

2. Anforderungen, Projekt, Funktionalität

2 – Halooo Matematikeeer

Konzulent:
Márton Kovács

Mitglieder

Henrietta Domokos	J0DCPT	domoheni@gmail.com
Ábel Borbély	DRP0DT	babel1122@gmail.com
Renátó Hrotkó	OIT6HD	renatohrotko@gmail.com
Vince Pongrácz	MKMDJO	pongvin@gmail.com

17.02.2021

2. Anforderungen, Projekt, Funktionalität

2.1 Einführung

2.1.1 Ziel

Diese Dokument beschäftigt sich mit dem Asteroidenspiel von dem Team Halooo Matematikeeer. In diesem Dokument kann man über Planung, Implementation usw. lesen.

2.1.2 Fachgebiet

Dieser Software ist ein Computerspiel, also es lässt sich nicht behaupten, dass es zu einem bestimmten Fachgebiet gehört. Sonder ist der Ziel unsere fachliche Entwicklung und die Unterhaltung der Spieler.

2.1.3 Definitionen, Abkürzungen

gs.: grundsätzlich

wi.: wichtig

opt.: optionell

Besch.: Beschreibung (Aufgabenbeschreibung)

TDR: Team Defined Requirement (Team definierte Anforderung)

2.1.4 Bezug

Vorige Hausaufgaben aus Softwaretechnologie

Aufgabebeschreibung: <https://www.iit.bme.hu/file/11582/feladat>

2.1.5 Zusammenfassung

Die Folgenden sind in diesem Dokument zu finden:

- Überblick: Das stellt der ganze Projekt in einem kurzen Form vor.
- Anforderungen: Diese Anforderungen werden später in Betracht genommen werden und beim Softwareentwurf benutzt.
- Wichtige Use-Case Diagramme: Die wichtigste Diagramme werden aufgelistet und beschreibt.
- Wörterbuch: Die Erklärung/Definition der solchen nicht alltäglichen Wörter, die in diesem Dokument und in dem Spiel vorkommt.
- Projekt Plan: Die realisierende Aufgaben und Termine werden aufgelistet.
- Tagebuch: Die fertige Aufgaben werden hier angegeben und auch die benötigte Zeit.

2.2 Überblick

2.2.1 Allgemeiner Überblick

Die Grundarchitektur des Programms besteht aus zwei Hauptelementen. Die Asteroiden, und die Dinge die sich auf ihnen befinden. Diese Dingen sind die Raumschiffe der Siedler, die von den Spielern gesteuert sind, autonome Roboter, die sich selbst steuern, und Teleporterpaare, die Verkehr zwischen nicht benachbarten Asteroiden ermöglichen.

2.2.2 Funktionen

Die Software ist ein Computerspiel, in dem Spieler versuchen, ein gemeinsames Ziel zu erreichen. Das Spiel findet in einem Asteroidenfeld statt. Die Spieler übernehmen die Rolle von Siedlern, die mit den im Kern der Asteroiden enthaltenen Ressourcen eine Basis aufbauen wollen. Es gibt Elemente, die den Spielern helfen können, dieses Ziel zu erreichen, und es gibt einige, die ihren Fortschritt behindern oder sie sogar töten.

Ziel des Spieles: Die Spieler müssen Ressourcen sammeln. Die Spieler gewinnen, wenn mindestens drei Einheiten jedes Materials auf einem einzelnen Asteroiden gesammelt werden.

Reisen durch das Spielfeld: Die Spieler starten auf einem HomeAsteroid, worauf sie die Basis aufbauen müssen. In einem Schritt können Spieler von einem Asteroiden zum anderen reisen, aber mit der wichtigen Einschränkung, dass sie benachbart sind. Zu einem Asteroiden kann theoretisch von einigen bis Hunderte von Nachbarn gehören. Siehe auch: Teleporterpaare.

Ressourcen: Im Kern der meisten Asteroiden finden Spieler Ressourcen. Das innere von Asteroiden ist homogen, so dass Sie nur eine Einheit von einem Material in einem einzelnen Asteroiden finden können. Es gibt auch hohle Asteroiden, die überhaupt nichts enthalten. Die möglichen Materialen sind Eis, Eisen, Kohle und Uran.

Bohren und Bergbau nach Ressourcen: Um Ressourcen sammeln zu können, müssen die Spieler die äußere Schichten der Asteroiden die den Kern bedecken durchbohren. Nur dann können sie die Ressource abbauen, wenn überhaupt eine vorhanden ist. Ein Siedler kann höchstens 10 Ressourceneinheiten halten. Wenn der Spieler es wünscht, kann er eine Ressourceneinheit in einem hohlen, gebohrten Asteroiden aufbewahren.

