# 4. Ausarbeitung von Analyse-Modell
## 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 |


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 gefriertes Essen generiert)*. Sie haben drei Zustände: gefrieren, geworfen und in der Hand, es ist in einem Enum. Diese Gegenstände erlauben verschieden 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.

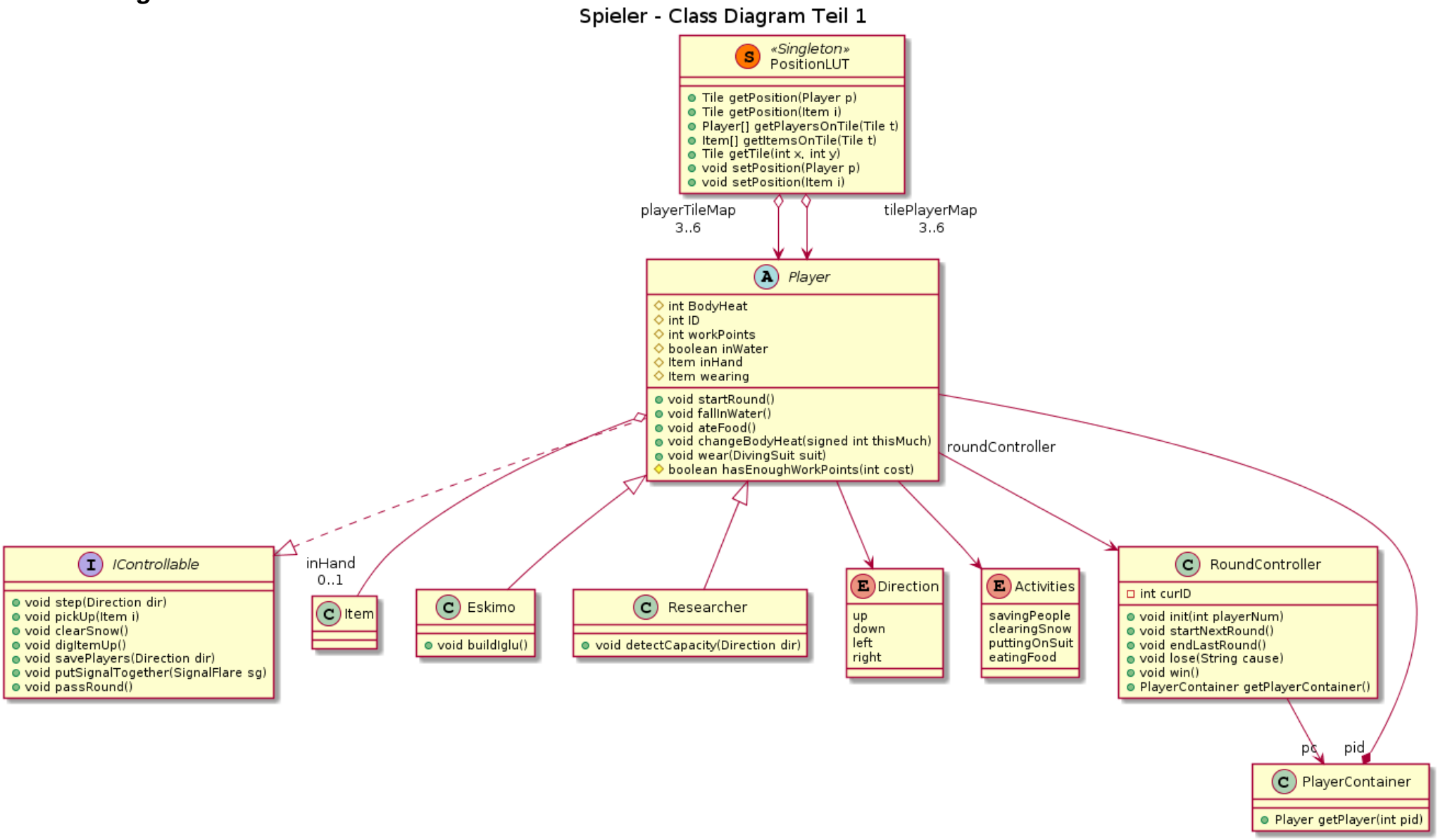
### 4.1.5 RoundController

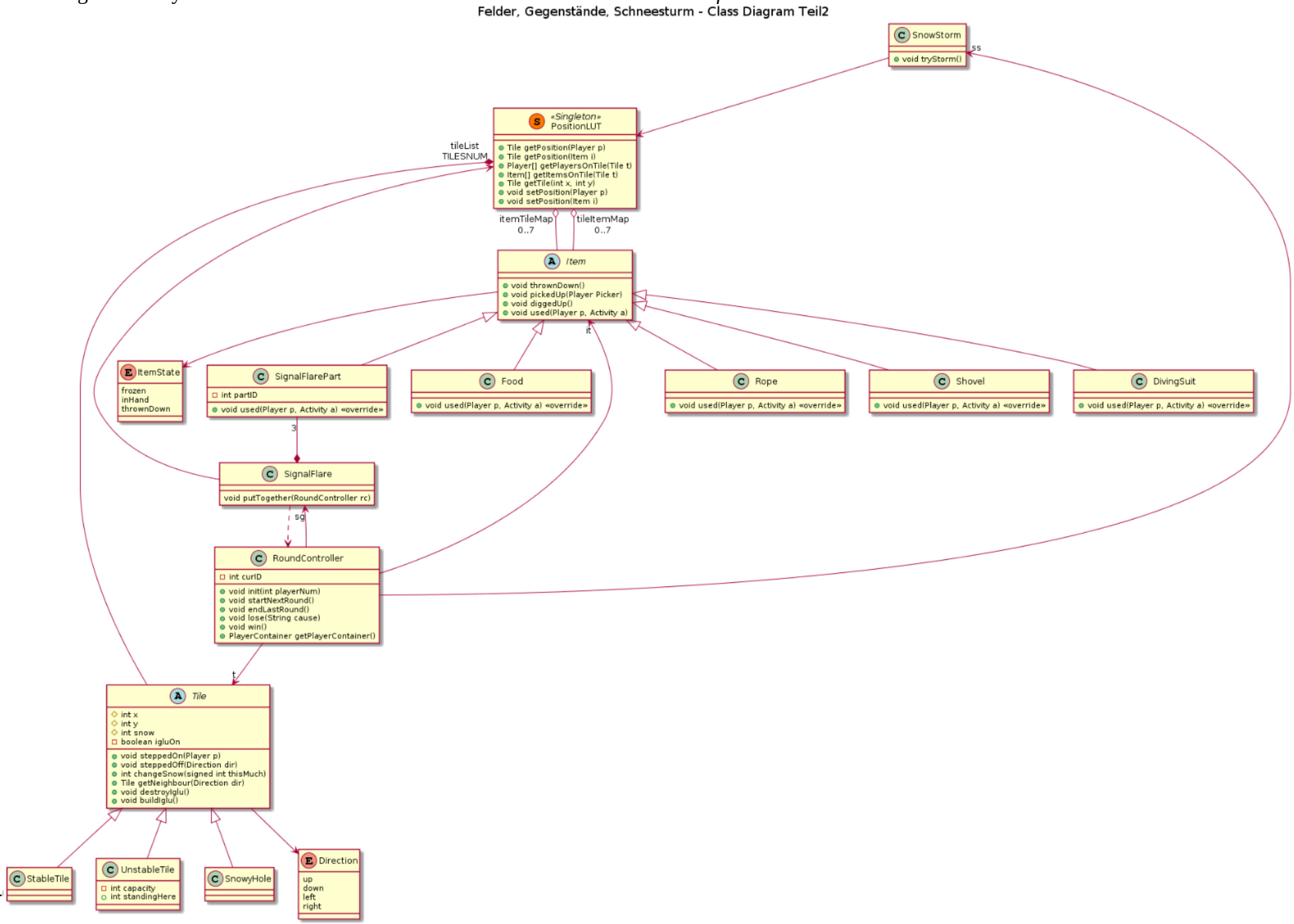
Dieses einzige Objekt wird für Initiierung und Laufen des Spiels verantwortlich. Es wird nach der Anzahl der Spieler fragen und dann die anderen Objekte initialisieren. Es mach die gebrauchte „checks“ und „cleanup“/Initiierung zwischen die Runde der Spieler und behandelt Sieg/Fehlschlag.

