7. Konzeption von Prototyp

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

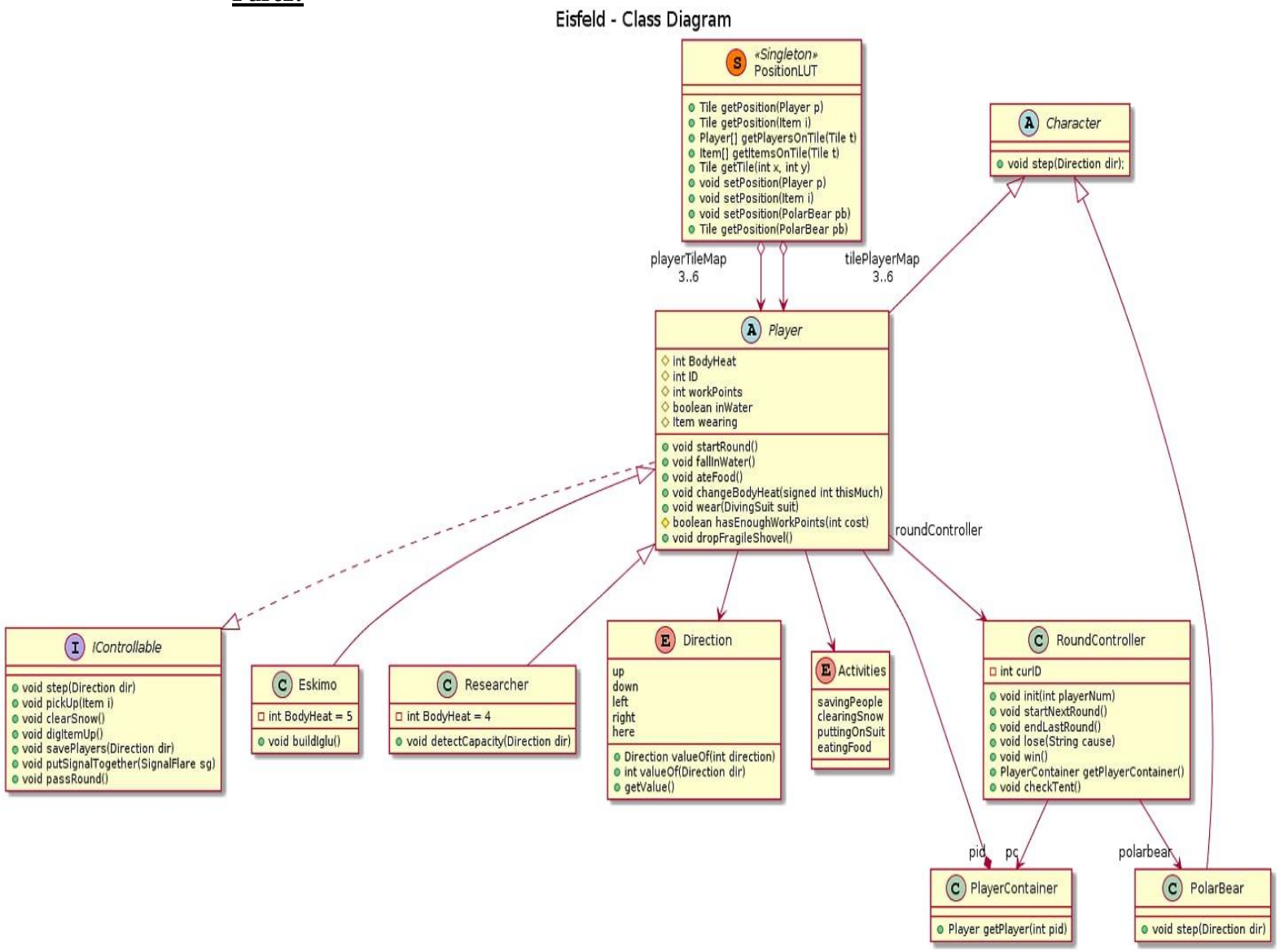
7. April 2020

7. Konzeption von Prototyp

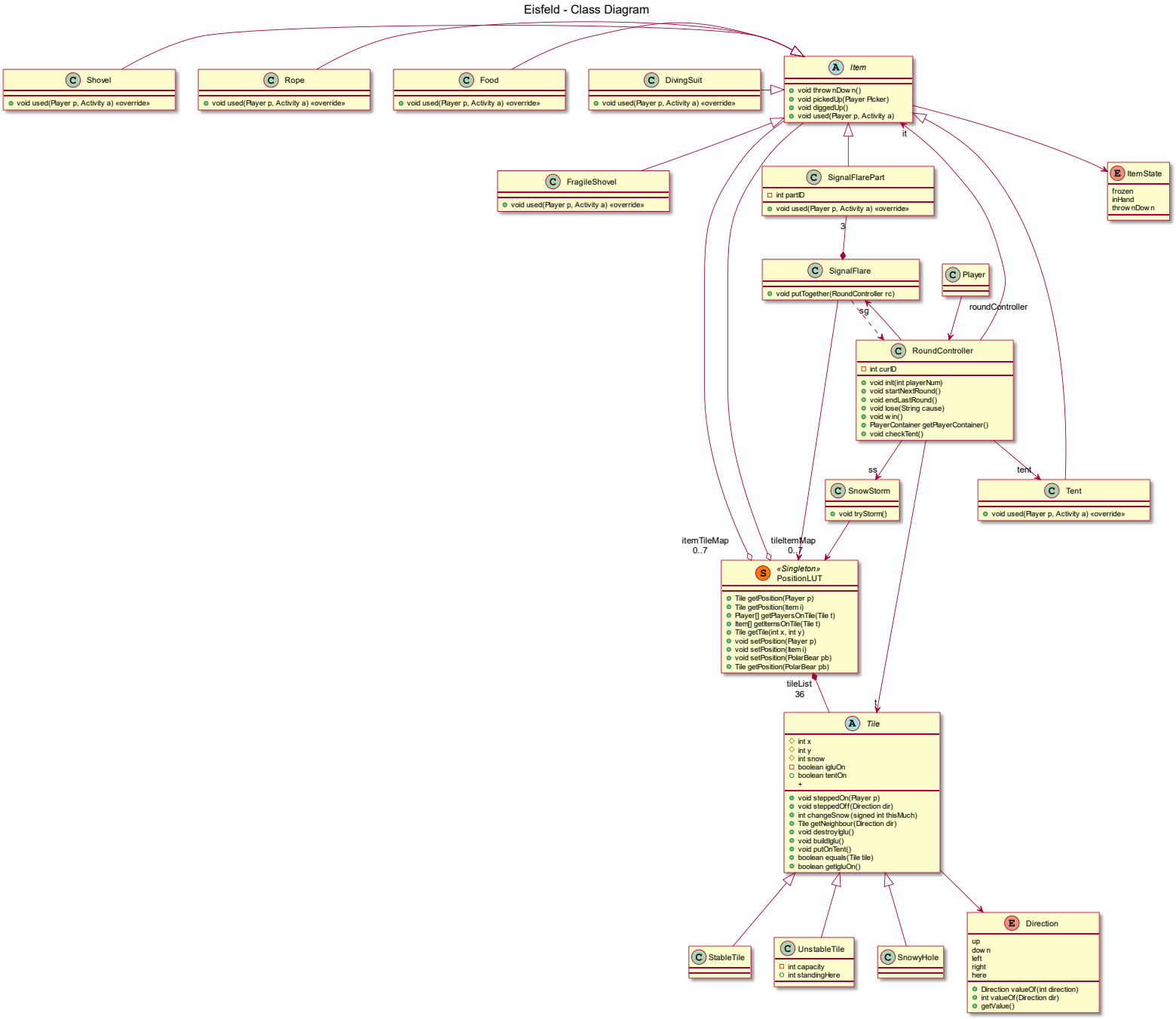
7.0 Wirkung der Änderung auf das Modell

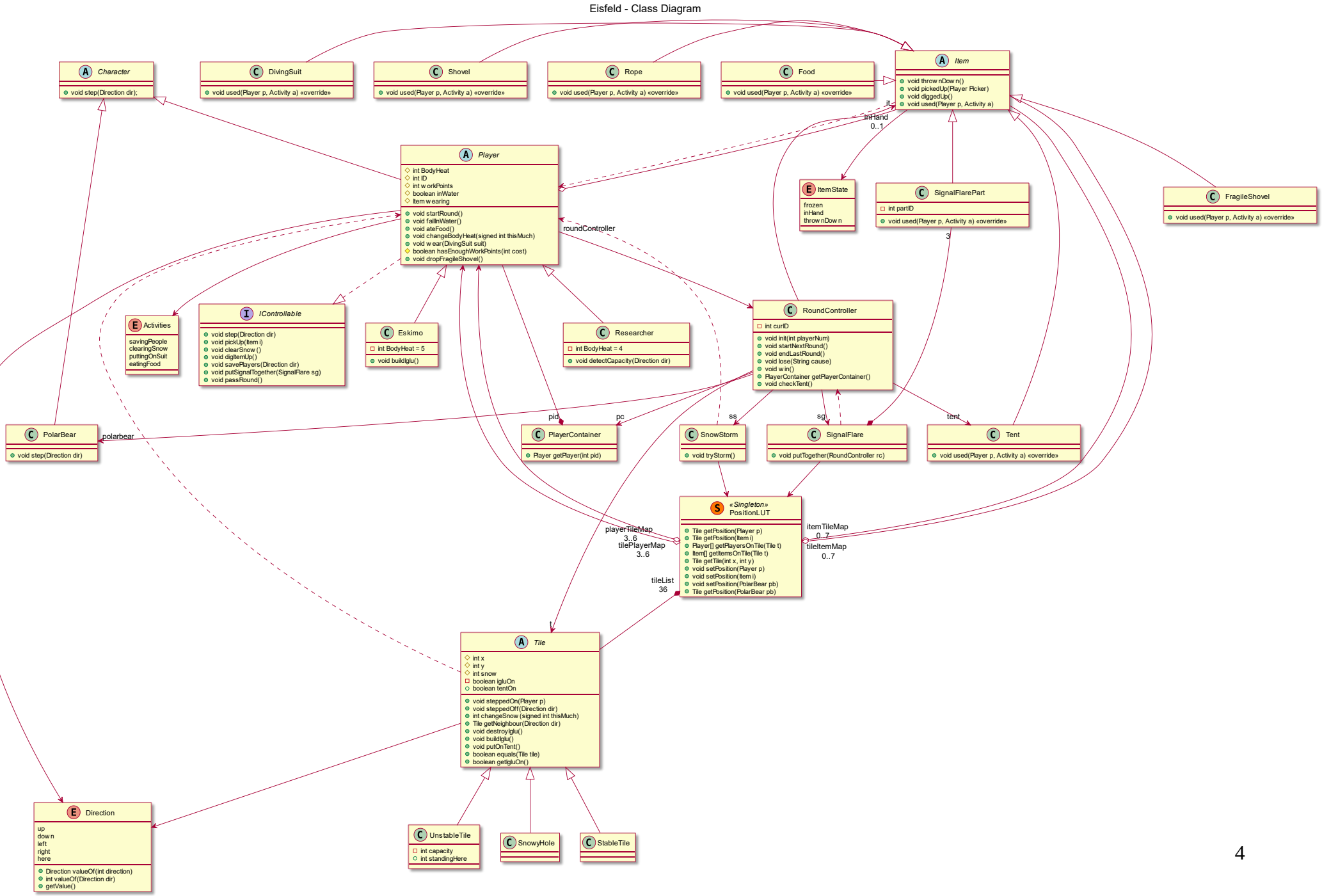
7.0.1 Verändertes Klassendiagramm

- **Part1:**



- **Part2:**


Eisfeld - Class Diagram
Shovel
void used(Player p, Activity a) «override»
Rope
void used(Player p, Activity a) «override»
Food
void used(Player p, Activity a) «override»
DivingSuit
void used(Player p, Activity a) «override»
Item
void throwNDown()
void pickedUp(Player Picker)
void diggedUp()