Autonome Roboter bauen: Die Spieler können die gesammelten Ressourcen verwenden, um autonome Roboter zu bauen, die gemäß den zuvor genannten Regeln reisen können. Ansonsten können sie die äußeren Schichten eines Asteroiden bohren, aber nicht die Ressourcen darin abbauen. Roboter können aus jeweils einer Einheit Eisen, Kohle und Uran gebaut werden.

Teleporterpaare: 2 Einheiten Eisen und 1 Einheit Eis und Uran sind erforderlich, um ein Paar Teleporter zu bauen. Spieler können Teleporter im Asteroidenfeld platzieren. Wenn beide Teleporter platziert sind, können Spieler und Roboter zwischen ihnen reisen. Ein Spieler kann höchstens ein Paar Teleporter halten.

Radioaktive Ressourcen und die Sonne: Die Spieler müssen vorsichtig mit Asteroiden sein, die der Sonne nahe sind. Uran ist offensichtlich radioaktiv. Ein Asteroid mit radioaktivem Kern explodiert, wenn er sich in der Nähe der Sonne befindet, und seine Außenseite ist vollständig durchbohrt, wodurch die Siedler auf diesem Asteroiden getötet werden. Roboter überleben die radioaktive Explosion und landen auf einem benachbarten Asteroiden.

Solarflairs: Von Zeit zu Zeit treten Sonnenflair auf, die den Asteroidengürtel erreichen. Dies wirkt sich sowohl auf Roboter als auch auf Spieler aus und kann nur überlebt werden, wenn sie sich auf einem vollständig gebohrten Asteroiden mit hohlen Innenseiten befinden (sie verstecken sich).

Spiel vorbei: Das Spiel ist vorbei und die Spieler verlieren, wenn jeder Siedler stirbt.

2.2.3 Benutzern

Die Benutzer sollen keine extra, oder vorausgesetzte Kenntnisse über das Spiel haben, unser Ziel ist ein selbstverständige Benutzerschnittstelle und ein genüssliche Spiel am Ende der Entwicklungsprozess herstellen.

2.2.4 Begrenzungen

Das Spiel soll die definierte Anforderungen, und Spielscenarios realisieren. Es muss auch stabil sein, muss es so funktionieren, wie es erwartet werden kann. z.B: eine unerwartete Stillstand ohne Grund, oder ohne additionelle Information ist nicht erlaubt.

2.2.5 Voraussetzungen, Beziehungen

Der Benutzer soll nicht vollständig blöd sein, z.B: wenn eine Taste gedrückt werden soll, dann sei diese Taste nicht das Ein- und Ausschaltertaste der Maschine.

Aufgabebeschreibung: <https://www.iit.bme.hu/file/11582/feladat>

Unsere vorige Programmierung 3, und Softwaretechnologie Notizen, und aus diesem Fach wurde benutzt.

2.3 Anforderungen

2.3.1 Funktionale Anforderungen

Prioritäten:

- *gs: grundsätzlich*
- *wi: wichtig*
- *opt: optionell*

Anforderungen aus:

- *Besch.: Beschreibung (Aufgabenbeschreibung)*
- *TDR: Team Defined Requirement (Team definierte Anforderung)*

ID	Beschreibung	Kontrolle	Priorit ät	Quelle	Use-case	Komment
1.01	<i>Die Spieler steuern die Siedler</i>	Präsentation	gs	Besch.	Alle use-case	-
1.02	<i>Die Spieler und Roboter können von einem zu einen anderen Asteroid bewegen.</i>	Präsentation	gs	Besch.	Move Settler	-
1.03	<i>Die Spieler und Roboter können auf einem Asteroid bohren.</i>	Präsentation	gs	Besch.	Drill Asteroid	-
1.04	<i>Die Roboter helfen die Spieler.</i>	Auswertung	wi	Besch.	-	-
1.05	<i>Spieler können Roboter erzeugen.</i>	Präsentation	gs	Besch.	Create Robot	-
1.06	<i>Spieler können JumpGate-en herstellen.</i>	Präsentation	gs	Besch.	Manage JumpGate	-
1.07	<i>Asteroiden haben Mantelgröße, die verschieden sein können.</i>	Auswertung	wi	Besch.	-	-
1.08	<i>Ein Rohstoff befindet sich in den Kern einer Asteroid</i>	Präsentation	gs	Besch.	-	-
1.09	<i>Existieren gefährliche Asteroiden, die explodieren kann.</i>	Präsentation	gs	Besch.	-	-
1.10	<i>Existiert mindestens eine</i>	Präsentation	gs	Besch.	-	-