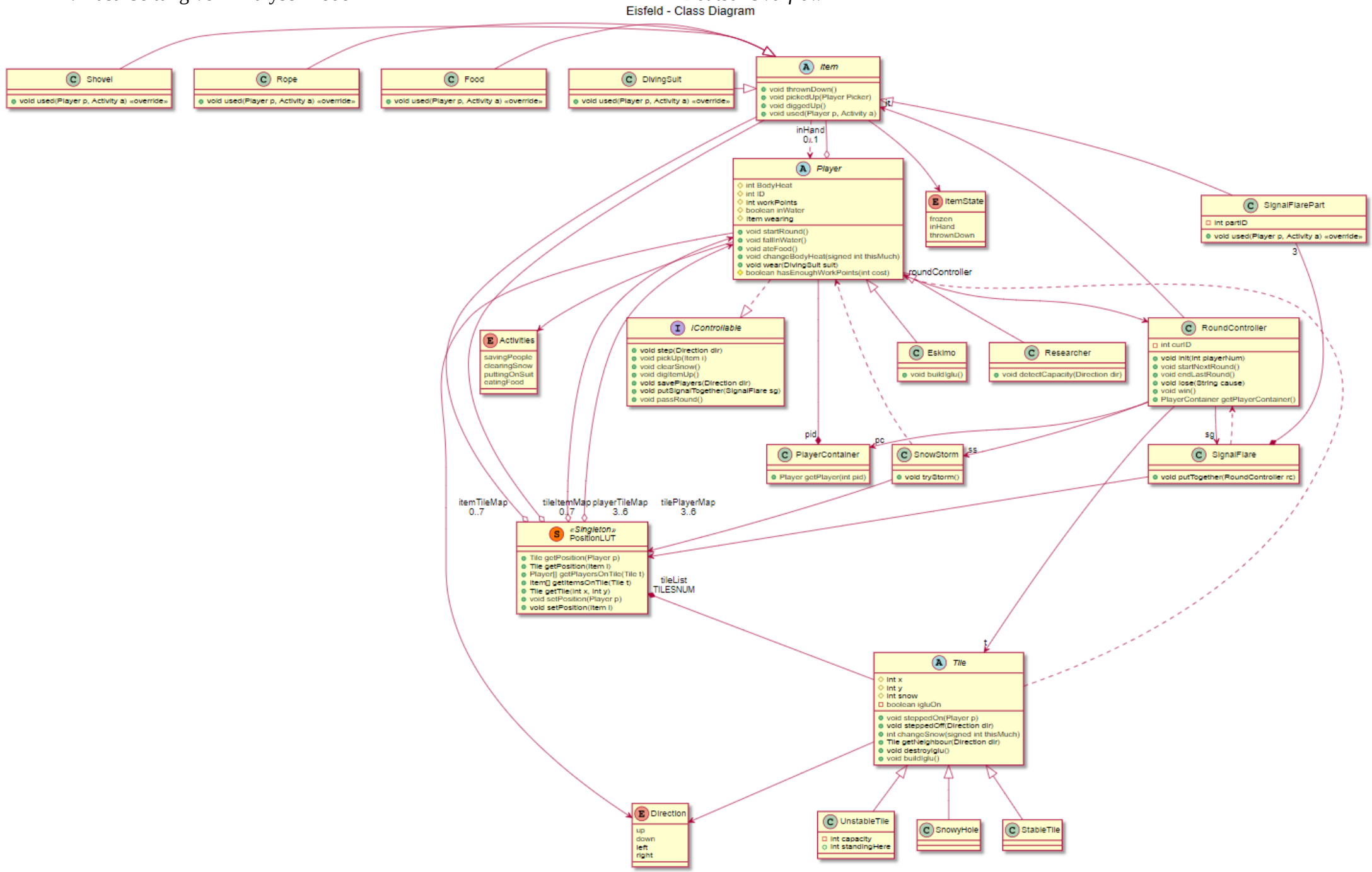
### 4.1.6 SignalFlare

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

## 4.2 Klassendiagramme




Felder, Gegenstände, Schneesturm - Class Diagram Teil2
SnowStorm
void tryStorm()
ss
«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)
tileList
TILESNUM
itemTileMap
0..7
tileItemMap
0..7
Item
void thrownDown()
void pickedUp(Player Picker)
void diggedUp()
void used(Player p, Activity a)
it
ItemState
frozen
inHand
thrownDown
SignalFlarePart
int partID
void used(Player p, Activity a) «override»
Food
void used(Player p, Activity a) «override»
Rope
void used(Player p, Activity a) «override»
Shovel
void used(Player p, Activity a) «override»
DivingSuit
void used(Player p, Activity a) «override»
3
SignalFlare
void putTogether(RoundController rc)
sg
RoundController
int curID
void init(int playerNum)
void startNextRound()
void endLastRound()
void lose(String cause)
void win()
PlayerContainer getPlayerContainer()
t
Tile
int x
int y
int snow
boolean igluOn
void steppedOn(Player p)
void steppedOff(Direction dir)
int changeSnow(signed int thisMuch)
Tile getNeighbour(Direction dir)
void destroyIglu()
void buildIglu()
StableTile
UnstableTile
int capacity
int standingHere
SnowyHole
Direction
up
down
left
right


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
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)
ItemState
frozen
inHand
thrownDown
SignalFlarePart
int partID
void used(Player p, Activity a) «override»
3
roundController
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
void buildIglu()
Researcher
void detectCapacity(Direction dir)
RoundController
int curID
void init(int playerNum)
void startNextRound()
void endLastRound()
void lose(String cause)
void win()
PlayerContainer getPlayerContainer()
pid
pc
ss
sg
PlayerContainer
Player getPlayer(int pid)
SnowStorm
void tryStorm()
SignalFlare
void putTogether(RoundController rc)
itemTileMap
0..7
tileItemMap
0..7
playerTileMap
3..6
tilePlayerMap
3..6
«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)
tileList
TILESNUM
t
Tile
int x
int y
int snow
boolean igluOn
void steppedOn(Player p)
void steppedOff(Direction dir)
int changeSnow(signed int thisMuch)
Tile getNeighbour(Direction dir)
void destroyIglu()
void buildIglu()
Direction
up
down
left
right
UnstableTile
int capacity
int standingHere
SnowyHole
StableTile

## *4.3 Beschreibung der Klassen*

### 4.3.1 Player

- **Verantwortung**

*Diese Klasse ist eine abstrakte Klasse, also man kann es nicht instanziieren, 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 roundCotroller:** 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.

- **+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 schütz 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 instanziieren. 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 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.

