.
- **+boolean inWater:** Es zeigt, ob der Spieler im Wasser ist. (true: Ja, false: Nein)

- **Methoden**

- **+void startRound():** Mit dieser Methode startet ein Spieler ihre Runde.
- **+void fallInWater():** Der Spieler fällt ins Wasser. Er wartet auf die Rettung.
- **+void changeBodyHeat(int thisMuch):** Der Spieler kann sich seine Körpertemperatur erhöhen (mit dem Essen) oder es wird durch eine Schneesturm verringert.
- **+void ateFood():** Der Spieler isst sein Essen, so seine Körpertemperatur wird sich erhöht. (*Bemerkung: dieses Method wird von dem Essen in seinem Hand aufgerufen, weil Essen wird immer automatisch gegessen, falls es aufgenommen ist*)
- **+void wear(DivingSuit suit):** Der Spieler trägt den Taucheranzug. Wenn er ins Wasser fällt, wird nicht sterben. (*Bemerkung: dieses Method wird von dem Taucheranzug in seiner Hand aufgerufen, weil das Taucheranzug immer automatisch angezogen wird, falls es aufgenommen ist*)
- **+void pickUp(Item i):** Der Spieler kann damit einen Gegenstand aufnehmen. Wenn in seiner Hand sich schon ein Gegenstand befindet, wird der vorherige Gegenstand abwerfen.
- **+void clearSnow():** Der Spieler räumt Schnee von der Eisplatte. (*1 oder 2, hängt davon ab, ob das Spieler eine Schaufel in der Hand hat*)
- **+void digItemUp(Item i):** Der Spieler gräbt ein Gegenstand aus.

- **+void savePlayers(Direction dir):** Ein Spieler kann seinen Kumpel retten. Der Spieler muss eine Richtung eingeben.
- **+void putSignalTogether(SignalFlare sf):** Der Spieler setzt der SignalFackel zusammen.
- **+void passRound():** Der Spieler passt seine Runde.
- **+void dropFragileShovel():** Der Spieler wirft seine zerbrechliche Schaufel ab. (*Wird von zerbrechlicher Schaufel aufgerufen, ähnlich wie beim ateFood/wear*)

8.1.2 Character

- **Verantwortung**

Diese Klasse ist eine abstrakte "Superklasse". Es ist eigentlich ein Container für alle Charakter, die in dem Spiel vorkommen kann. Es gibt eine gemeinsame Eigenschaft zwischen allen Charakter: der Schritt.

- **Methoden**

- **+void step(Direction dir):** Diese Methode wird gerufen, wenn ein Charakter treten möchte. Wir müssen die Richtung eingeben.

8.1.3 IControllable

- **Verantwortung**

IControllable ist eine Schnittstelle. Einige Methoden für die Spieler befinden sich hier. Diese Methoden werden durch Klasse "Player" benutzt: Treten, Gegenstand aufnehmen, schneeräumen, Gegenstand ausgraben, Teile vom Signalfackel zusammenbauen, Runde passen.

- **Methoden**

- **+ void passRound():** Mit dieser Methode kann der Spieler passen, also das Recht für Tritt weitergeben.
- **+void digItemUp():** Der Spieler räumt mit Hilfe von einem Gräber Schnee, wenn er ein Gräber hat.
- **+void pickUp(Item i):** Der Spieler kann einen Gegenstand aufnehmen, wenn der Gegenstand in die Eisplatte nicht einfriert.
- **+void clearSnow():** Der Spieler macht eine Eisplatte sauber.
- **+void putSignalTogether(Signalflare sf):** Am Ende des Spiels feuern ein Spieler die Signalfackel. Es ist möglich, wenn alle Spieler in einer Eisplatte steht und die Teile von der Signalfackel da sind. Es kostet 1 Arbeit (Einheit).
- **+void savePlayers(Direction dir):** Der Spieler rettet sein Kumpel mit Hilfe vom Seil. Man muss eingeben, auf welche Eisplatte (Direction) wird der Spieler gerettet.

8.1.4 Eskimo

- **Verantwortung**

Diese Klasse spezialisiert die Player Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Superklasse, weil allen Attributen in beiden Subklassen vorkommen, die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **SuperKlassen**

Character->Player->Eskimo

- **Methoden**

- **+void buildIgloo():** Die Eskimos können ein Iglu bauen. Es schützt von dem Schneesturm und von dem Eisbären.

8.1.5 Researcher

- **Verantwortung**

Diese Klasse spezialisiert die “Spieler“ Klasse. Es definiert weitere Methoden, weil diese Klasse anders verhalten kann, als die andere Spieler Klasse. Den Attributen von dieser Klasse definiert in der Vorklasse, weil allen Attributen beiden Subklassen vorkommt, nur die Grundwerte können anders sein. (zum Beispiel Körpertemperatur).

- **SuperKlassen**

Character->Player->Researcher

- **Methoden**

- **+void detectCapacity(Direction dir):** Die Forscher können mit dieser Methode anschauen, ob eine Eisplatte stabil oder instabil ist, also wie viel Spieler kann auf einer bestimmten Eisplatte stehen.

8.1.6 PolarBear

- **Verantwortung**

Diese Klasse verwirklicht der Eisbär. Der Eisbär tritt in jeder Runde in einer zufälligen Richtung. Er kann schwimmen. Diese Klasse vererbt die Superklasse Character.

- **SuperKlassen**

Character->PolarBear

- **Methoden**

- **+void step(Direction dir):** Die “Step” Methode von Character wird überschreibt.

8.1.7 Item

- **Verantwortung**

Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanziiieren. Es ist ein Container von verschiedenen Gegenständen. Die gemeinsame Eigenschaft von verschiedenen Gegenständen ist, dass es in dem Spiel immer 1 von jedem gibt.

- **Attributen**

- **+ItemState state:** Es ist der Zustand des Gegenstands. Im Konstruktor wird es am Anfang auf "frozen" eingestellt.

- **Methoden**

- **+void thrownDown():** Der Gegenstand wird abgeworfen. In diesem Fall ist der Gegenstand in die Eisplatte nicht eingefroren und bleibt ebendorf.
 - **+void pickedUp(Player picker):** Der Gegenstand wird durch einen bestimmten Spieler aufgenommen. Der Spieler wird im Parameter gegeben.
 - **+void diggedUp():** Mit dieser Methode können wir den Gegenstand ausgraben.
 - **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch einen bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten.
 - **+void ItemState getState():** Es gibt den Zustand des Gegenstands zurück.

8.1.8 Shovel

- **Verantwortung**

Diese Klasse spezialisiert die "Item" Klasse. Diese Klasse verwirklicht den Schaufel Gegenstand.

- **SuperKlassen**

Item->Shovel

- **Methoden**

- **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch einen bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten. (*override*)

8.1.9 Rope

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Seil Gegenstand.

- **SuperKlassen**

Item->Rope

- **Methoden**

- **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch ein bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten. (*override*)
- **-Direction getDir():** Wir bekommen eine Richtung, wohin wir den anderen Spieler retten möchten.

8.1.10 DivingSuit

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Taucheranzug Gegenstand.

- **SuperKlassen**

Item->DivingSuit

- **Methoden**

- **+ void pickedUp(Player picker):** Der Gegenstand wird durch einen bestimmten Spieler aufgenommen. Der Spieler wird im Parameter gegeben. (*override*)

8.1.11 Food

- **Verantwortung**

Diese Klasse spezialisiert die “Item“ Klasse. Diese Klasse verwirklicht den Essen Gegenstand. Mit diesem Gegenstand können die Spieler sich ihre Körpertemperatur erhöhen. Falls das Essen gegessen ist, dann es wird auf ein Tile gefroren nochmal erschienen (it “respawns”).

- **SuperKlassen**

Item->Food

- **Methoden**

- **+ void pickedUp(Player picker):** Der Gegenstand wird durch einen bestimmten Spieler aufgenommen. Der Spieler wird im Parameter gegeben. (*override*)

8.1.12 Tent

- **Verantwortung**

Diese Klasse verwirklicht das Zelt. Wir haben es als Gegenstand behandelt, aber nur bis es gebaut wird, danach wird es nur als ein “Zähler” arbeiten (nach einem ganzen Runde wird es zerstört).

- **SuperKlassen**

Item->Tent

- **Attributen**

- **+int counter:** Es zählt das Leben des Zelts. Wenn es größer als die Anzahl der Spieler, wird der Zelt verschwinden.
- **+int x,y:** Die Position des Zelt.

- **Methoden**

- **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch ein bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten. (*override*)

8.1.13 FragileShovel

- **Verantwortung**

Diese Klasse verwirklicht die zerbrechliche Schaufel. Es vererbt die Klasse Item.

- **SuperKlassen**

Item->Shovel->FragileShovel

- **Attributen**

- **-int counter:** Es zählt das Leben des Schaufels. Wenn es größer oder gleich als 3, wird der Schaufel zerbrechen. (*Es wird 1 größer bei jeder Benutzung*)

- **Methoden**

- **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch ein bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten. (*override*)

8.1.14 ItemState

- **Verantwortung**

ItemState ist eine Enumeration. In dieser Enumeration befinden sich die Zustände von einem Gegenstand. Keine anderen Zustände existieren. In einem Augenblick kann sich zu einem Gegenstand pünktlich ein Zustand gehören.

- **Aufzählungsliterale**

- **frozen:** Wenn ein Gegenstand gefroren ist, gehört sich zu diesem Zustand.
- **inHand:** Wenn ein Gegenstand sich in der Hand von einem Spieler befindet, gehört sich zu diesem Zustand.
- **thrownDown:** Ein Gegenstand kann auf eine Eisplatte nicht in gefroren Zustand sein, also durch einen Spieler abgeworfen wird. In diesem Fall gehört sich zu diesem Zustand

8.1.15 Activity

- **Verantwortung**

Diese Klasse ist eine Enumeration. Es ist gegeben, welche Aktivitäten ein Spieler durchführen kann. Diese Enumeration speichert diese Möglichkeiten. Andere Aktivitäten können während des Spiels nicht durchgeführt werden.

- **Aufzählungsliterale**

- **savingPeople:** Ein Spieler kann einen anderen Spieler von dem Wasser retten.
- **clearingSnow:** Die Spieler können die Eisplatten saubermachen, also der Schnee wird von der Eisplatte weggeputzt.
- **eatingFood:** Wenn ein Spieler sich mit einem Essen begegnet, isst automatisch es. Danach wird ein anderes Essen irgendwo erschien. Während dieser Tätigkeit wird sich die Körpertemperatur von dem bestimmten Spieler erhöht.
- **puttingOnSuit:** Der Spieler nimmt den Taucheranzug auf sich. Es wird automatisch durchgeführt, wenn ein Spieler einen Taucheranzug aufnehmen
- **putUpTent:** Der Spieler kann ein Zelt auf eine Schneepalte bauen, falls dort sich kein Iglu befindet.

8.1.16 RoundController

- **Verantwortung**

Diese Klasse ist eine Singleton Klasse. Diese Klasse steuert das Ende und der Beginn eines Spiels, also es ist ein Kontroller. Es beinhaltet ein Spieler Container. In diesem Container befinden sich den Spielern. Die Runden werden auch in dieser Klasse kontrolliert. Am Anfang des Spiels kann man durch diese Klasse den Spielern initialisieren.

- **Attributen**

- **-int curID:** Es speichert, welche Spieler kommt. Es speichert den Identifikationsnummer des Spielers.
- **+SignalFlare sg:** Es ist der Signalfackel als Attribut in dieser Klasse.