void used(Player p, Activity a)
it
ItemState
frozen
inHand
throwNDown
FragileShovel
void used(Player p, Activity a) «override»
SignalFlarePart
int partID
void used(Player p, Activity a) «override»
3
SignalFlare
void putTogether(RoundController rc)
Player
roundController
sg
RoundController
int curID
void init(int playerNum)
void startNextRound()
void endLastRound()
void lose(String cause)
void win()
PlayerContainer getPlayerContainer()
void checkTent()
ss
tent
SnowStorm
void tryStorm()
Tent
void used(Player p, Activity a) «override»
itemTileMap
0..7
tileItemMap
0..7
«Singleton»
PositionLUT
Tile getPosition(Player p)
Tile getPosition(Item i)
Player[] getPlayersOnTile(Tile t)
Item[] getItemsOnTile(Tile t)
Tile getTile(int x, int y)
void setPosition(Player p)
void setPosition(Item i)
void setPosition(PolarBear pb)
Tile getPosition(PolarBear pb)
tileList
36
t
Tile
int x
int y
int snow
boolean igluOn
boolean tentOn
+
void steppedOn(Player p)
void steppedOff(Direction dir)
int changeSnow(signed int thisMuch)
Tile getNeighbour(Direction dir)
void destroyIglu()
void buildIglu()
void putOnTent()
boolean equals(Tile tile)
boolean getIgluOn()
StableTile
UnstableTile
int capacity
int standingHere
SnowyHole
Direction
up
down
left
right
here
Direction valueOf(int direction)
int valueOf(Direction dir)
getValue()