### 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 abzuschießen. 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[TILESNUM]:** Die Eisplatten (alle) werden in dieser Klasse (*PositionLUT* ) in diesem Array gespeichert. Es ist TILESNUM, weil es fest TILESNUM 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 instanziieren. 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(Plfayer 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

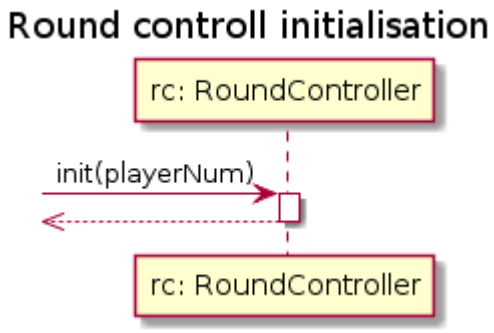
- **Verantwortung**

*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.*

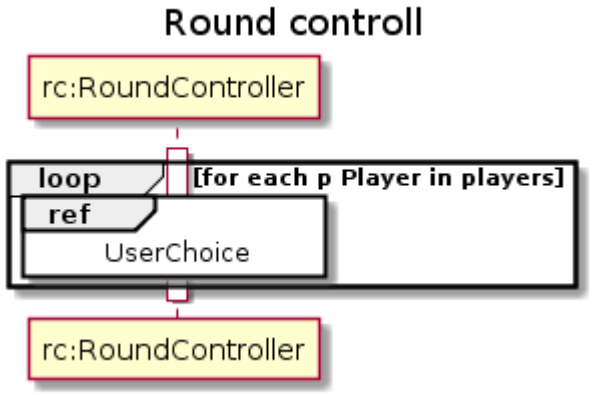
- **Aufzählungsliterale**
  - **up:** Auf die nördliche Platte treten. (Nord)
  - **down:** Auf die untere Platte treten. (Süd)
  - **left:** Auf die linke Platte treten. (West)
  - **right:** Auf die rechte Platte treten. (Ost)