- **+SnowStorm ss:** Es ist der Schneesturm als Attribut in dieser Klasse.
- **-static RoundController rc:** Globale, Singleton RoundController. Es können wir mit der “`getInstance()`” Methode abfragen.
- **+ArrayList<PolarBear> polarbearList:** Die Eisbären werden hier gespeichert und kontrolliert. (*Normalerweise haben wir 1 Eisbär, aber die Möglichkeit für mehrere ist mit diesen List gegeben*)
- **+Tent tent:** Der Zustand des Zelts können wir hier kontrollieren. Wenn eine Runde wird ablauf, dann verschwindet das Zelt.

- **Methoden**

- **+void init(int playerNum):** Mit dieser Methode können wir einen Spieler initialisieren. Dazu müssen wir ein Spieler Nummer eingeben. Diese Nummern identifizieren eindeutig den Spielern, also müssen unterschiedlich sein.
- **+void startNextRound():** Am Ende der Runde wird die Nächste gerufen. Der erste Spieler der Round setzt das Spiel fort.
- **+void endLastRound():** Die jetzige Runde beenden.
- **+void lose(String cause):** Das Spiel beendet. Der Grund der Niederlage wird eingegeben.
- **+void win():** Die Spieler haben gewonnen, das Spiel wird beendet.
- **+ int getcurID():** Es gibt den Identifikationsnummer des derzeitigen Spieler zurück.
- **+ void checkTent():** Nach den Treten des Spieler wird das Lebens des Zelt vermindern. Wenn eine ganze Runde beendet wird, verschwindet das Zelt.
- **+ static RoundController getInstance():** Es gibt der “`rc`” zurück.

8.1.17 SignalFlare

- **Verantwortung**

Diese Klasse ist eigentlich die Signalfackel. Dieser Gegenstand besteht aus 3 Teile: Pistole, Leuchte, Patron. Mit diesem kann die Spieler das Spiel gewinnen.

- **Attributen**

- **-ArrayList<SignalFlarePart> signalFlareParts[3]:** Es ist ein Array und dieser Array beinhaltet die 3 Teile vom Signalfackel. Kein Teil des Signalfackels kann verschwinden.

- **Methoden**

- **+void putTogether(RoundController rc):** Mit dieser Methode können die Spieler das Spiel beenden. Wenn alle Spieler auf derselbe Eisplatte stehen und sie haben alle 3 Signalfackel Teil in der Hand, dann es kostet eine Arbeit diese Methode zu rufen und das Spiel zu beenden.

8.1.18 PlayerContainer

- **Verantwortung**

Diese Klasse beinhaltet die Spieler. Die Spieler werden durch ihren Identifikationsnummer eindeutig identifiziert.

- **Attributen**

- **-int playerNum:** Die Anzahl des Spielers.
- **-static PlayerContainer pc:** Es ist eine Instanz vom Playercontainer. Es ist globale und es können wir mit der “`getInstance()`” Methode abfragen.
- **-ArrayList<Player> players:** Diese Liste beinhaltet die Spieler in dem Spiel.

- **Methoden**

- **+ static PlayerContainer getInstance():** Es gibt der “`pc`” zurück.
- **+Player getPlayer(int pid):** Diese Methode gibt einen Spieler zurück. Wir müssen den Identifikationsnummer dem bestimmten Spieler eingeben. Die verschiedenen Nummern werden zu verschiedenen Spielern zugeordnet.
- **+static void Initialize(int num):** Wir können am Anfang einen neuen Playercontainer initialisieren.
- **+int getPlayerNum():** Es gibt “`playerNum`” zurück.

8.1.19 SnowStorm

- **Verantwortung**

Diese Klasse erzeugt den Schneesturm. Dieser Sturm verringert die Körpertemperatur des Spielers, falls er sich nicht in einem Iglu oder in einer Zelt und erhöht sich die Dicke der Schneeschichten auf die verschiedenen Eisplatten.

- **Methoden**

- **+ void tryStorm():** Diese Methode erzeugt eigentlich den Schneesturm.

8.1.20 SignalFlarePart

- **Verantwortung**

Diese Klasse speichert ein Teil von Signalfackel. Es gibt 3 Teile und das Ziel des Spiels ist, diese 3 einzubauen und abzuschießen. Die verschiedenen Teile sind verschiedene Instanzen von dieser Klasse

- **SuperKlassen**

Item->SignalFlarePart

- **Attributen**

- **-int partID:** Die Identifikationsnummer von dem Teil. Es muss eindeutig sein.

- **Methoden**

- **+void used(Player player, Activity activity):** Die verschiedenen Gegenstände können durch einen bestimmten Spieler benutzt werden. Wir müssen die Aktivität auch eingeben, also was wir mit dem bestimmten Gegenstand machen möchten.

8.1.21 PositionLut

- **Verantwortung**

Diese Klasse ist eine Singleton Klasse. Diese Klasse handelt sich die Bewegungen von den Spielern. Den Spielern können mit Hilfe von dieser Klasse von einer Platte auf eine andere Platte treten. Des Weiteren können wir die Position von verschiedenen Spielern abfragen oder welchen Gegenständen befindet sich in einer bestimmten Eisplatte.

- **Attributen**

- **#static PositionLUT pLUT:** Unser einzige “PositionLUT” Instanz.
- **-HashMap<Item, Tile> itemTileMap:** Hier wird die Eisplatte-Gegenstand Paare gespeichert.
- **-HashMap<Tile,ArrayList<Item>> tileItemMap:** Zu der Eisplatte gehörende Liste des Gegenstandes.
- **-HashMap<Player, Tile>playerTileMap:** Hier wird der Spieler-Gegenstand Paare gespeichert.
- **-HashMap<PolarBear, Tile> polarbearTileMap:** Hier wird zu dem Eisbären eine Eisplatte geordnet. (Wo er sich befindet.)
- **-HashMap<Tile,ArrayList<PolarBear>> tilePolarBearMap:** Zu der Eisplatte gehörende Liste des Eisbären. (*Jetzt haben wir nur ein Eisbär, aber wir haben die Möglichkeit, mehrere zu haben.*)
- **-ArrayList<ArrayList<Tile>> tileList:** Die Liste der Eisplatten.
- **-HashMap<Tile,ArrayList<Player>> tilePlayerMap:** Zu der Eisplatte gehörende Liste des Spielers.

- **Methoden**

- **+Tile getPosition(Player p):** Mit dieser Methode können wir die Position des bestimmten Spieler abfragen. Wir müssen einen Spieler eingeben und es wird zurückgegeben, auf welche Eisplatte steht er.
- **+Tile getPosition(Item i):** Mit dieser Methode können wir die Position des bestimmten Gegenstand abfragen. Wir müssen einen Gegenstand eingeben und es wird zurückgegeben, auf welche Eisplatte befindet sich es.
- **+ArrayList<Player> getPlayersOnTile(Tile t):** Diese Methode ergibt eine Spieler ArrayList. In dieser ArrayList befindet sich ein Spieler, falls er auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.
- **+ArrayList<Item> getItemOnTile(Tile t):** Diese Methode ergibt eine Gegenstand ArrayList. In dieser ArrayList befindet sich ein Gegenstand, falls es auf die eingegebenen Platte steht, also müssen wir eine Eisplatte eingeben.
- **+Tile getTile(int x, int y):** Es gibt eine Eisplatte anhand der Koordinaten zurück.
- **+void setPosition(Player p, Tile t):** Die Position des Spielers wird auf eine Eisplatte eingestellt.
- **+void setPosition(Player p, Item i):** Die Position des Gegenstandes wird auf eine Eisplatte eingestellt.

- **+Tile getPosition(PolarBear pb):** Es gibt die Position des Eisbären zurück.
- **+void setPosition(PolarBear pb, Tile t):** Die Position des Eisbären wird auf eine Eisplatte eingestellt.

8.1.22 Tile

- **Verantwortung**

Diese Klasse ist ein Container, also zusammenfasst alle Art von den Eisplatten. Die Klasse ist abstrakt, man kann es nicht instanziiieren. Es gibt 3 Arten von den Eisplatten. Es kann stabil, instabil oder ein schneebedecktes Loch sein. Diese Arten sind verschiedene Subklassen, vererben diese abstrakten Klasse.

- **Attributen**

- **#int x:** Es ist die x Koordinate (horizontale) von der bestimmten Eisplatte.
- **#int y:** Es ist die y Koordinate (vertikale) von der bestimmten Eisplatte.
- **#int snow:** Diese Attribute speichert, wie viel Schnee (wie viel Einheit) sich auf die Eisplatte befindet.
- **#boolean iglooOn:** Diese Attribute bestimmt, ob ein Iglu sich auf diese Eisplatte befindet. (true: Ja, false: Nein)
- **+boolean tentOn:** Es zeigt, ob ein Zelt sich auf diese Eisplatte befindet.
- **#int capacity:** Die Kapazität der Eisplatte.
- **+int standingHere:** Die Anzahl der Spieler, die auf diese Platte stehen.

- **Methoden**

- **+void steppedOn(Player p):** Ein Spieler tritt auf die Eisplatte. Wir müssen den Spieler eingeben.
- **+void steppedOff(Direction dir):** Ein Spieler tritt auf eine andere Eisplatte. Wir müssen die Richtung eingeben. Direction ist eine Enumeration, also von bestimmten Richtungen (4 Himmelsrichtung) können wir wählen.
- **+void changeSnow(signed int thisMuch):** Die Höhe des Schneeschicht verändert sich. Es wird eingegeben, mit wie vielen. Es kann höher oder kleiner sein. Zum Beispiel höher, falls ein Schneesturm kommt.
- **+Tile getNeighbour(Direction dir):** Mit dieser Methode können wir die benachbarten Eisplatten von einer Platte abfragen. Wir müssen die Richtung auswählen und eingeben.
- **+void destroyIgloo():** Diese Methode baut den Iglu, der auf diese Eisplatte steht, ab.
- **+void buildIgloo():** Eskimos können auf eine bestimmten Platte ein Iglu bauen.
- **+int getSnow():** Diese Methode gibt die Schneedicke der Eisplatte zurück.
- **+int getCapacity():** Diese Methode gibt die Kapazität der Eisplatte zurück.
- **+boolean getIglooOn():** Diese Methode gibt den Wert des "iglooOn" Attribut zurück.
- **+void putOnTent():** Mit dieser Methode kann der Spieler ein Zelt auf die Eisplatte bauen.
- **+boolean equals(Tile t):** Es liefert zurück, ob die Position von zwei Eisplatten gleich ist.

8.1.23 SnowyHole

- **Verantwortung**

Diese Klasse spezialisiert die "Tile" Klasse. Diese Klasse verwirklicht das schneebedeckte Loch. Diese Eisplatten siehe so aus, wie die andere schneebedecktes Platte, aber wenn ein Spieler auf diese tritt, fällt ins Wasser. Auf diesen Platten können 0 Spieler stehen.

- **Superklasse**

Tile-> SnowHole

- **Methoden**

- **+void steppedOn(Player p):** Ein Spieler tritt auf die Eisplatte. Wir müssen den Spieler eingeben. (*override*)
- **+void destroyIgloo():** Diese Methode baut den Iglo, der auf diese Eisplatte steht, ab. (*overrirde*)
- **+void buildIgloo():** Eskimos können auf eine bestimmten Platte ein Iglo bauen. (*override*)

8.1.24 StableTile

- **Verantwortung**

Diese Klasse spezialisiert die "Tile" Klasse. Diese Klasse verwirklicht die stabile Eisplatte. Auf diesen Platten können unendlich viel Spieler stehen. Auf diesen Platten kann sich natürlich auch Schnee befinden.

- **Superklasse**

Tile-> SnowHole

8.1.25 UnstableTile

- **Verantwortung**

Diese Klasse spezialisiert die "Tile" Klasse. Diese Klasse verwirklicht die instabile Eisplatte. Auf diesen Platten können nur begrenzt viel Spieler stehen. Die Anzahl den Spielern ist nicht sichtbar, aber die Forscher können es anschauen.