Eisfeld - Class Diagram
Character
void step(Direction dir);
DivingSuit
void used(Player p, Activity a) «override»
Shovel
void used(Player p, Activity a) «override»
Rope
void used(Player p, Activity a) «override»
Food
void used(Player p, Activity a) «override»
Item
void throwDown()
void pickedUp(Player Picker)
void diggedUp()
void used(Player p, Activity a)
it
inHand
0..1
Player
int BodyHeat
int ID
int workPoints
boolean inWater
Item wearing
void startRound()
void fallInWater()
void ateFood()
void changeBodyHeat(signed int thisMuch)
void wear(DivingSuit suit)
boolean hasEnoughWorkPoints(int cost)
void dropFragileShovel()
roundController
ItemState
frozen
inHand
thrown Down
SignalFlarePart
int partID
void used(Player p, Activity a) «override»
3
FragileShovel
void used(Player p, Activity a) «override»
Activities
savingPeople
clearingSnow
puttingOnSuit
eatingFood
IControllable
void step(Direction dir)
void pickUp(Item i)
void clearSnow()
void digItemUp()
void savePlayers(Direction dir)
void putSignalTogether(SignalFlare sg)
void passRound()
Eskimo
int BodyHeat = 5
void buildIglu()
Researcher
int BodyHeat = 4
void detectCapacity(Direction dir)
RoundController
int curID
void init(int playerNum)
void startNextRound()
void endLastRound()
void lose(String cause)
void win()
PlayerContainer getPlayerContainer()
void checkTent()
PolarBear
void step(Direction dir)
polarbear
pid
pc
PlayerContainer
Player getPlayer(int pid)
ss
SnowStorm
void tryStorm()
sg
SignalFlare
void putTogether(RoundController rc)
tent
Tent
void used(Player p, Activity a) «override»
«Singleton»
PositionLUT
Tile getPosition(Player p)
Tile getPosition(Item i)
Player[] getPlayersOnTile(Tile t)
Item[] getItemsOnTile(Tile t)
Tile getTile(int x, int y)
void setPosition(Player p)
void setPosition(Item i)
void setPosition(PolarBear pb)
Tile getPosition(PolarBear pb)
playerTileMap
3..6
tilePlayerMap
3..6
itemTileMap
0..7
tileItemMap
0..7
tileList
36
t
Tile
int x
int y
int snow
boolean igluOn
boolean tentOn
void steppedOn(Player p)
void steppedOff(Direction dir)
int changeSnow(signed int thisMuch)
Tile getNeighbour(Direction dir)
void destroyIglu()
void buildIglu()
void putOnTent()
boolean equals(Tile tile)
boolean getIgluOn()
Direction
up
down
left
right
here
Direction valueOf(int direction)
int valueOf(Direction dir)
getValue()
UnstableTile
int capacity
int standingHere
SnowyHole
StableTile

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" Attribute 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" Attribute 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:

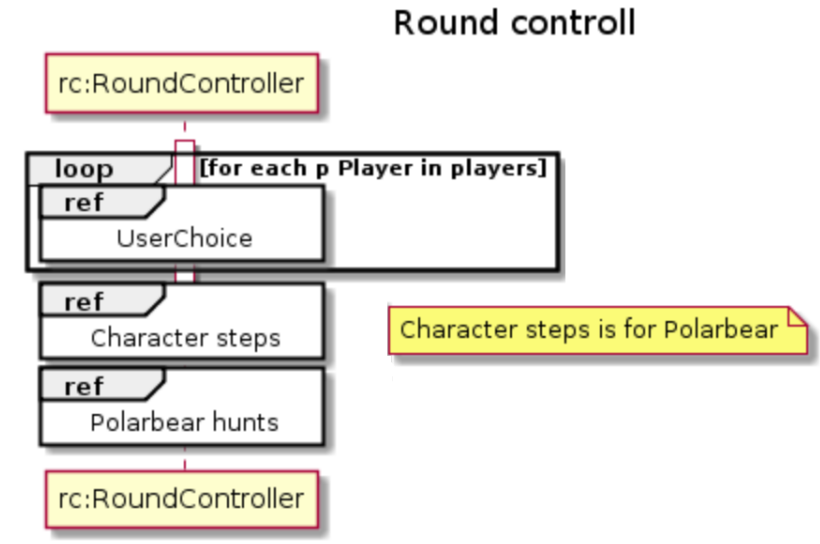
- **+boolean tentOn:** Es zeigt, ob ein Zelt sich auf dieser Platte befindet. True: Ja, False: Nein.