## *4.4 Sequenz Diagramme*

### 4.4.1 Round controll initialization

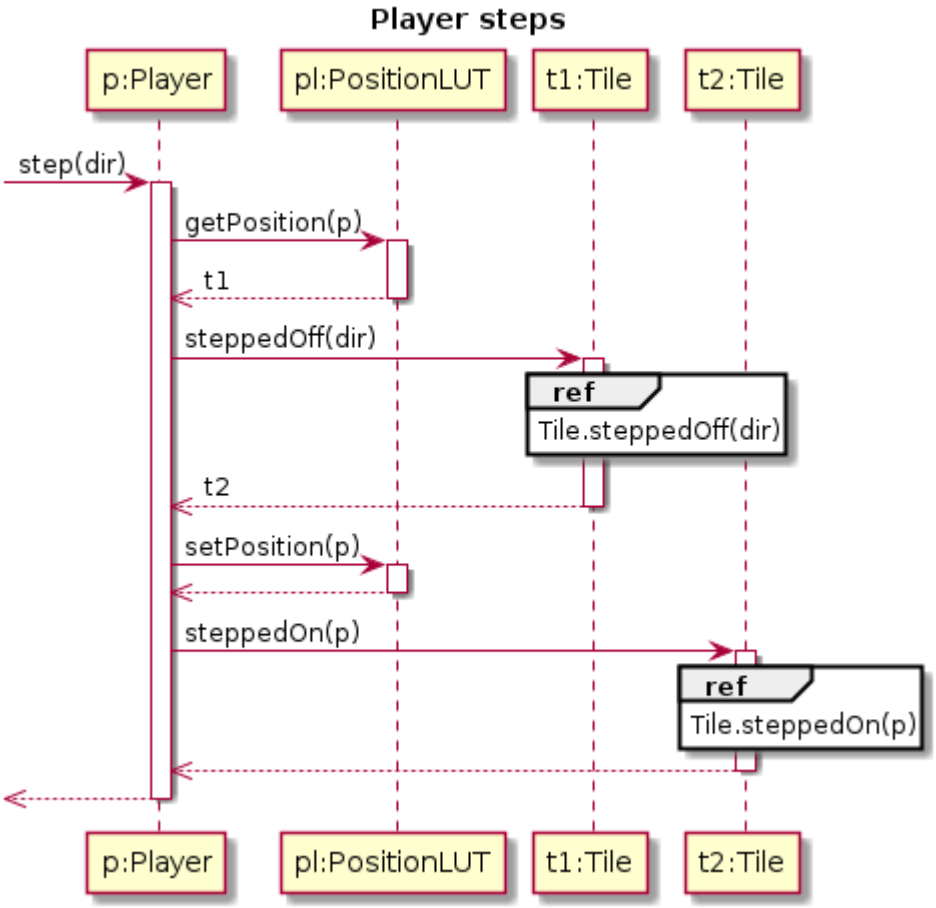



### 4.4.2 Round controll

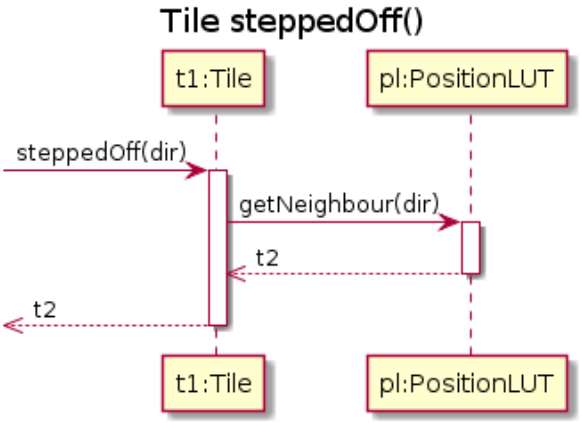


### 4.4.3 UserChoice

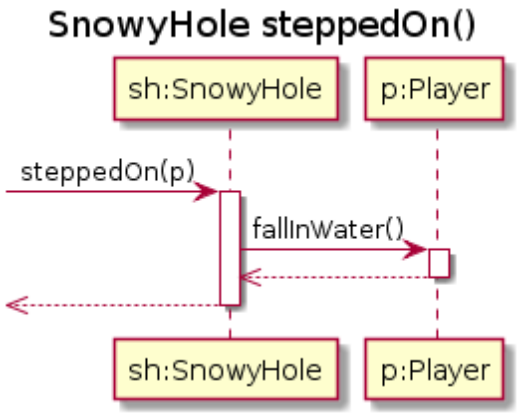
### 4.4.4 Player steps



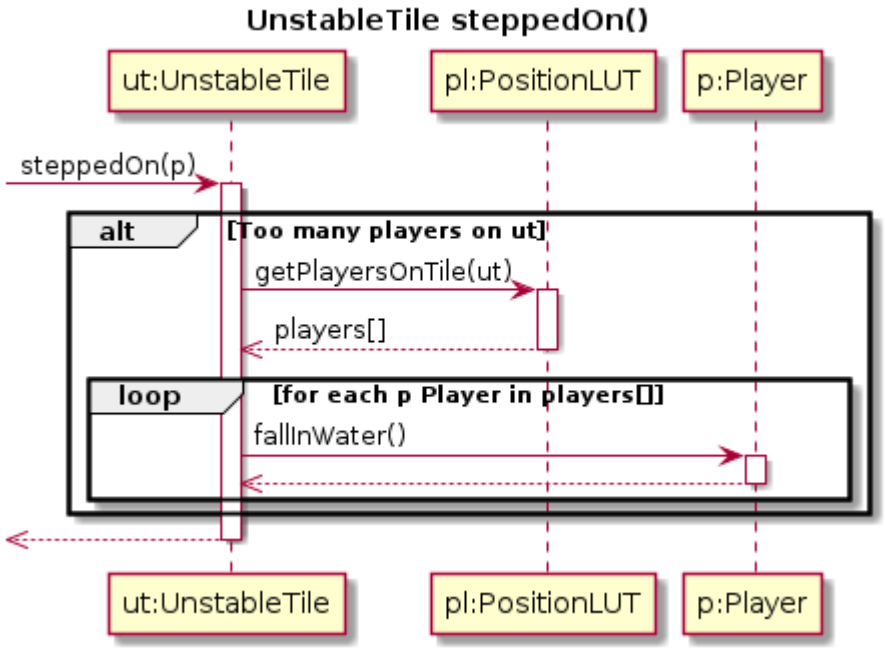

### 4.4.5 Tile steppedOff



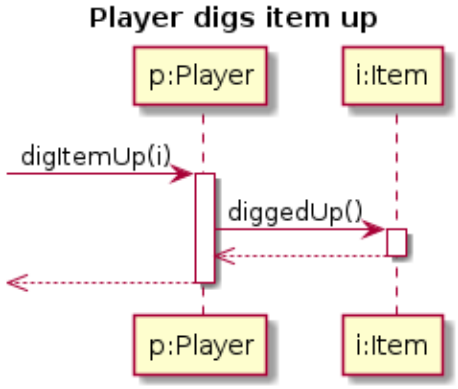