- **Superklasse**

Tile-> SnowHole

- **Methoden**

- **+void steppedOn(Player p):** Ein Spieler tritt auf die Eisplatte. Wir müssen den Spieler eingeben. (*override*)
- **+void steppedOff(Direction dir):** Ein Spieler tritt auf eine andere Eisplatte. Wir müssen die Richtung eingeben. Direction ist eine Enumeration, also von bestimmten Richtungen (4 Himmelsrichtung) können wir wählen. (*override*)

8.1.26 Direction

- **Verantwortung**

Diese Klasse ist eine Enumeration und beinhaltet die verschiedenen Richtungen. Wenn ein Spieler treten möchte, sollte er von diesen Richtungen wählen. Es ist eigentlich die 4 Himmelrichtungen. Andere Richtung existiert in diesem Spiel nicht, also können die Spieler diagonale weise nicht treten.

- **Aufzählungsliterale**

- **up:** Auf die nördliche Platte treten. (Nord)
- **down:** Auf die untere Platte treten. (Süd)
- **left:** Auf die linke Platte treten. (West)
- **right:** Auf die rechte Platte treten. (Ost)
- **here:** Der Spieler bleibt auf die Platte.

- **Attributen**

- **-int value:** Zu der bestimmten Richtung gehörende Wert. (zwischen 0 und 4)
- **-static HashMap<int,Direction> map:** Es speichert der Wert-Direction Paare.

- **Methoden**

- **+static Direction valueOf(int direction):** Es ist eine int->Direction Umwandlung.
- **+static int valueOf(Direction dir):** Es ist eine Direction->int Umwandlung.
- **+int getValue():** Es gibt den “value” Attribut zurück.

8.2 Detaillierte Pläne von Testfällen, Beschreibungen

Wir testen das Programm mit dem deterministischen Map, so brauchen wir PrintMap (Character, Item) Kommanden nicht unbedingt, es kann aber manchmal hilfreich sein. Wir können Printkommanden auch mit watch Funktion von Debugger ersetzen, das ist aber nicht unbedingt bequemer.

8.2.1 Used test(Food)

- **Beschreibung**

Beobachtung von Temperaturänderung, wenn ein Spieler ein Essen (Food) aufnimmt.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Testet used Funktion von Food und pickedUp Funktion von Player

- **Eingänge**

PrintPlayer //Körpertemperatur check

PrintCharacterMap //sehen wo unsere Spieler steht

PrintItemMap //ItemCheck

PickUp //Aktion, Food addiert sofort 1 Temperatur

PrintItemMap //sehen ob das Essen verschwindet

PrintPlayer //Körpertemperatur check

- **Erwartete Ausgänge**

Player Eigenschaften //Beobachtung, jetzt Temperatur ist wichtig für uns

PlayerMap //Beobachtung

ItemMap //Beobachtung

„Picked up Food“ //erwartete, falls ein Food und aktivPlayer auf einem Eisplatte sind, sonst:

„Nothing to pick up“// „Picked up <other Item>“

ItemMap //Food musste verschwinden

Player Eigenschaften // Temperatur muss mit 1 grösser sein, falls „Picked up Food“ abläuft

8.2.2 Used test(DivingSuit)

- **Beschreibung**

Wir nehmen ein Player und ein DivingSuit auf derselben Eisplatte, und sehen wir, dass das Divingsuit in Wearing geht.

- **Getestete Funktionalitäten, erwartbare Fehler**

Test von used Funktion von DivingSuit, wear und pickUp Method von Player.

- **Eingänge**

PrintPlayer //Beobachtung

PrintCharacterMap //Beobachtung

PrintItemMap //Beobachtung

PickUp //Aktion

PrintItemMap //DivingSuit verschwindet

PrintPlayer //Wearing attribut check

- **Erwartete Ausgänge**

Player Eigenschaften //Beobachtung, jetzt Temperatur ist wichtig für uns

PlayerMap //Beobachtung

ItemMap //Beobachtung

„Picked up DivingSuit“ //erwartete, falls ein DivingSuit und Player auf einem Eisplatte sind

ItemMap //DivingSuit musste verschwinden

Player Eigenschaften //Wearing attribut muss DivingSuit sein.

8.2.3 Used test(Rope)

- **Beschreibung**

Wir nehmen ein Player mit ein Rope und mindestens ein Player in einem benachbarten Feld, in dem Wasser. Wir testen hier die Rettung.

- **Getestete Funktionalitäten, erwartbare Fehler**

Hier testen wir savePlayer Method von Player, used Method von Rope, step Method von Player kann man auch hier testen, ob es gut funktioniert.

- **Eingänge**

PrintCharacterMap//Beobachtung

SavePlayers <dir>//Richtung wählen

Step <dir>//Spielern in Wasser können treten (falls mehrere Spieler, dann mehrere Step)

PrintCharacterMap//War die Rettung erfolgreich?

- **Erwartete Ausgänge**

PlayerMap

„Saved Players <Player IDs> //falls möglich sonst no rope oder no player to save Warnungen

“Step successful, stepped from (x,y) to (x,y)” //falls Spielern treten auf gute Feld, sonst: „step not succesful“ oder „Player fell in water“ //letzte wird auf deterministische Map nicht möglich.

PlayerMap//check, Spielern sind nicht auf dem WasserFeld.

8.2.4 Used test(Shovel)

- **Beschreibung**

Wir nehmen ein Player mit einer Schaufel. Wir testen hier Schneeräumung mit Schaufel.
Bemerkung: Wir können eine zerbrechliche Schaufel auch testen: wenn wir es dreimal rufen, und am Ende sehen wir das inHand mit PrintPlayer. Wenn es leer ist, dann funktioniert die zerbrechliche Schaufel gut.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Testet clearSnow Method von Player, used Method von Schaufel.

- **Eingänge**

PrintSnowMap// Beobachtung

PrintCharactersMap//Beobachtung

PrintPlayer //InHand

Shovel/fragileShovel?

ClearSnow//Aktion

PrintSnowMap// snow auf Tile muss mit 2 weniger sein (kann nicht negativ sein, min = 0)

Optional für zerbrechliches Schaufel: nach 3 erfolgreiche Clearsnow: PrintPlayer, InHand muss leer sein.

- **Erwartete Ausgänge**

SnowMap

CharacterMap

Player Eigenschaften

“Cleared <1 oder 2> Snow off from Tile.
”//clear confirm

SnowMap//snow prüfen auf Tile, wo Player das Schnee räumt. es muss 2 weniger sein (>=0)

Optional für FragileShovel: Player Eigenschaften nach 3 „Cleared“, InHand muss leer sein

8.2.5 Used test(Tent)

- **Beschreibung**

Wir nehmen ein Player mit einem Zelt in Hand, und sehen wir was passiert, wenn er es benutzt.

- **Getestete Funktionalitäten, erwartbare Fehlern**

Testet used Method von Zelt.

- **Eingänge**

PrintHeimMap//map von Iglus und Zelt

BuildTent//Aktion

PrintHeimMap//sehen ob Zelt platziert ist oder nicht

- **Erwartete Ausgänge**

HeimMap

“Successfully built a tent”// kann „Can’t build tent here“ oder „I have no tent in my hand“ sein

HeimMap// falls erfolgreich: mit neuem Zeltoptional: wenn wir nach einer Rund es sehen, muss das Zelt verschwinden

8.2.6 PutTogether Test

- **Beschreibung**

Wir sehen in ItemMap mit PrintItemMap, und PlayerMap mit PrintCharacterMap. Wenn alle sind auf einem feld, und alle SignalFlarePart ist dort, dann sollte auf dem Bildschirm: “The signal flare is done!” Sonst entweder: “We don’t have all the 3 parts” oder “All players should stand here to do that”

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Testet putTogether Methode von Signalfackel.

- **Eingänge**

PrintCharacterMap//alle Player muss auf einem Feld sein

PrintItemMap//hier müssen wir kein SignalFlarePart sehen→ die sind inHand

PutSignalTogether//Aktion

- **Erwartete Ausgänge**

CharacterMap//alle Player muss auf einem Feld sein

ItemMap//hier müssen wir kein SignalFlarePart sehen→ die sind inHand

“The signal flare is done!”// wenn Bedingungen sind gut, sonst kann: “We don’t have all the 3 parts” /“All players should stand here to do that”

8.2.7 Player Test

- **Beschreibung**

Test von oben noch nicht getesteten Methoden in Player Klasse.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Es testet startRound, fallInWater, digItemUp und passRound Methoden von Player, andere Methoden haben wir früher getestet in used-testen.

- **Eingänge**

PassRound// 2mal Aktion

PrintItemMap//Beobachtung(Item& Zustand)

DigItemUp//mit einem Player über ein Item mit frozen itemState.

PrintItemMap//Zustandsänderungs-check

PrintCharacterMap

Step < dir : in Wasser >//fallInWaterTest, step ist schon getestet

PrintPlayer

- **Erwartete Ausgänge**

“Player <ID> passed”// wenn es erschien zweimal, mit verschiedene ID-s, dann wissen wir, dass aktuelle Player verändert. Es testet passRound und startRound auch.

ItemMap//Items mit Zuständen

„digged up <Item>”// diggedUp test. Könnte auch „no frozen Item here” sein.

ItemMap//digged up Item muss in thrownDown Zustand sein.

CharacterMap// wo ist aktuelle Player

“Player fell in water”//fallInWater Methode geruft

Player Eigenschaften//inWater ==true

8.2.8 Researcher Test

- **Beschreibung**

Wir testen mit einem Researcher, ob detectCapacity Method gut funktioniert. Es kann man mit der Vergleichung der erwarteten und wirklichen Ausgänge machen.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

DetectCapacity Method von Researcher wird getestet.

- **Eingänge**

UseSkill <dir>//sehen das capacity won gegebene Tile

PrintTile <x,y>//Vergleichung, x,y sind von dir und playerpos kalkuliert

- **Erwartete Ausgänge**

“Capacity of Tile (x, y) is <Capacity>” //number, kann „no tile there” sein

Tile Eigenschaften//capacity muss gleich sein, falls Tile existiert

8.2.9 Eskimo Test

- **Beschreibung**

Wir testen mit einem Eskimo, ob buildIgloo Method gut funktioniert. Es kann man auch mit der Vergleichung der erwarteten und wirklichen Ausgänge.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Testet buildIgloo Method.

- **Eingänge**

PrintHeimMap //Beobachtung

UseSkill //Eskimo baut ein Iglu

PrintHeimMap //check

- **Erwartete Ausgänge**

HeimMap//Beobachtung

“Successfully built an igloo”// “Can’t build igloo here”: wenn ein Iglu schon dort war

HeimMap //ein neuer Iglu muss erschienen.

8.2.10 TileTest (Auch für abgeleitete Klassen: StableTile, UnstableTile, SnowyHole)

- **Beschreibung**

Dieser Test ist in 8.2.1-8.2.9 Testfallen schon gemacht, wir können diese Klassen „direkt“ mit Kommanden nicht erreichen, aber andere Methoden benutzen es und der fehlerfreie Ablauf von diesen Methoden sind benötigt.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Wir haben getNeighbour() Methode, und die steppedOff und steppedOn Funktionen von abgeleiteten Klassen (StableTile, UnstableTile, SnowyHole) getestet. In getNeighbour Funktion soll man auf Indexierung und gute Rückgabewert besonders achten

8.2.11 RoundController Test

- **Beschreibung**

Dieser Test ist schon fast gemacht, wenn man das Spiel benutzt und es scheint gut laufend, dann ist es ok. (Dieser Test ist basiert auf anderen Testen die auch gut laufen)

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Hier Testen wir nur lose Methode.

- **Eingänge**

Step <in water dir>//ein Spieler geht ins Wasser

PassRound//jeder Spieler passt

- **Erwartete Ausgänge**

“Player fell in water”

“Player <ID> passed”//Anzahl:
SpielerAnzahl – 1

„lose <cause>”//in diesem Fall <cause> == „drowning”, kann aber „polarBearAttack” oder „cold” wenn wir das deterministisch proto Map benutzen, dann können wir beide testen.