Methode:

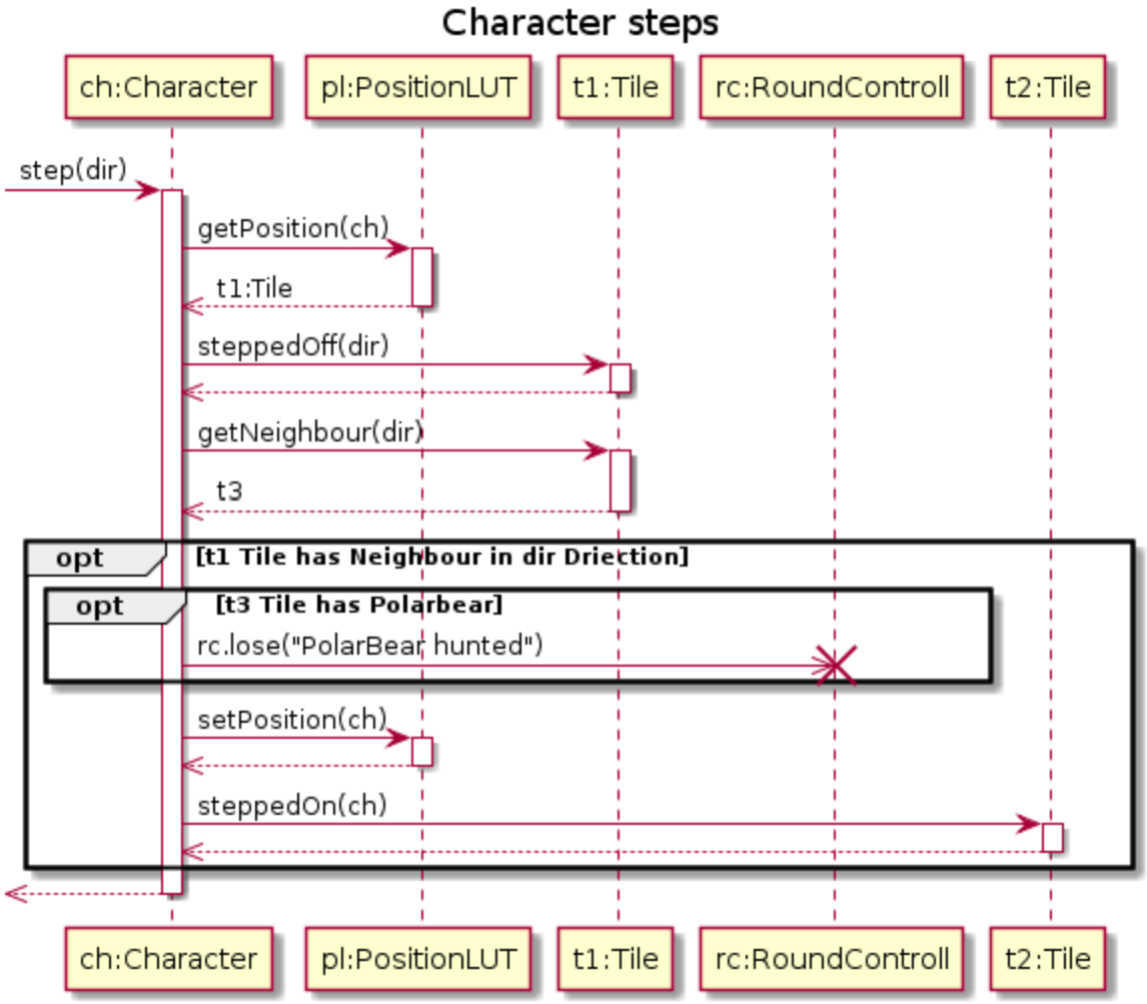
- **+boolean equals(Tile tile):** Es zeigt, ob die Position von zwei Platte gleich sind.
- **+boolean getIgluOn():** Es liefert den Wert von "igluOn" Attribut zurück.
- +**void putOnTent():** Diese Methode baut ein Zelt auf diese Platte auf.