### 4.4.6 SnowyHole steppedOn



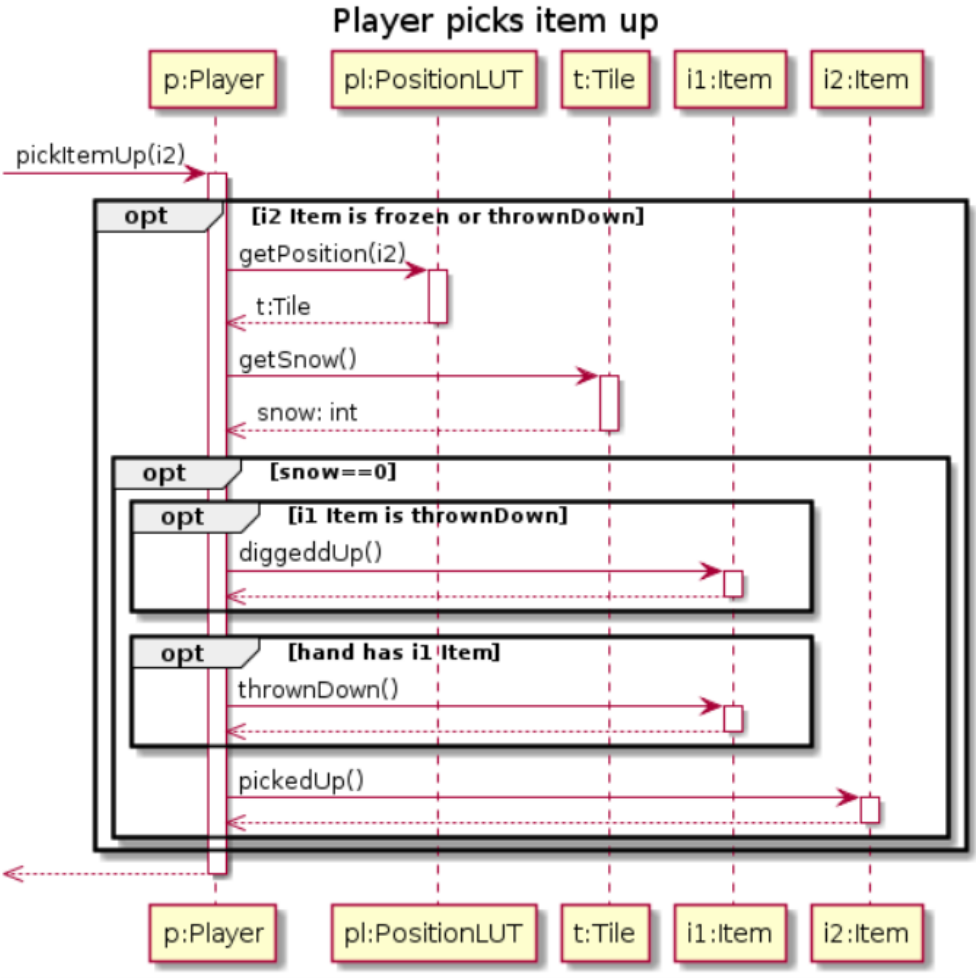

### 4.4.7 UnstableTile steppedOn



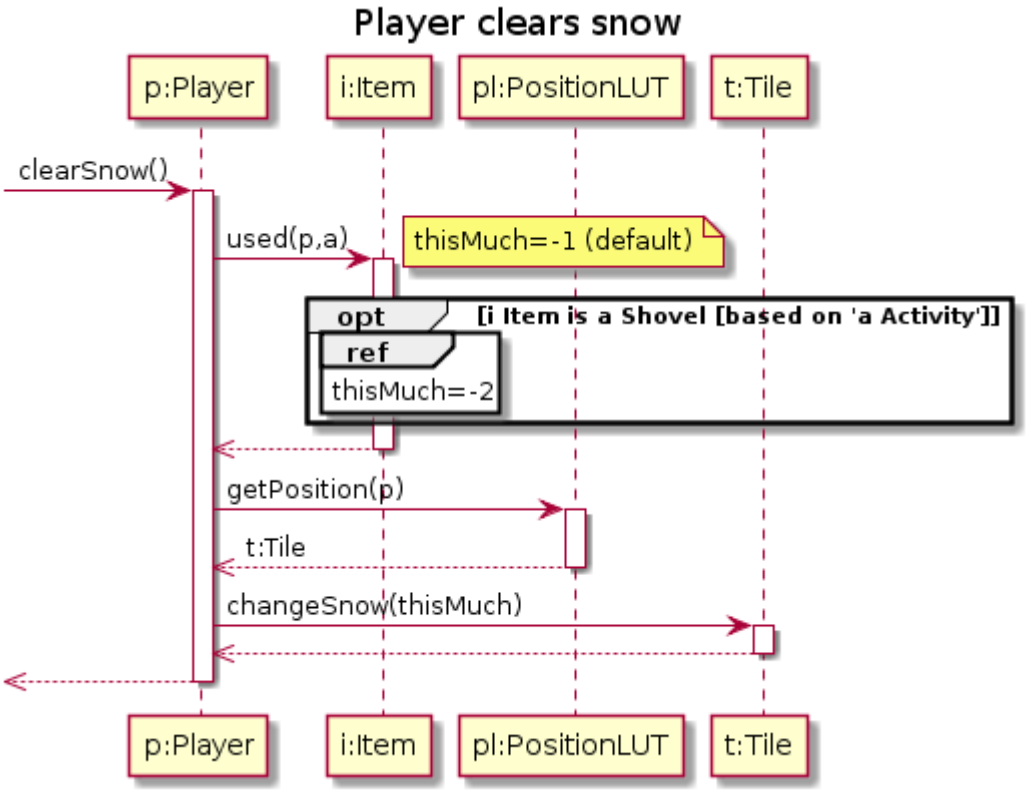
### 4.4.8 Player digs item up



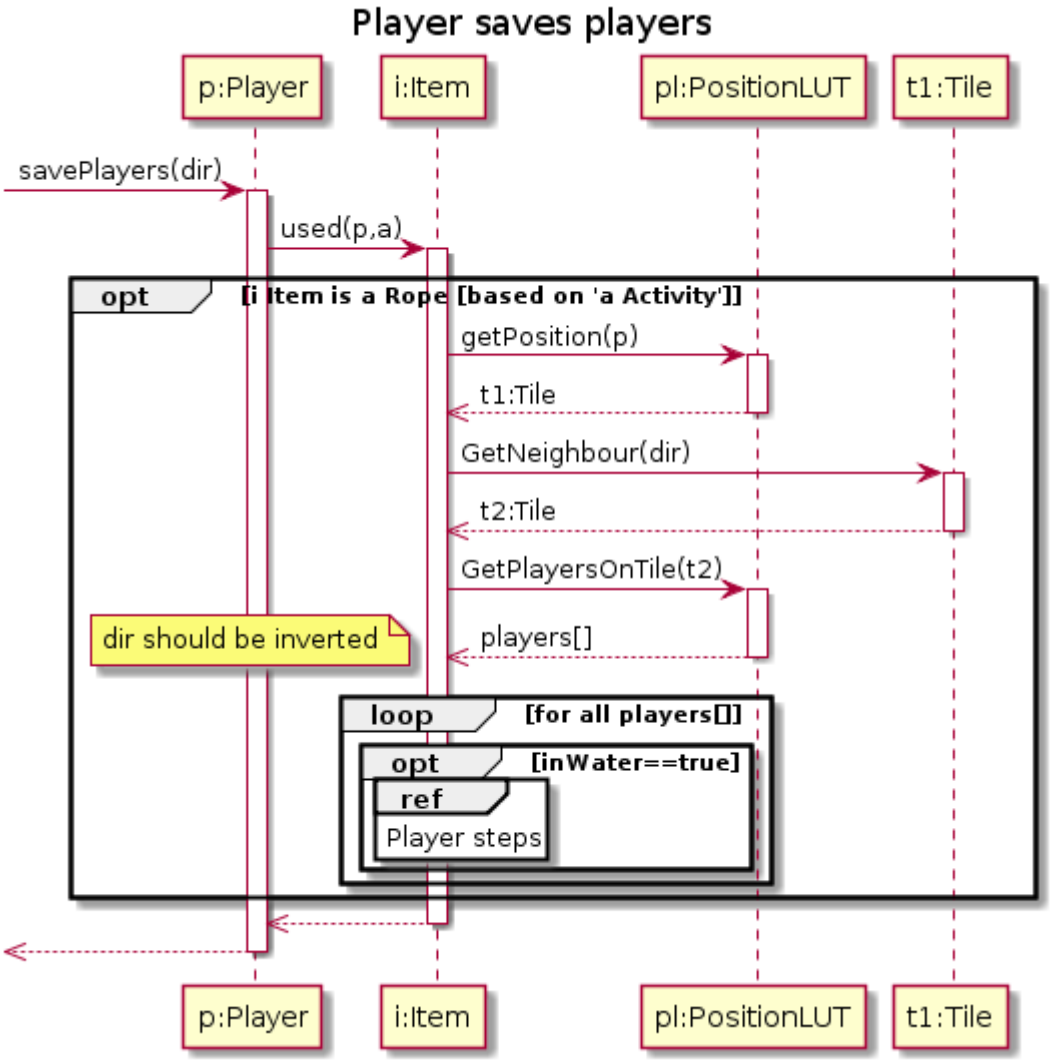

### 4.4.9 Player picks item up



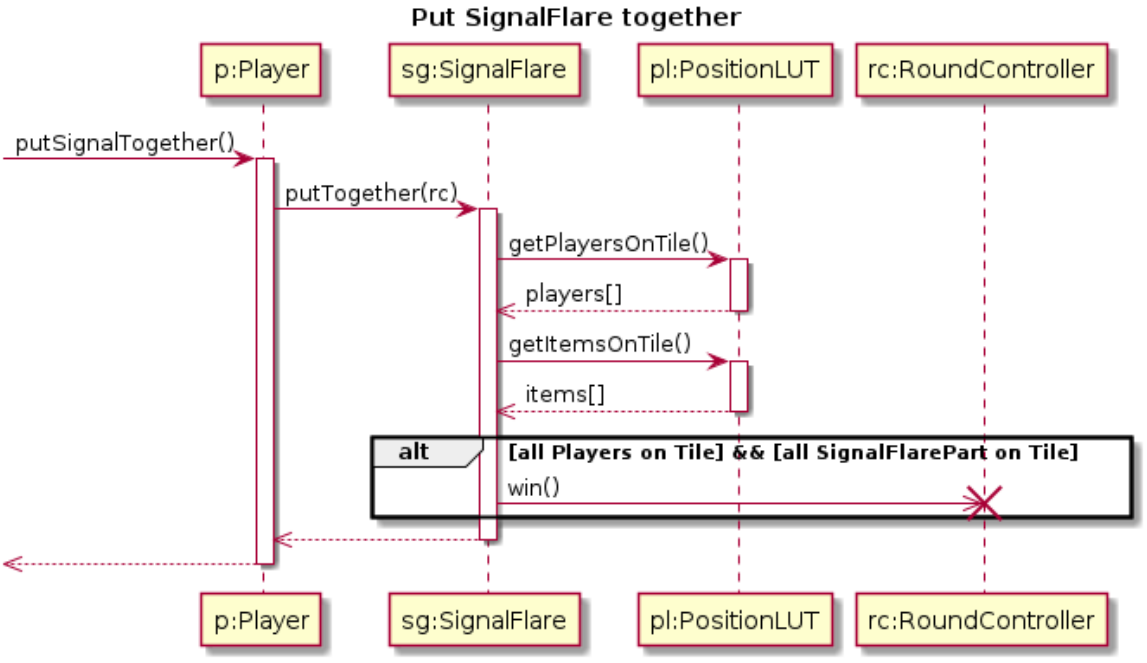
### 4.4.10 Player clears snow



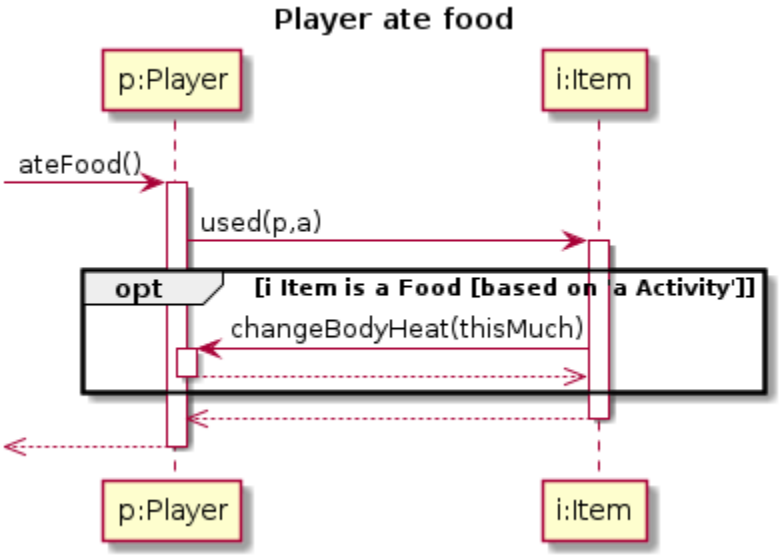