8.2.12 SnowStorm Test

- **Beschreibung**

In Proto-Version wurde auch ein deterministisches Drehbuch dafür angezeigt, um zu testen und demonstrieren.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Es testet die Wirkung von Schneesturm.

- **Eingänge**

PrintHeimMap

PrintSnowTileMap//Beobachtung

PrintPlayer//Player muss auf einem Feld sein wo Schneesturm kommt
(deterministisch!)

PrintSnowTileMap//nach einer Rund,
wenn ein Schneesturm kommen muss.

PrintPlayer//Falls kein Iglu oder Zelt:
Temperatur--, sonst bleibt.

PrintHeimMap

- **Erwartete Ausgänge**

HeimMap//Beobachtung

SnowTileMap// Beobachtung

Player Eigenschaften//Temperatur ist interessant

SnowTileMap// nach SnowStorm muss in einige Felden snow++

Player Eigenschaften// Falls kein Iglu oder Zelt: temperatur--, sonst bleibt.

HeimMap//Zelt, Iglos müssen zerstört werden, wo Storm war

8.2.13 PolarBear Test

- **Beschreibung**

In proto-Version gibt es eine deterministische Version, das testen wir auch hier.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Wenn ein PolarBear und ein Spieler treffen, das Spiel beendet. Es ist in endLastRound von RoundController gemacht.

- **Eingänge**

Step dir<polarbear nextPos dir>

- **Erwartete Ausgänge**

“Step successful, stepped from (x,y) to (x,y)”//polarBear ist deterministisch, wir wissen wohin es geht
 „lose <cause>”// cause: PolarBear

8.2.14 JUnit test: State check(Item)

- **Beschreibung**

Wir nehmen ein Item, rufen wir die Methoden, die zu Zustandsänderungen führen (thrownDown, pickedUp, diggedUp), und testen, ob der Zustand der Gegenstand (ItemState) des Items gut ist nach diesen Methoden.

- **Getestete Funktionalitäten, erwartbare Fehlern**

Zustandsänderungen sind hier getestet, wir beobachten itemState Membervariable von einem Item. Mögliche Fehler kann zB.: falsche Zustandsänderung sein.

- **Eingänge**

In JUnit wir testen thrownDown, pickedUp, diggedUp Methoden von einem Item.

- **Erwartete Ausgänge**

Nach thrownDown(): von InHand :itemState == ThrownDown

Nach diggedUp(): von frozen :itemState == ThrownDown

Nach pickedUp(): von ThrownDown :itemState == InHand

8.2.15 JUnit test: PlayerContainer Test

- **Beschreibung**

Unit-test von getPlayer Method in PlayerContainer Klasse. PlayerContainer ist deterministisch initialisiert.

- **Getestete Funktionalitäten, erwartbaren Fehlern**

Hier vergleichen wir das Rückgabewert mit players ArrayList Elementen.

- **Eingänge**

Wir geben verschiedene Nummern als getPlayer Parameter, und vergleichen wir das Rückgabewert mit players ArrayList.

- **Erwartete Ausgänge**

Gleichheit ist für gute Ergebnis.

8.3 Spezifikation der Test unterstützenden Hilfs- und Übersetzungsprogramme

Wir werden JUnit benutzen um Unit-testen zu machen. JUnit ist ein open source Java Testing Framework. Es hat built-in support in IntelliJ IDEA. Mit JUnit können wir testen das erwartete Ergebnis von einem Method mit konkreten Parametern gut ist, wir können auch erwartete Ausnahmen (Exceptions) prüfen.

11. Spezifikation der graphischen Oberfläche

DeutschOverflow

Supervisor:
Kovács Márton

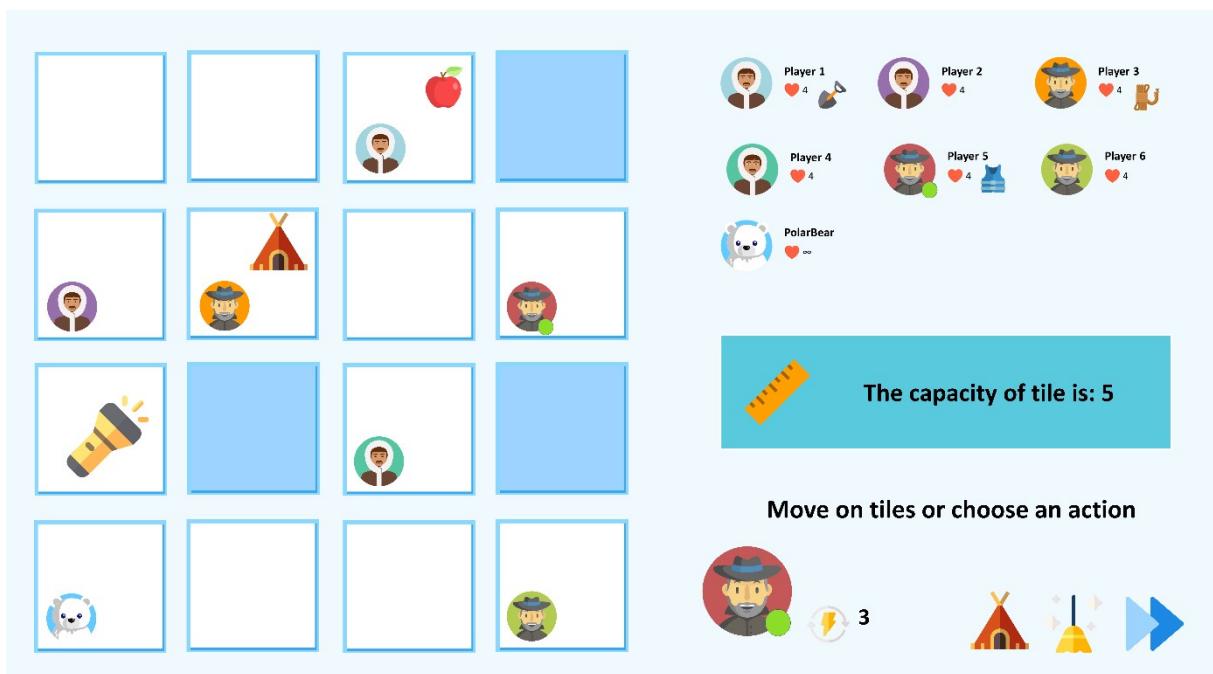
Members:

Ádám Zsófia	SOSK6A	adamzsofi.mail@gmail.com
Hedrich Ádám	H9HFFV	hedrichadam09@gmail.com
Pintér Balázs	ZGY18G	pinterbalazs21@gmail.com
Fucskár Patrícia	XKYAOO	fucskar.patricia@gmail.com
Tassi Timián	MYY53U	timian.tassi@gmail.com

5. Mai 2020

11. Spezifikation der graphischen Oberfläche

11.1 Die graphische Interface



11.2 Architektur der Benutzeroberfläche

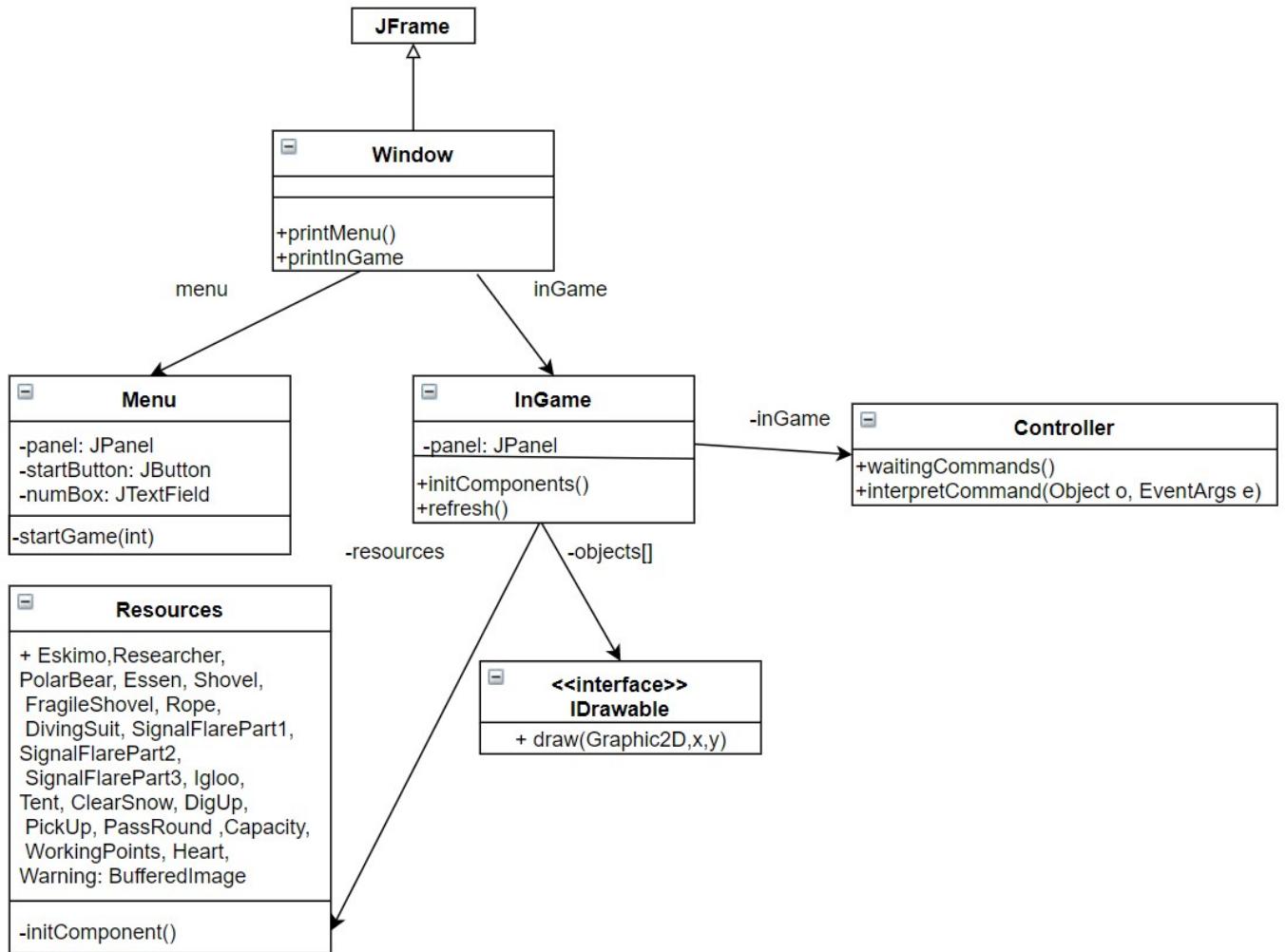
11.2.1 Prinzip und Arbeitsweise der Benutzeroberfläche