7.0.3 Sequenzdiagramme

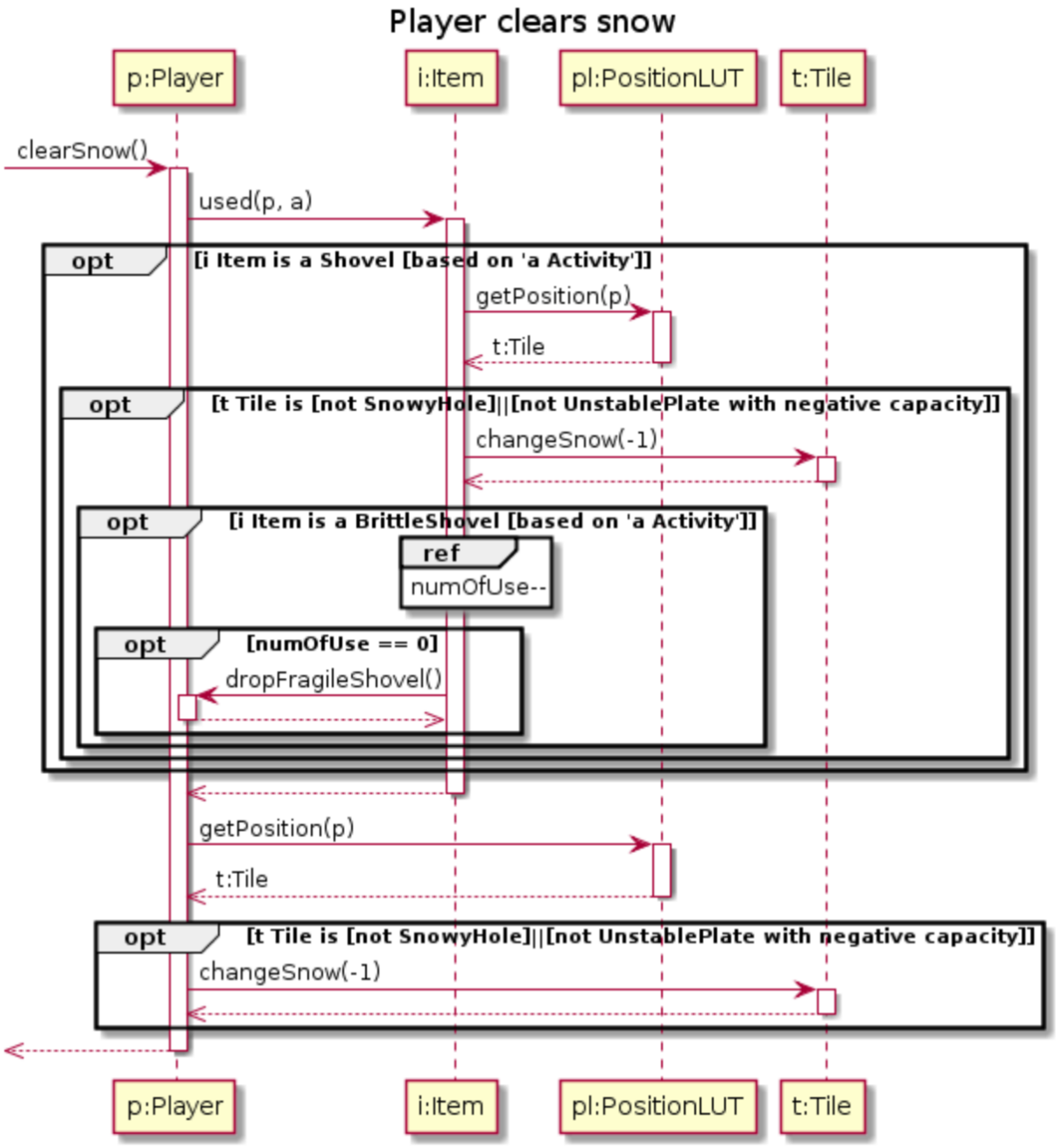
7.0.3.1 Round controll



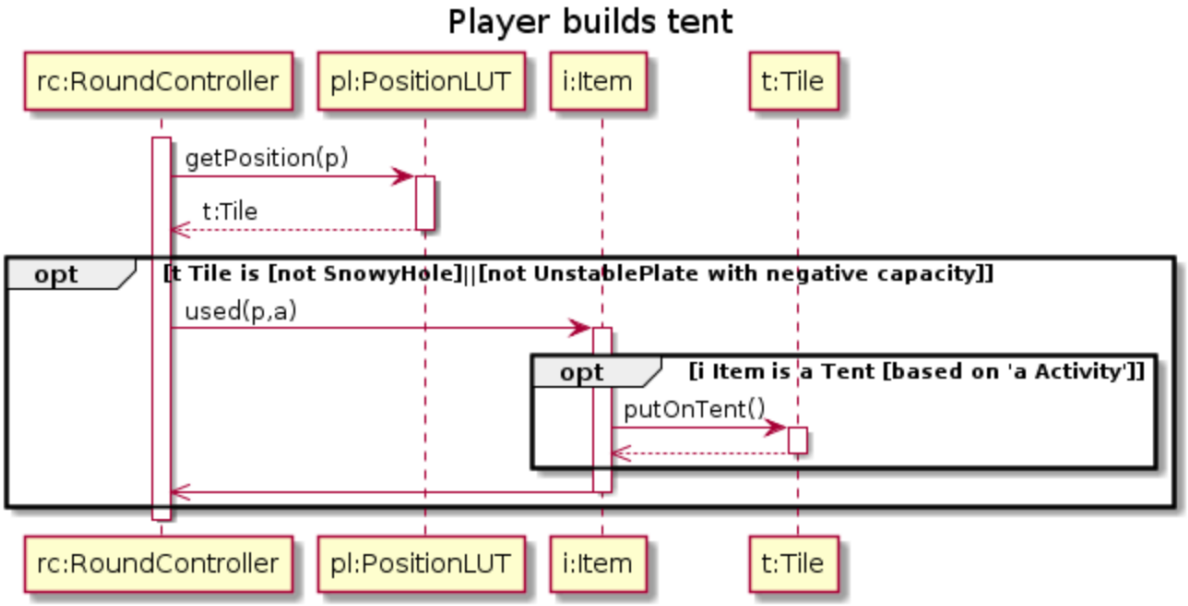
7.0.3.2 Character steps



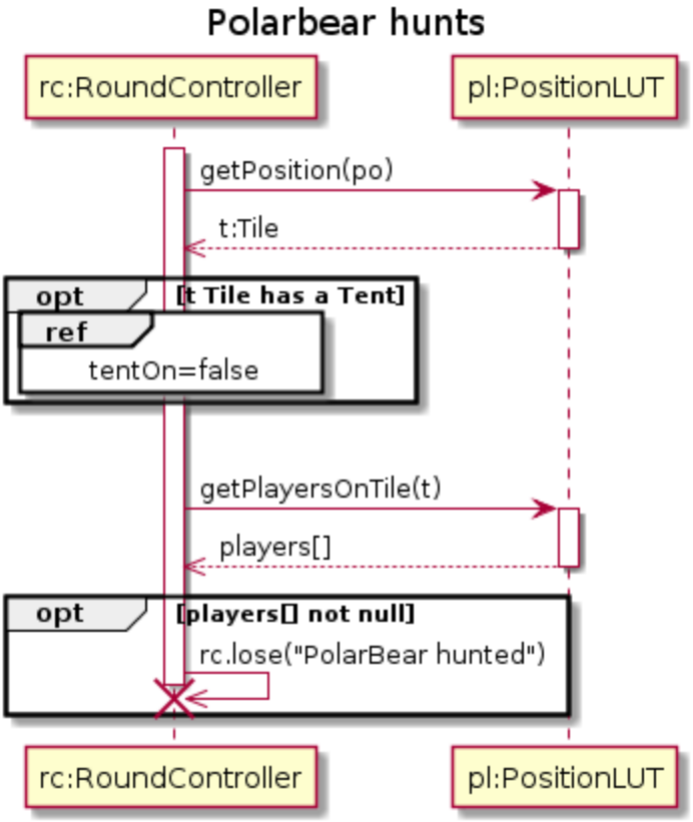
7.0.3.3 Player clears snow



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 das 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 Spieler 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 Schneeinheiten 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**	*Als Befehl:* Tritt das Spieler in die gegebene Richtung (falls möglich) *Als Option:* 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 *(Die Forscher können auch das Feld unten ihr analysieren – dazu müssen wir das Option leer lassen)*
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 bedutet "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 Iglu.