### 4.4.11 Player saves players



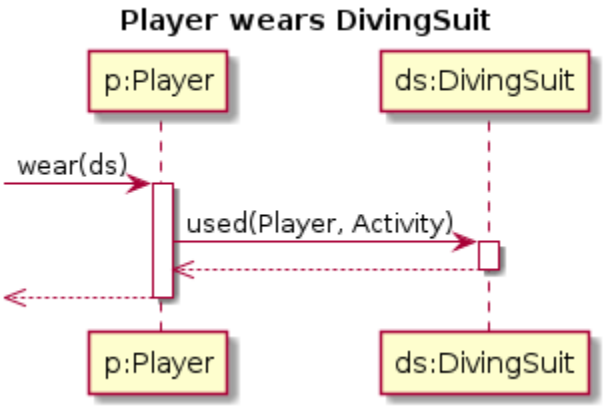

### 4.4.12 Put SignalFlare together



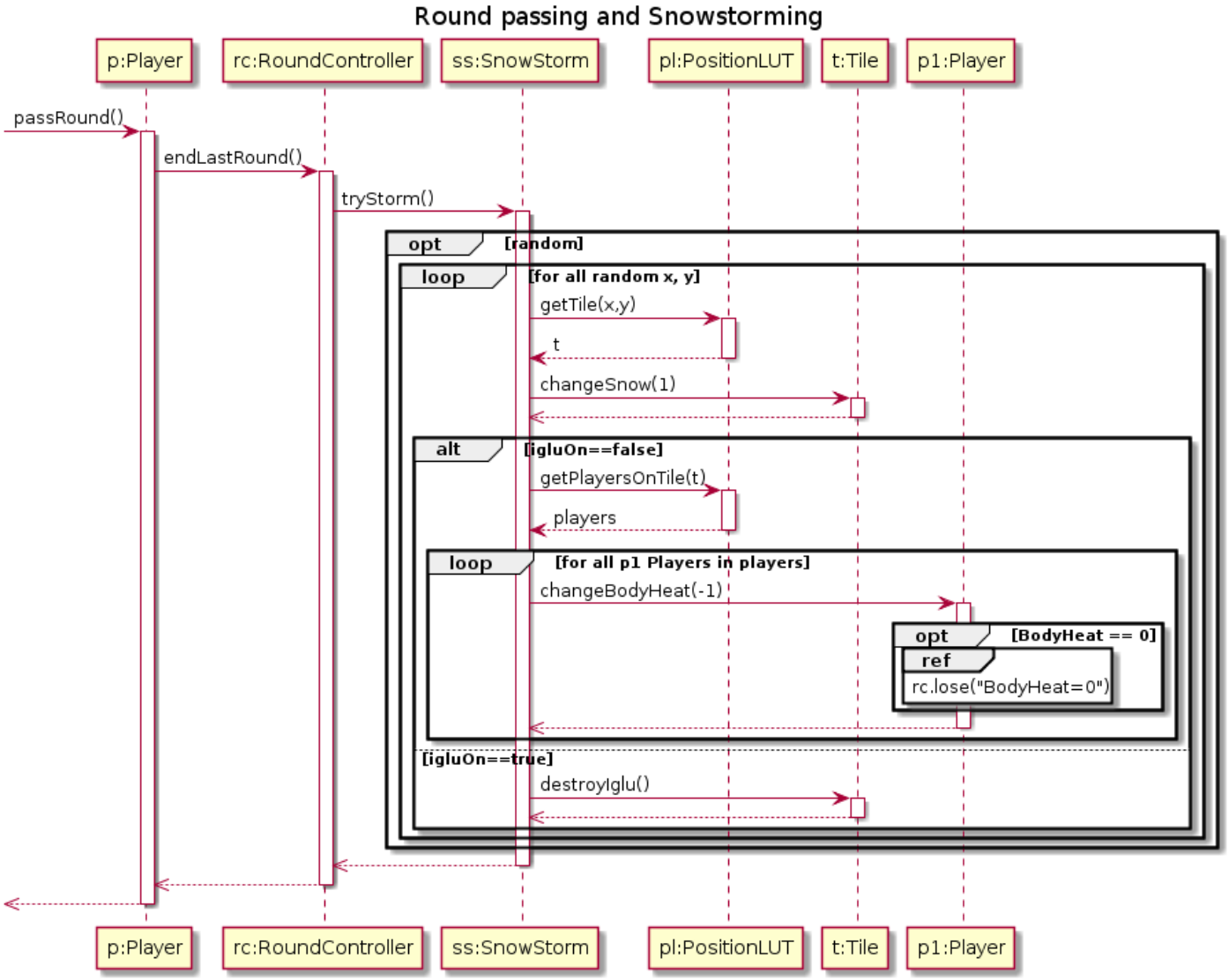
### 4.4.13 Player ate food



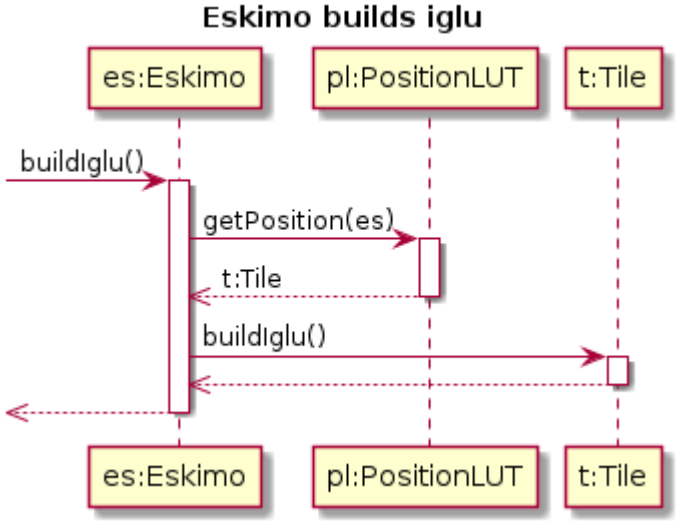

### 4.4.14 Player wears DivingSuit



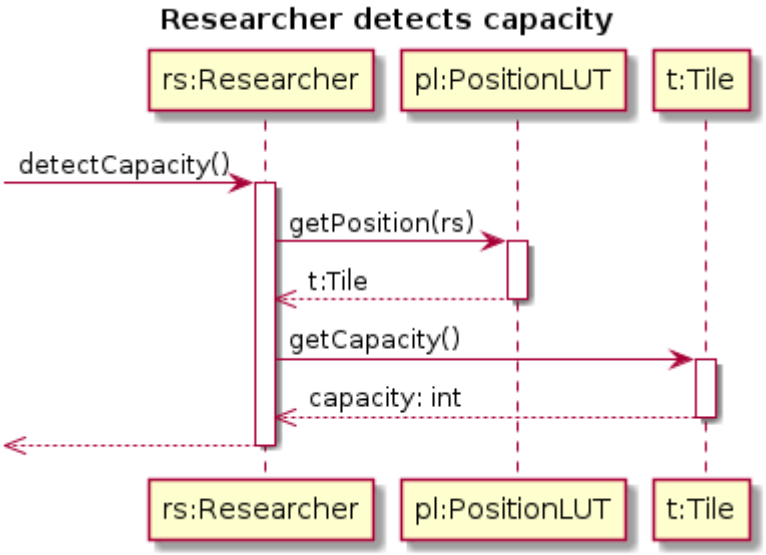
### 4.4.15 Round passing and Snowstorming