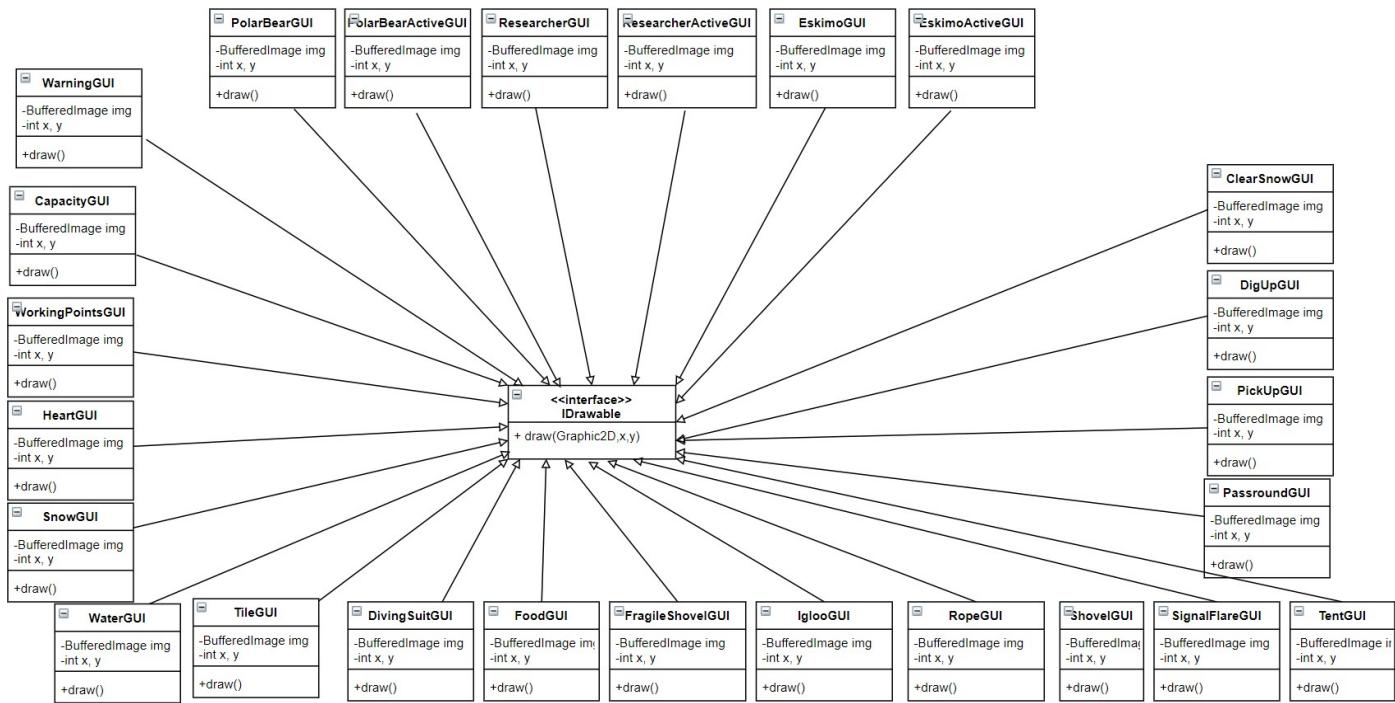
Die graphische Oberfläche wird nach MVC (Model-View-Controller) modell implementiert. Alle auf Bildschirm sichtbare Objekte haben ein graphische Klasse([classname]GUI) auch, was implementiert IDrawable Schnittstelle. Wir haben eine View Klasse, was die graphische Objekte von Spiel speichert und wenn etwas sich verändert, dann erfrischt es das Bildschirm. Unsere GUI ist push-modell basiert, wenn eine Änderung in Modell kommt (von Controller), dann Game Klasse nachrichtet das View Klasse. View Klasse kennt PositionLUT Klasse, deshalb kann es jedes Objekt in gute position platzieren.

Am Anfang des Spiels kann man im Menu die Spieleranzahl einstellen(3-6) und das Spiel mit einer Taste starten.

Dann erschien das GamePanel, wo man das Spielscene sehen kann und man kann interaktiv spielen.

11.2.2 Struktur der Oberfläche





11.3 Aufzählung der graphischen Objekte

ALLGEMEINE KLASSEN

11.3.1 Resources

Hier speichern wir die Ikonen, damit nur einmal laden müssen

- **Eskimo**
- **Researcher**
- **PolarBear**
- **Essen**
- **Shovel**
- **FragileShovel**
- **Rope**
- **DivingSuit**
- **SignalFlarePart1**
- **SignalFlarePart2**
- **SignalFlarePart3**
- **Igloo**
- **Tent**
- **ClearSnow**
- **DigUp**
- **PickUp**
- **PassRound**
- **Capacity**
- **WorkingPoints**
- **Heart**
- **Warning**

- **-void initComponents():** Lädt die Objekte am Anfang ein. (Der Konstruktor ruft diese Methode.)

11.3.2 Menu

- **Verantwortung**

Diese Klasse zeichnet und handelt das Menu.

- **Attributen**

- **- JPanel panel:** Die Panel des Menus.
- **- JButton startButton**
- **- JTextField numBox**

- **Methoden**

- **-startGame(int I):** Das Spiel wird gestartet. Menu panel verschwindet, inGame panel kommt.

11.3.3 InGame

- **Verantwortung**

Diese Klasse zeichnet und handelt das Spiel.

- **Attributen**

- **- JPanel panel:** Die Panel des “inGame”s.
- **- View view:** Ein View Attribut, es zeichnet die Objecte aus.

- **Methoden**

- **+refresh():** erfrischt das Bildschirm, mit aktuelle Positionen und Objekten von PositionLUT, es benutzt die Bilden von Resources Klasse
- **+initComponents():** Die Komponente werden initialisiert.

11.3.4 Controller

- **Verantwortung**

Wartet auf Tastendrucke und Mauseklick, kontrolliert das Game Klasse.

- **Methoden**

- **waitingCommands():** game loop, es wartet auf Ereignisse, und ruft methoden von Game klasse.
- **interpretCommand(Object o, EventArgs e):** interpretiert Kommand

11.3.5 Window

- **Verantwortung**

Diese Klasse zeichnet das Menu und das Spiel.

- **Attributen**

- - **Menu menu:** In der Window befindet sich ein Menu.
- - **InGame inGame:** In der Window befindet sich ein inGame Instanz.

- **Methoden**

- - **printMenu():** am Anfang zeichnet es das Menu
- - **printInGame():** ruft refresh() method von inGame

11.3.6 IDrawable

- **Verantwortung**

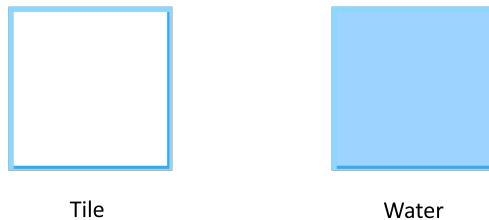
Schnittstelle um die Elemente auszeichnen. Die verschiedene GUI der Objecten verwirklicht diese Schnittstelle.

- **Methoden**

- + **draw (Graphics2D g, int x, int y)** Objekte anzuzeichen. Wir müssen eine Position und ein Graphics2D Objekt eingeben, um ein Objekt zuzeichnen.

PLATTE UND WASSER

Tile & Water



11.3.7 TileGUI

- **Verantwortung**

Diese Klasse representiert eine Platte. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.8 WaterGUI

- **Verantwortung**

Diese Klasse representiert das Wasser (SnowyTile oder UnsatbleTile nach einem Treten). Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

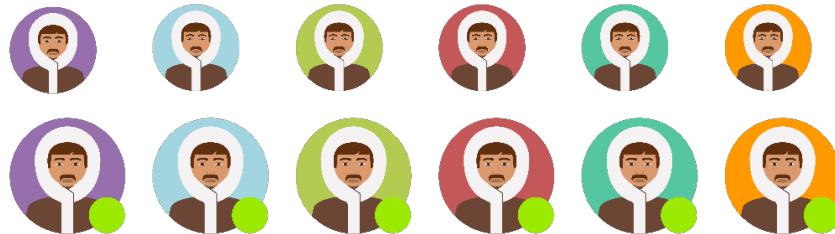
- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

CHARAKTEREN

Characters

Eskimos



Researchers



PolarBear



Hinweis! Konzeptuell möchten wir für jeden Character (im Spiel gibt es 3-6 Spieler) eine dedizierte Farbe geben, was in dem Bild kodiert ist.

Also wir haben im GUI Pack für jeden CharacterTyp [hier ist Eskimo und Researcher gemeint] 2x6 Bilder (aktiv und nicht-aktiv).

Diese wird in Laufzeit zum Spieler geordnet, ob es ein Eskimo oder Reseacher ist. ()

11.3.9 PolarBearGUI

- **Verantwortung**

Diese Klasse representiert ein Eisbär. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.10 PolarBearActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung des Eisbärs. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.11 ResearcherGUI

- **Verantwortung**

Diese Klasse representiert ein Forscher. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.12 ResearcherActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung eines Forschers. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.13 EskimoGUI

- **Verantwortung**

Diese Klasse representiert ein Eskimo. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.14 EskimoActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung eines Eskimos. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

GEGENSTÄNDE**Items**

DivingSuit



Food



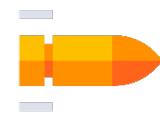
Shovel



FragileShovel



Rope

SignalFlare
Part1SignalFlare
Part2SignalFlare
Part3

Igloo



Tent

11.3.15 DivingSuitGUI

- **Verantwortung**

Diese Klasse representiert ein Taucheranzug. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.16 FoodGUI

- **Verantwortung**

Diese Klasse representiert ein Essen. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.17 ShovelGUI

- **Verantwortung**

Diese Klasse representiert eine Schaufel. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.18 FragileShovelGUI

- **Verantwortung**

Diese Klasse representiert eine zerbrechliche Schaufel. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

11.3.19 RopeGUI

- **Verantwortung**

Diese Klasse representiert ein Seil. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

11.3.20 SignalFlarePart1GUI

- **Verantwortung**

Diese Klasse representiert das erste Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

11.3.21 SignalFlarePart2GUI

- **Verantwortung**

Diese Klasse representiert das zweite Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.22 SignalFlarePart3GUI

- **Verantwortung**

Diese Klasse representiert das dritte Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.23 IglooGUI

- **Verantwortung**

Diese Klasse representiert ein Iglu. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.24 TentGUI

- **Verantwortung**

Diese Klasse representiert ein Zelt. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

AKTIONEN

Actions



ClearSnow



DigUp



PickUp



PassRound

Bemerkung! Gegenstände können auch zur Veranschaulichung der Aktionen verwendet werden. Zum Beispiel: wenn der Benutzer FragileShovel in seinem Hand hat, dann erscheint bei der ActionListe statt ClearSnowGUI ein FragileShovelGUI.

11.3.25 ClearSnowGUI

- **Verantwortung**

Diese Klasse representiert das Räumen des Schnees. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
- - **int x**: Position nach der x-Achse
- - **int y**: Position nach der y-Achse

11.3.26 DigUpGUI

- **Verantwortung**

Diese Klasse representiert das Ausgraben des Gegenstands. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.27 PickUpGUI

- **Verantwortung**

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.28 PassRoundGUI

- **Verantwortung**

Diese Klasse representiert das Passen des Spielers. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

ANDERE ELEMENTE

Other Icons



Capacity



WorkingPoints



Heart



PassRound



Warning

11.3.29 CapacityGUI

- **Verantwortung**

Diese Klasse representiert die Kapazität einer Platte. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.30 WorkingPointsGUI

- **Verantwortung**

Diese Klasse representiert die Arbeitspunkte eines Spielers. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.31 HeartGUI

- **Verantwortung**

Diese Klasse representiert das Leben eines Charakters. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.32 SnowGUI

- **Verantwortung**

Diese Klasse representiert der Schnee. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

11.3.33 WarningGUI

- **Verantwortung**

Diese Klasse representiert eine Warnung. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