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” /
“Succesfully 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” /
“Succesfully 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"

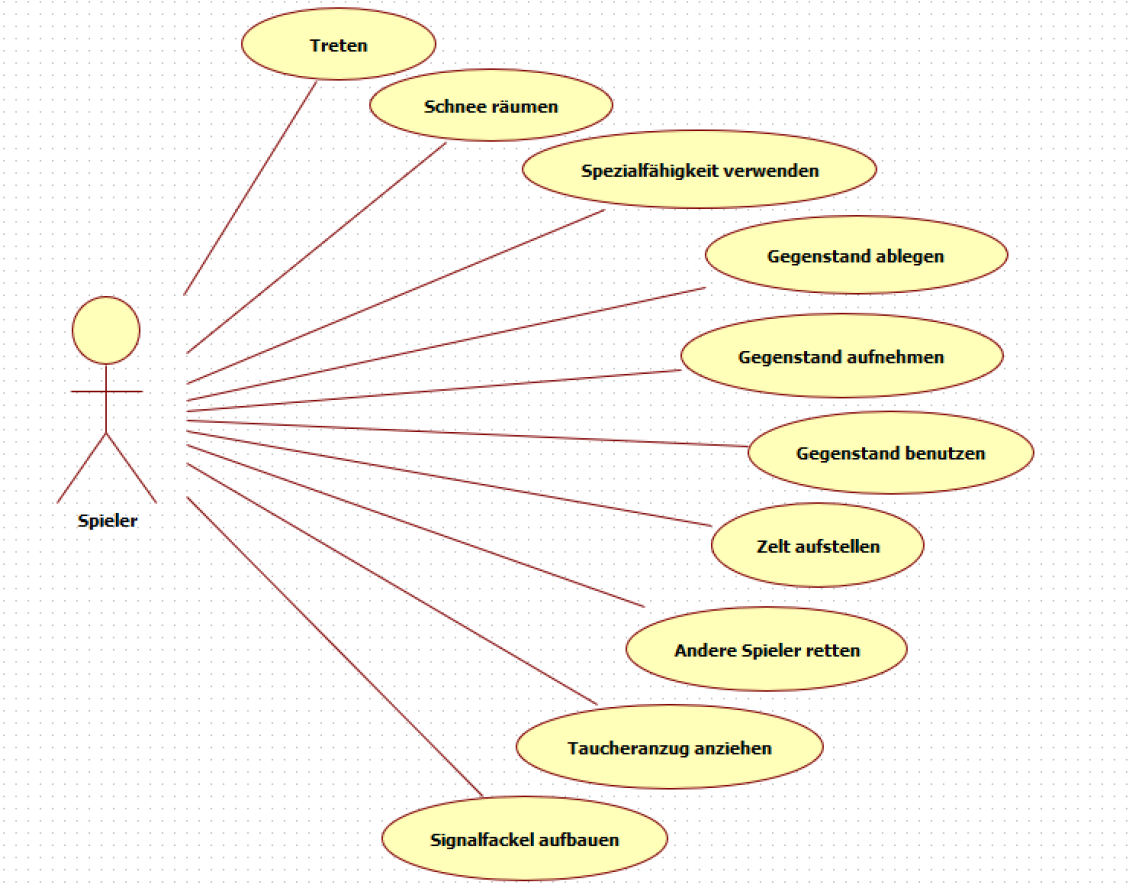
Andere, allgemeine Ausgänge:

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 detailierte use-case



Name von Use-case	Treten
Kurze Beschreibung	Spieler tretet an eine andere Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	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
	1.2.1. Es gibt keine Platte in von Spieler

Name von Use-case	Schnee räumen
Kurze Beschreibung	Spieler entfernt Schnee von Platte
Akteur	Spieler, Kontroller
Haupttätigkeit	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	1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt:'UseSkill' (Spezialfähigkeit verwenden) 3. Kontroller speichert die Anfrage von Spieler

Bemerkung: Bei Spezialfähigkeit ist gemeint die Gegenstände im Spiel: Essen, Taucheranzug, Schaufel, Teile der Signalfackel.

Name von Use-case	Gegenstand aufnehmen
Kurze Beschreibung	Spieler nimmt einen Gegenstand auf
Akteur	Spieler, Kontroller
Haupttätigkeit	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	Zelt aufstellen
Kurze Beschreibung	Spieler stellt Zelt auf
Akteur	Spieler, Kontroller
Haupttätigkeit	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	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	1. Kontroller bittet den Spieler einen Befehl anzugeben 2. Spieler drückt: ’ PutSignalTogether’ (Signalfackel aufbauen) 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 änlicherweise 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.