### 4.4.16 Eskimo builds iglu

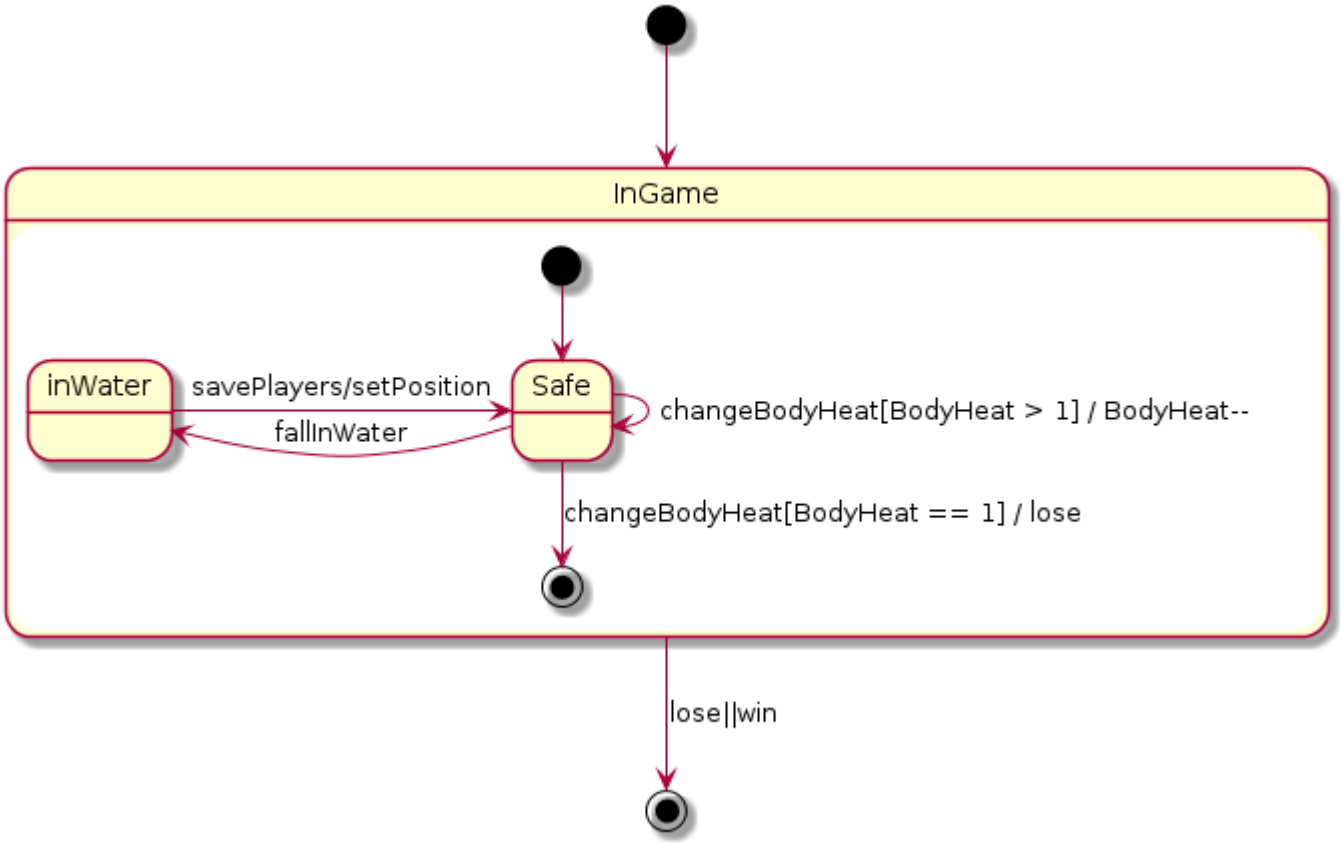


### 4.4.17 Researcher detects capacity

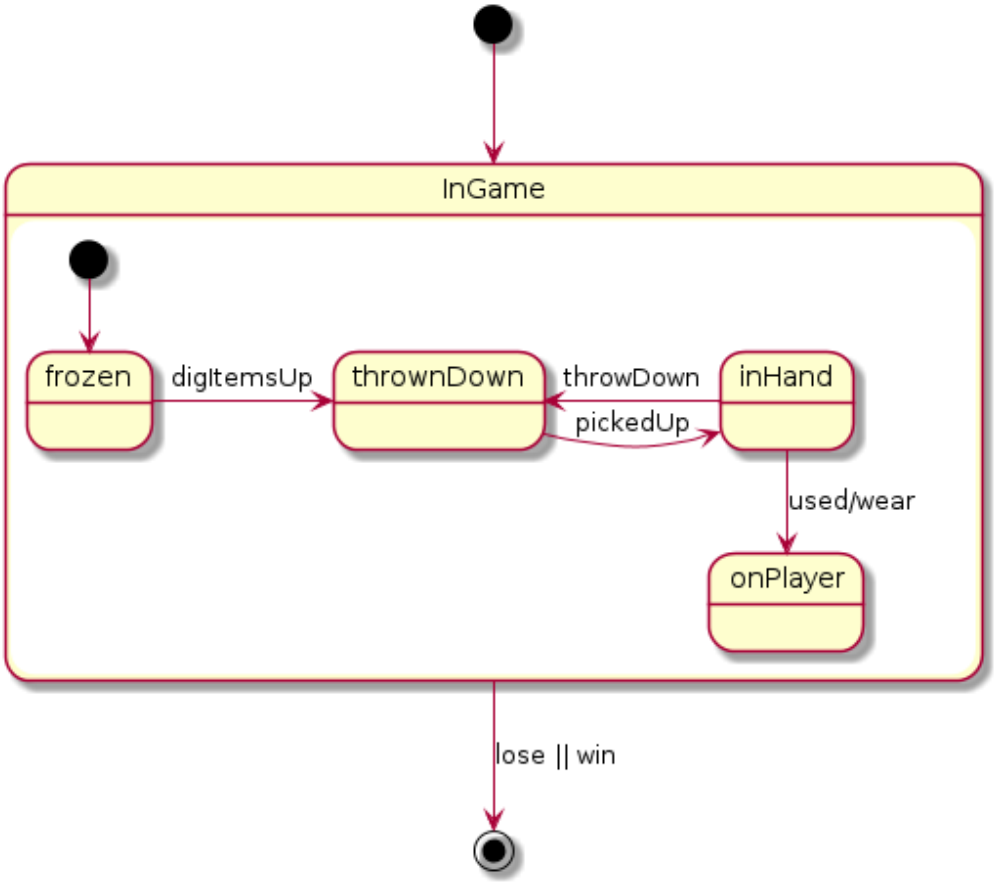



## *4.5 Zustandsdiagramme (State-charts)*

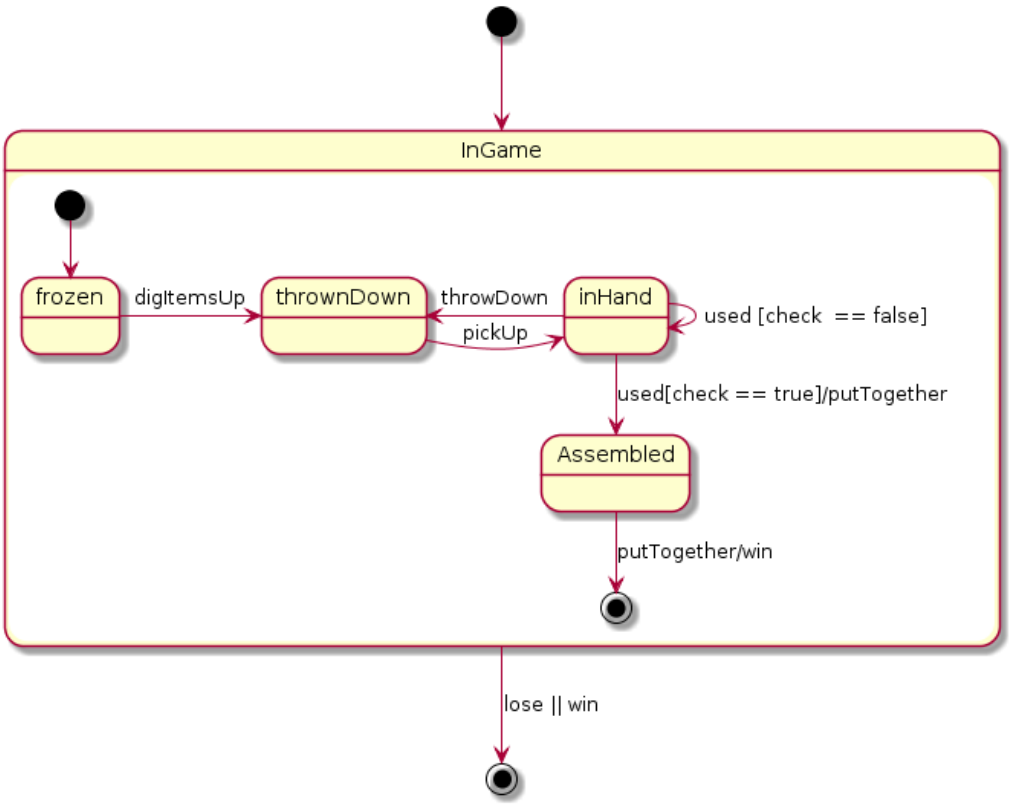