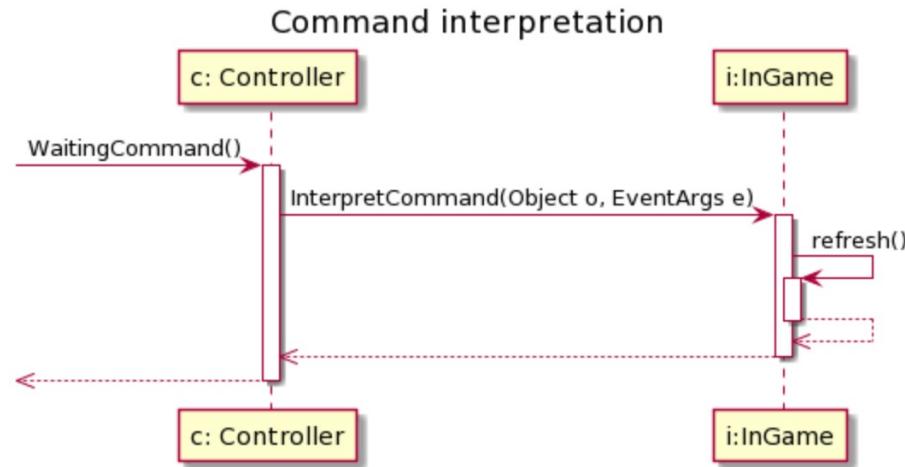
- IDrawable

- **Attributen**

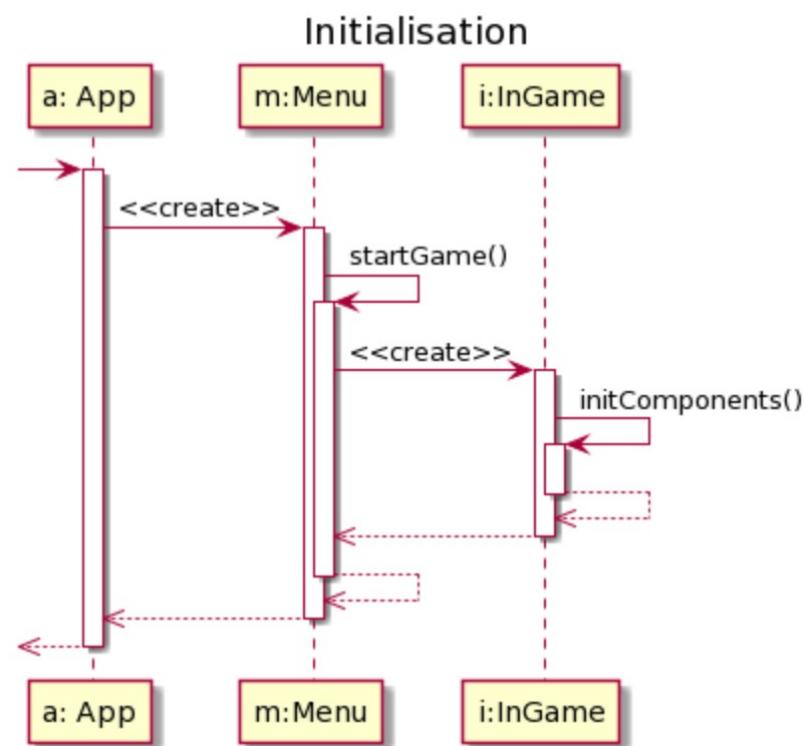
- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

11.4 Beziehung mit dem Anwender-System

11.4.1 Command interpretation



11.4.2 Initialisation



11. Spezifikation der graphischen Oberfläche

DeutschOverflow

Supervisor:
Kovács Márton

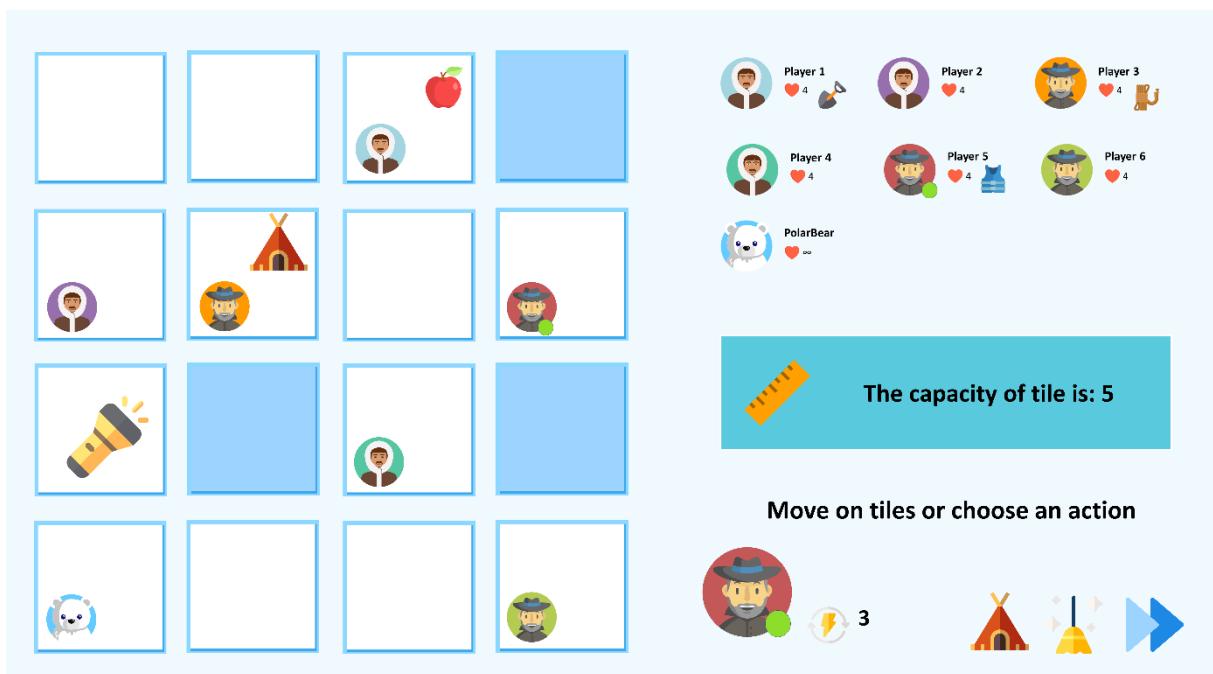
Members:

Ádám Zsófia	SOSK6A	adamzssofi.mail@gmail.com
Hedrich Ádám	H9HFFV	hedrichadam09@gmail.com
Pintér Balázs	ZGY18G	pinterbalazs21@gmail.com
Fucskár Patrícia	XKYAOO	fucskar.patricia@gmail.com
Tassi Timián	MYY53U	timian.tassi@gmail.com

5. Mai 2020

11. Spezifikation der graphischen Oberfläche

11.1 Die graphische Interface



11.2 Architektur der Benutzeroberfläche

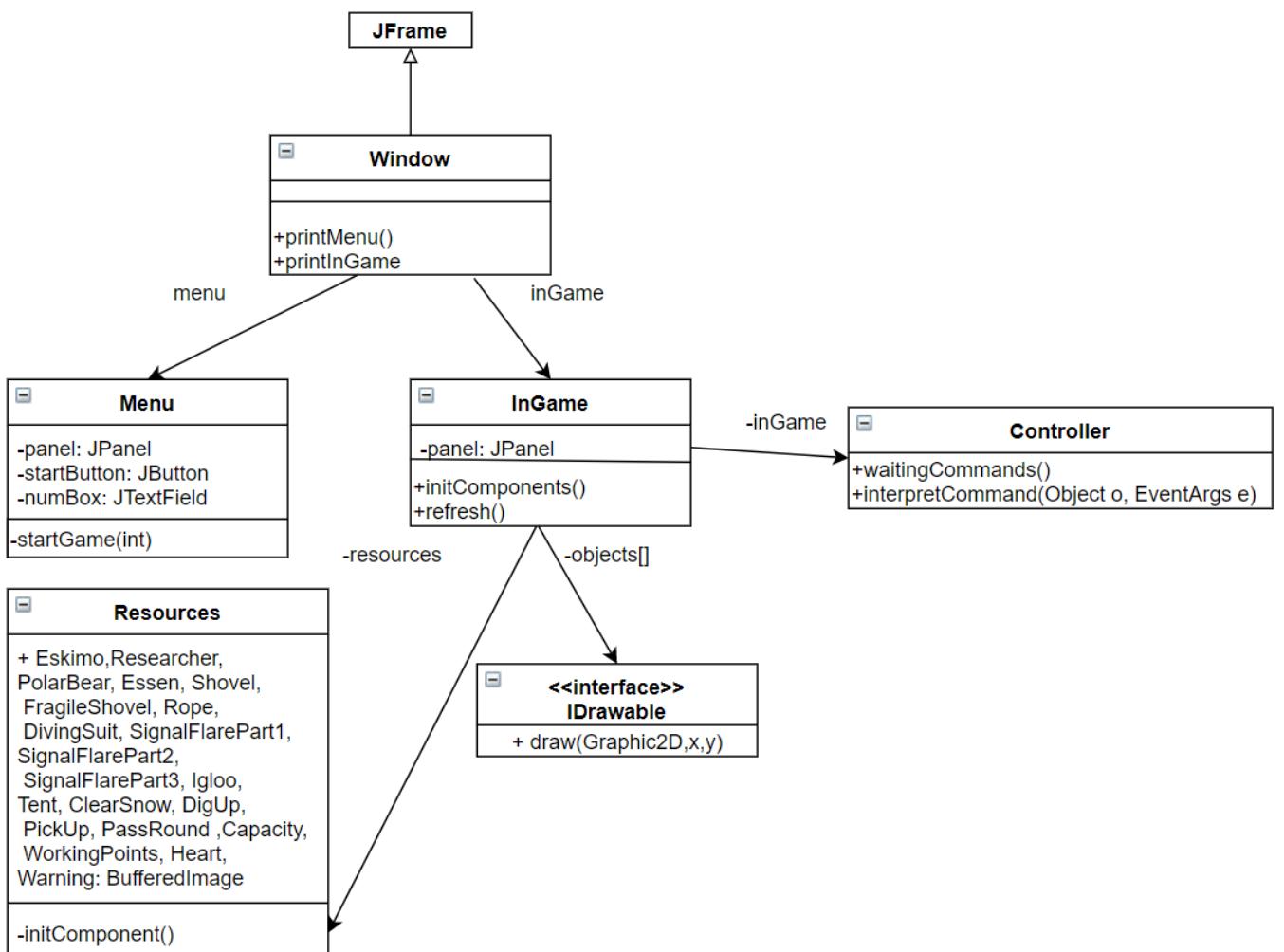
11.2.1 Prinzip und Arbeitsweise der Benutzeroberfläche

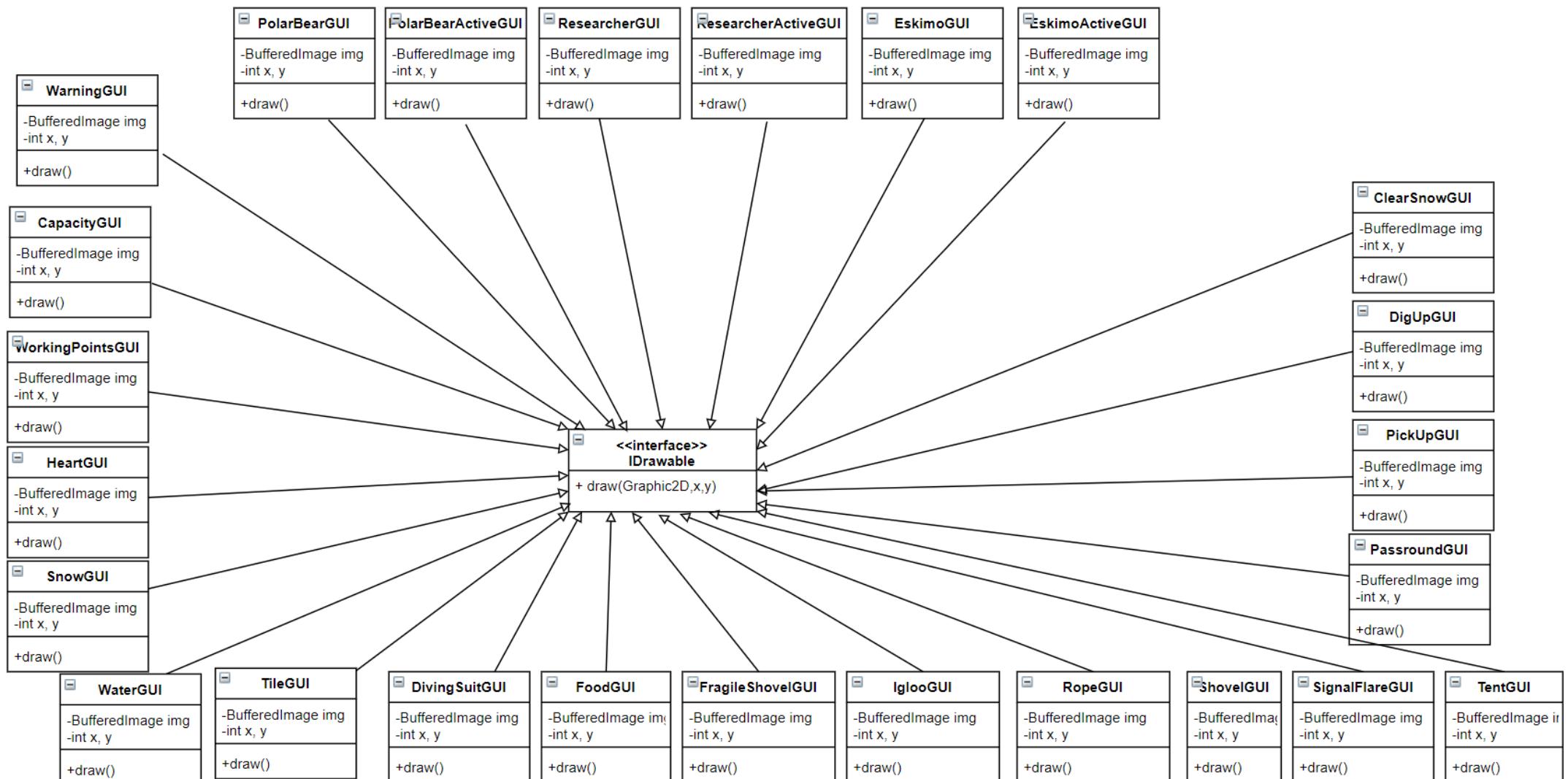
Die graphische Oberfläche wird nach MVC (Model-View-Controller) modell implementiert. Alle auf Bildschirm sichtbare Objekte haben eine graphische Klasse([classname]GUI) auch, was implementiert IDrawable Schnittstelle. Wir haben eine View Klasse, was die graphische Objekte von Spiel speichert und wenn etwas sich verändert, dann erfrischt es das Bildschirm. Unsere GUI ist push-modell basiert, wenn eine Änderung in Modell kommt (von Controller), dann Game Klasse nachrichtet das View Klasse. View Klasse kennt PositionLUT Klasse, deshalb kann es jedes Objekt in gute position platzieren.

Am Anfang des Spiels kann man im Menu die Spieleranzahl einstellen(3-6) und das Spiel mit einer Taste starten.

Dann erschien das GamePanel, wo man die Spielscene sehen kann und man kann interaktiv spielen.

11.2.2 Struktur der Oberfläche





11.3 Aufzählung der graphischen Objekte

ALLGEMEINE KLASSEN

11.3.1 Resources

Hier speichern wir die Ikonen, damit nur einmal laden müssen

- **Eskimo**
- **Researcher**
- **PolarBear**
- **Essen**
- **Shovel**
- **FragileShovel**
- **Rope**
- **DivingSuit**
- **SignalFlarePart1**
- **SignalFlarePart2**
- **SignalFlarePart3**
- **Igloo**
- **Tent**
- **ClearSnow**
- **DigUp**
- **PickUp**
- **PassRound**
- **Capacity**
- **WorkingPoints**
- **Heart**
- **Warning**
- **-void initComponents():** Lädt die Objekte am Anfang ein. (Der Konstruktor ruft diese Methode.)

11.3.2 Menu

- **Verantwortung**

Diese Klasse zeichnet und handelt das Menu.

- **Attributen**

- **- JPanel panel:** Die Panel des Menus.
- **- JButton startButton**
- **- JTextField numBox**

- **Methoden**

- **-startGame(int I):** Das Spiel wird gestartet. Menu panel verschwindet, inGame panel kommt.

11.3.3 InGame

- **Verantwortung**

Diese Klasse zeichnet und handelt das Spiel.

- **Attributen**

- - **JPanel panel:** Die Panel des “inGame”s.
- - **View view:** Ein View Attribut, es zeichnet die Objecte aus.

- **Methoden**

- +**refresh()**: erfrischt das Bildschirm, mit aktuelle Positionen und Objekten von PositionLUT, es benutzt die Bilden von Resources Klasse
- +**initComponents()**: Die Komponente werden initialisiert.

11.3.4 Controller

- **Verantwortung**

Wartet auf Tastendrucke und Mauseklick, kontrolliert das Game Klasse.

- **Methoden**

- **waitingCommands()**: game loop, es wartet auf Ereignisse, und ruft methoden von Game klasse.
- **interpretCommand(Object o, EventArgs e)**: interpretiert Kommand

11.3.5 Window

- **Verantwortung**

Diese Klasse zeichnet das Menu und das Spiel.

- **Attributen**

- - **Menu menu:** In der Window befindet sich ein Menu.
- - **InGame inGame:** In der Window befindet sich ein inGame Instanz.

- **Methoden**

- -**printMenu()**: am Anfang zeichnet es das Menu
- -**printInGame()**: ruft refresh() method von inGame

11.3.6 IDrawable

- **Verantwortung**

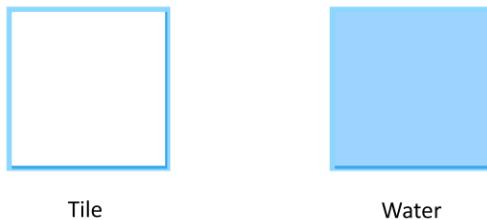
Schnittstelle um die Elemente auszeichnen. Die verschiedene GUI der Objecten verwirklicht diese Schnittstelle.

- **Methoden**

- + **draw (Graphics2D g, int x, int y)** Objekte anzuziehen. Wir müssen eine Position und ein Graphics2D Objekt eingeben, um ein Objekt zu zeichnen.

PLATTE UND WASSER

Tile & Water



11.3.7 TileGUI

- **Verantwortung**

Diese Klasse repräsentiert eine Platte. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.8 WaterGUI

- **Verantwortung**

Diese Klasse repräsentiert das Wasser (SnowyTile oder UnsatbleTile nach einem Treten). Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

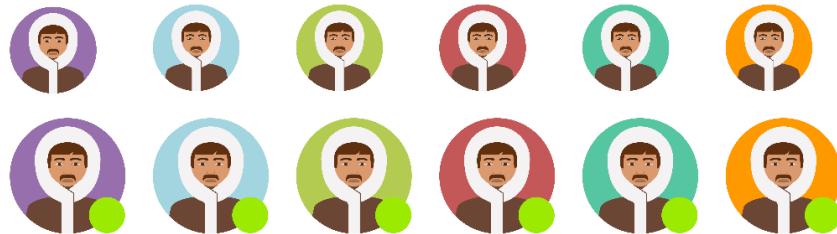
- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

CHARAKTEREN

Characters

Eskimos



Researchers



PolarBear



Hinweis! Konzeptuell möchten wir für jeden Character (im Spiel gibt es 3-6 Spieler) eine dedizierte Farbe geben, was in dem Bild kodiert ist.
Also wir haben im GUI Pack für jeden CharacterTyp [hier ist Eskimo und Researcher gemeint] 2x6 Bilder (aktiv und nicht-aktiv).
Diese wird in Laufzeit zum Spieler geordnet, ob es ein Eskimo oder Reseacher ist. ()

11.3.9 PolarBearGUI

- **Verantwortung**

Diese Klasse representiert ein Eisbär. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.10 PolarBearActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung des Eisbärs. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.11 ResearcherGUI

- **Verantwortung**

Diese Klasse representiert ein Forscher. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.12 ResearcherActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung eines Forschers. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.13 EskimoGUI

- **Verantwortung**

Diese Klasse representiert ein Eskimo. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.14 EskimoActiveGUI

- **Verantwortung**

Diese Klasse representiert die Bewegung eines Eskimos. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

GEGENSTÄNDE

Items



DivingSuit



Food



Shovel



FragileShovel



Rope

SignalFlare
Part1SignalFlare
Part2SignalFlare
Part3

Igloo



Tent

11.3.15 DivingSuitGUI

- **Verantwortung**

Diese Klasse representiert ein Taucheranzug. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.16 FoodGUI

- **Verantwortung**

Diese Klasse representiert ein Essen. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.17 ShovelGUI

- **Verantwortung**

Diese Klasse representiert eine Schaufel. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
- - **int x:** Position nach der x-Achse
- - **int y:** Position nach der y-Achse

11.3.18 FragileShovelGUI

- **Verantwortung**

Diese Klasse representiert eine zerbrechliche Schaufel. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.19 RopeGUI

- **Verantwortung**

Diese Klasse representiert ein Seil. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.20 SignalFlarePart1GUI

- **Verantwortung**

Diese Klasse representiert das erste Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.21 SignalFlarePart2GUI

- **Verantwortung**

Diese Klasse representiert das zweite Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.22 SignalFlarePart3GUI

- **Verantwortung**

Diese Klasse representiert das dritte Teil des Signalfackels. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.23 IglooGUI

- **Verantwortung**

Diese Klasse representiert ein Iglu. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.24 TentGUI

- **Verantwortung**

Diese Klasse representiert ein Zelt. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
 - - **int x**: Position nach der x-Achse
 - - **int y**: Position nach der y-Achse

AKTIONEN

Actions



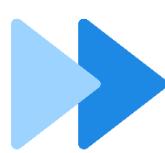
ClearSnow



DigUp



PickUp



PassRound

Bemerkung! Gegenstände können auch zur Veranschaulichung der Aktionen verwendet werden. Zum Beispiel: wenn der Benutzer FragileShovel in seinem Hand hat, dann erscheint bei der ActionListe statt ClearSnowGUI ein FragileShovelGUI.

11.3.25 ClearSnowGUI

- **Verantwortung**

Diese Klasse representiert das Räumen des Schnees. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage**: texture
- - **int x**: Position nach der x-Achse
- - **int y**: Position nach der y-Achse

11.3.26 DigUpGUI

- **Verantwortung**

Diese Klasse representiert das Ausgraben des Gegenstands. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.27 PickUpGUI

- **Verantwortung**

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.28 PassRoundGUI

- **Verantwortung**

Diese Klasse representiert das Passen des Spielers. Wir können es mit der “*draw*” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

ANDERE ELEMENTE

Other Icons



Capacity



WorkingPoints



Heart



Snow



Warning

11.3.29 CapacityGUI

- **Verantwortung**

Diese Klasse representiert die Kapazität einer Platte. Wir können es mit der “draw” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.30 WorkingPointsGUI

- **Verantwortung**

Diese Klasse representiert die Arbeitspunkte eines Spielers. Wir können es mit der “draw” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.31 HeartGUI

- **Verantwortung**

Diese Klasse representiert das Leben eines Charakters. Wir können es mit der “draw” Funktion (IDrawable Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.32 SnowGUI

- **Verantwortung**

Diese Klasse representiert der Schnee. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

- IDrawable

- **Attributen**

- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.3.33 WarningGUI

- **Verantwortung**

Diese Klasse representiert eine Warnung. Wir können es mit der “*draw*” Funktion (IDrawAble Schnittstelle definiert es) zeichnen.