### 4.5.1 PlayerStates:



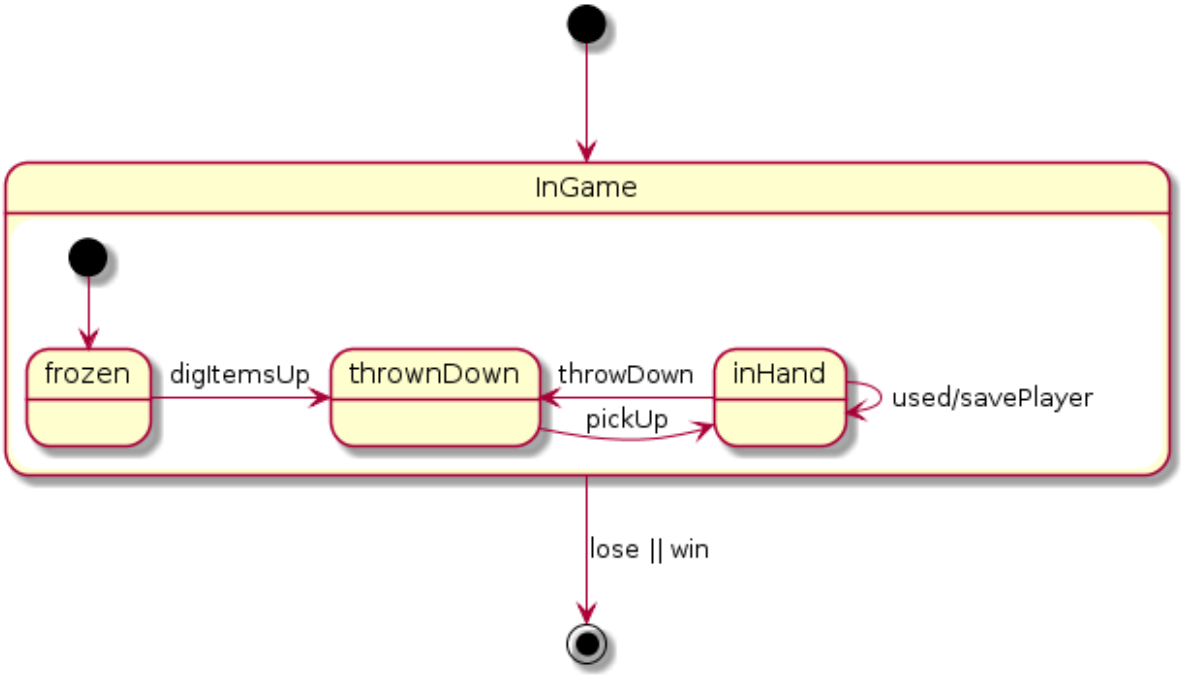
### 4.5.2 ItemStates(DivingSuit):



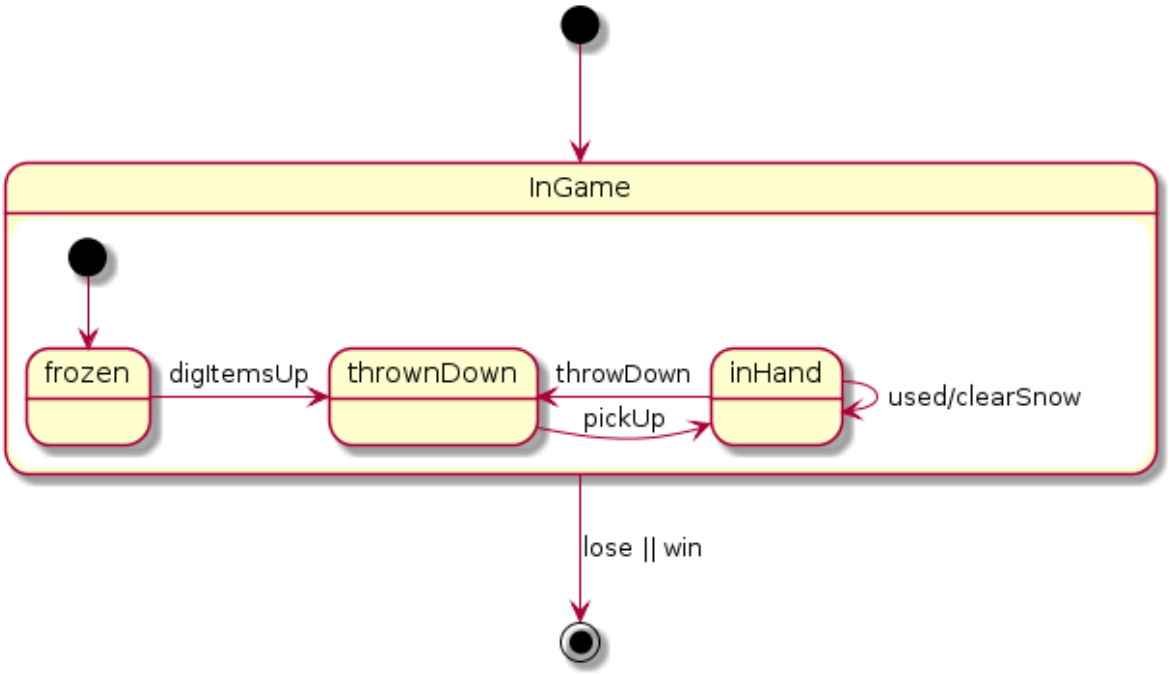

### 4.5.3 ItemStates(SignalFlarePart)



### 4.5.4 ItemStates(Rope):




### 4.5.5 ItemStates(Shovel)



### 4.5.6 ItemStates(Food)