- **Interfacen**

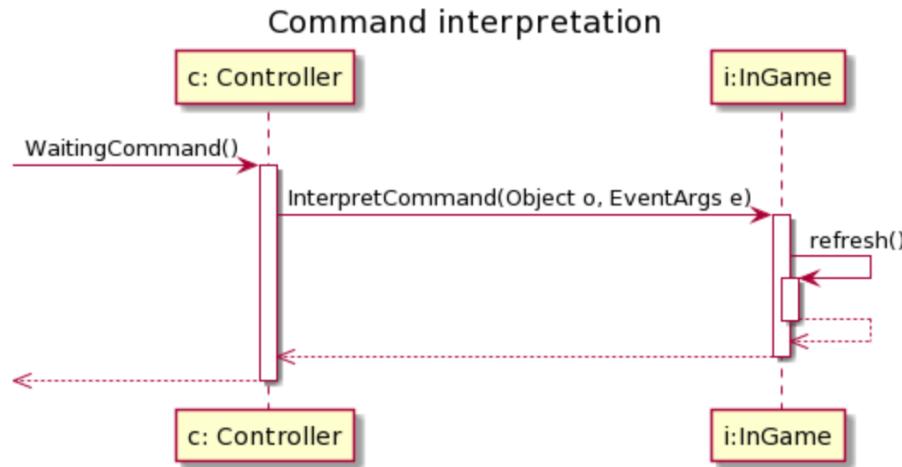
- IDrawable

- **Attributen**

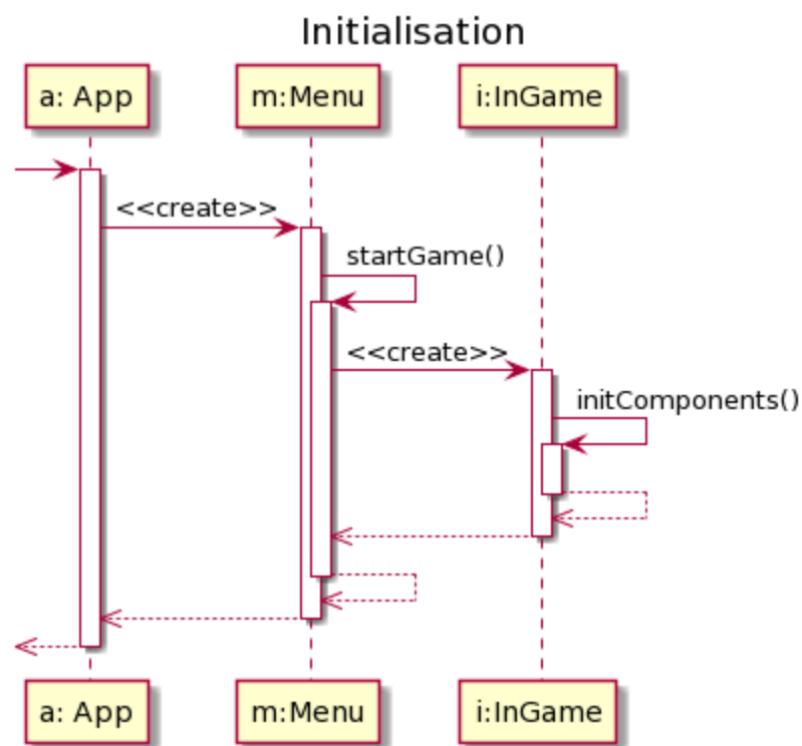
- - **BufferedImage:** texture
 - - **int x:** Position nach der x-Achse
 - - **int y:** Position nach der y-Achse

11.4 Beziehung mit dem Anwender-System

11.4.1 Command interpretation



11.4.2 Initialisation



13. Eingabe der graphischen Oberfläche

DeutschOverflow

Supervisor:
Kovács Márton

Members:

Ádám Zsófia	SOSK6A	adamzsofi.mail@gmail.com
Hedrich Ádám	H9HFFV	hedrichadam09@gmail.com
Pintér Balázs	ZGY18G	pinterbalazs21@gmail.com
Fucskár Patrícia	XKYAOO	fucskar.patricia@gmail.com
Tassi Timián	MYY53U	timian.tassi@gmail.com

19. Mai 2020

13. Eingabe der graphischen Oberfläche

13.1 Übersetzungs- und Ausführungsanleitung

13.1.1 Dateiliste

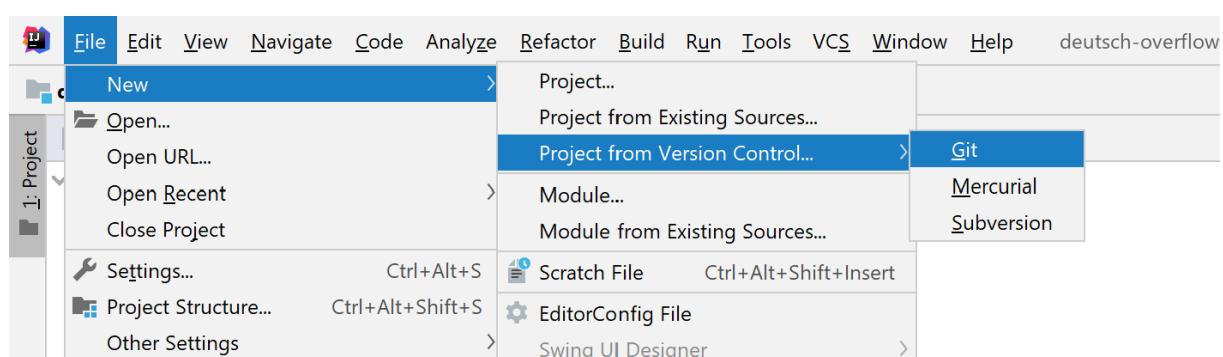
Siehe "dateiliste.xlsx" in Verzeichnis bugyi (*es war zu groß, um eingebettet zu sein*).

13.1.2 – 6.1.3 Übersetzung & Ausführung

Man kann das Projekt mit JDK 12 oder neuer laufen lassen. Für Entwicklung haben wir IntelliJ IDE benutzt, dort soll man erst das Projekt bilden, dann kann man es mit Start Button starten.

Die Übersetzung wird mithilfe von Github durch IntelliJ IDEA durchgeführt:

1. Öffnen Sie die Anwendung: IntelliJ IDEA
2. Klicken Sie auf *File>New>Project with Version Control>Git*

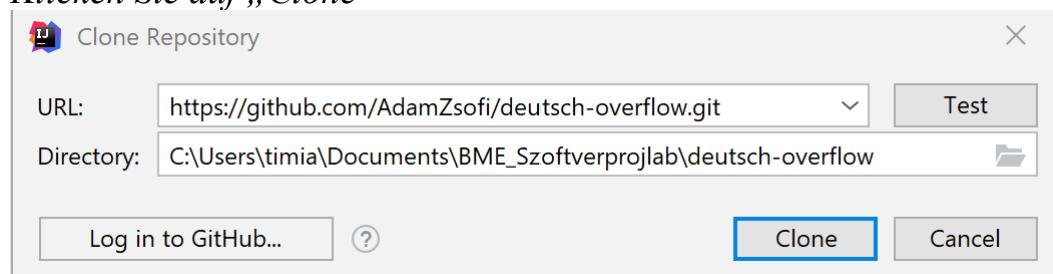


1. Abbildung: Öffnen des Projektes

3. Kopieren Sie den folgenden Link ins Fenster:

<https://github.com/AdamZsofi/deutsch-overflow.git>

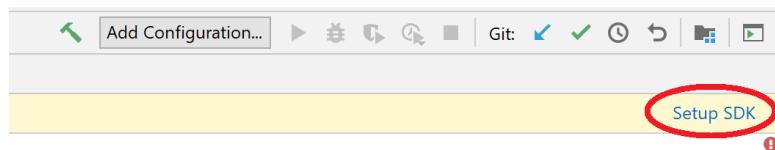
4. Klicken Sie auf „Clone“



2. Abbildung: Öffnen des Projektes

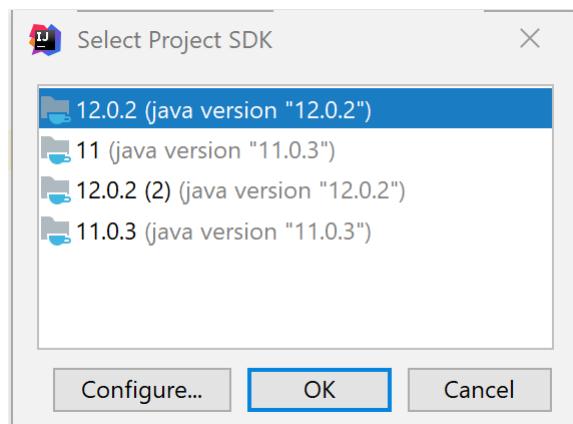
5. Navigieren Sie auf der linken Seite des Fensters im Baumstruktur *deutsch-overflow>src>SkeletonMain*

6. Wenn die Java SDK nicht eingestellt ist, dann klicken Sie an Setup SDK



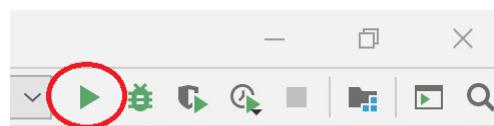
3. Abbildung

7. Stellen Sie die neueste Java Umgebung ein, dann klicken Sie auf „OK“



4. Abbildung

8. Klicken Sie auf „Ausführen“(Play)



5. Abbildung

Hinweis!

Später können Sie das Programm (.exe) auch mit Java Runtime Environment ausführen
(von der „bin“ Ordner des generierten Projektes)

14. Zusammenfassung

- Anzahl von Kodezeilen in den aufgeladenen Programmen**

(Info von IntelliJ Statistics Plugin)

Phase	Anzahl von Kodezeilen
Skeleton	1339
Prototyp	2993
Graphische Version	3750
Insgesamt	8082 (oder 3750!)

14.1 • Zusammenfassung des Projekts

14.1.1 Was haben sie von diesem Projekt gelernt, konkret und im Allgemeinen?

In dem Rahmen der Universität hatten wir vorher nie ein Semester langes Projekt ausgeführt, also die Charakteristiken eines solches Projekt waren richtig neu. Wir hatten auch keine großen Gruppenprojekte vorher; Im Allgemeinen mussten wir über das viel neues lernen - über Kommunikation und verschiedene Mittel einer Gruppenprojekt (Trello, Github, Benutzung von fortgeschrittenen Mitteln von einer IDE /IntelliJ/).

14.1.2 Was war die Schwierigste und die Leichteste?

In den zweiten Half der Semester könnten wir uns persönlich nicht treffen, deswegen könnten wir das Projekt nur Online besprechen, das war eine Schwierigkeit. Es kam mehrmals vor, dass wir etwas geplant haben, aber bei der Implementation sahen wir, dass es so nicht implementierbar ist, und mussten eine andere Lösung finden. Nach der Änderung der Anforderungen war es leicht, die Änderungen in unseren Skeleton einzubauen, weil es leicht erweiterbar war.

14.1.3 War die Zeit und die Punkte für die Aufgaben genug und nicht zu viel?

Wir hatten immer genug Zeit für die Aufgaben, aber manchmal hatten wir es zu spät angefangen, wenn wir z.B. andere Hausaufgaben beenden mussten, deswegen mussten wir uns beeilen. Punkten sind ungefähr proportional zu der Schwierigkeit den Aufgaben.

14.1.4 Falls nicht, dann wo hat es Schwierigkeiten verursacht?

Keine Probleme hier, siehe 14.1.3

14.1.5 Haben sie Vorschläge für Änderungen?

Wenigere Dokumentation!

Obwohl bei einer Firma müssen die Entwickler mehrere Dokumentationen herstellen, dort ist es nicht sinnlos – bei uns hat es viele Zeit gebraucht und hat es nicht zu dem Projekt gegeben, Planung war noch schwieriger.

14.1.6 Haben sie Vorschläge für zukünftliche Projektaufgaben?

keine

14.1.7 Andere Kritiken und Vorschläge

keine