	<i>Asteroid, und ein Asteroidenfeld</i>					
1.11	<i>Es gibt mehrere Rohstoffe, die sind z. B: WasserEis, Eisen, Kohle, und Uran.</i>	Präsentation	gs	<i>Besch., TDR</i>	-	-
1.12	<i>Ein Asteroid hat ein Kern, die genau eine Rohstoff enthielt.</i>	Präsentation	gs	<i>Besch.</i>	-	-
1.13	<i>Gibt's Asteroiden, die keine Rohstoff enthalten, sondern Hohlraum.</i>	Präsentation	gs	<i>Besch.</i>	-	-
1.14	<i>In einem Schritt kann ein Astronaut (Spieler) nur eine Tätigkeit durchführen</i>	Auswertung	wi	<i>Besch.</i>	<i>Alle use-case</i>	-
1.15	<i>Andere mögliche Tätigkeiten einer Spieler sind: bewegen, bohren, fördern, JumpGate aufbauen und Resource ablegen</i>	Präsentation	gs	<i>Besch.</i>	<i>Move Settler, Drill Asteroid, Mine Asteroid, Manage JumpGate, Create Robot</i>	-
1.16	<i>Möglichkeiten sind noch die Nachbarnasteroiden auslisten.</i>	Präsentation	wi	<i>TDR</i>	-	<i>Nützlich kann sein beim Bewegen.</i>
1.17	<i>Spieler kann eine bestimmte Menge der Ressourcen mitnehmen/transporieren.</i>	Präsentation	gs	<i>Besch.</i>	<i>Move Settler</i>	<i>Diese Menge wird wahrscheinlich 10. (integriert im Move Settler use-case)</i>
1.18	<i>Jede Asteroid hat beliebig viele Nachbarn.</i>	Präsentation	wi	<i>Besch., TDR</i>	-	<i>Es kann mehrere Hunderte sein, aber mindestens 1.</i>

1.19	<i>Beim Bohren wird das Loch mit einer Einheit tiefer.</i>	Auswertung	gs	Besch.	Drill Asteroid	<i>Loch wird so modelliert, dass die Manteldicke kleiner wird</i>
1.20	<i>Fördern ist dann und nur dann möglich, wenn das Loch fertig ist</i>	Auswertung	gs	Besch.	Mine Asteroid	<i>Loch ist fertig, wenn der Mantel ist völlig durchbohrt.</i>
1.21	<i>Asteroiden, JumpGate-n, und Sonne hat Position im universum</i>	Auswertung	opt	TDR	-	-
1.22	<i>Asteroid kann in der Sonnennähe, oder fern von der Sonne liegen.</i>	Auswertung	wi	Besch.	-	-
1.23	<i>Wenn in der Sonnennähe liegende Asteroid radioaktive Rohstoff gebohrt wurde, Asteroid explodiert.</i>	Auswertung/Präsentation	gs	Besch.	Mine Asteroid	-
1.24	<i>Gibt's Solarflair, das die Astronauten, und Roboter tötet, wenn sie nicht im Hohlraum einer Asteroid sind</i>	Präsentation	gs	Besch.	-	-
1.25	<i>Roboter sind von dem Spiel kontrolliert.</i>	Präsentation	wi	Besch.	-	-
1.26	<i>Roboter überleben die radioaktive Explosion, aber sie werden auf einer Nachbarasteroid geworfen.</i>	Präsentation	gs	Besch.	-	-
1.27	<i>Die Spieler haben gewonnen, wenn aus alle Rohstoffe mindestens 3</i>	Präsentation	gs	Besch.	-	-

	<i>Einheit gefördert, und diese Rohstoffe auf einer Asteroid transportiert wurde.</i>					
1.28	<i>Das Spiel hat zwei Szenarien zum Spielende: Gewinnen, und Verloren</i>	Präsentation	gs	Besch.	-	-
1.29	<i>Die Spieler haben verloren, wenn die allen tot sind.</i>	Präsentation	gs	Besch.	-	-
1.30	<i>Um eine Robot herzustellen muss man bestimmte Ressourcen bezahlen.</i>	Präsentation	wi	Besch.	Create Robot	1 Stahl, 1 Kohl, 1 Uran
1.31	<i>Mehrere Spieler können spielen.</i>	Präsentation	wi	TDR	-	1 und mehrere
1.32	<i>Eine Asteroid kann eine Spieler oder Robot von einem Solarflair schützen, wenn es Hohlraum hat.</i>	Präsentation	wi	TDR	-	-
1.33	<i>Wenn der Astronaut bewegt sich, er kann aus den Asteroiden wählen, dass sie auf welchem Asteroid fahren möchte.</i>	Präsentation	wi	TDR	Move Settler	-
1.34	<i>Ein Astronaut kann aus dem Spiel austreten</i>	Präsentation	opt	TDR	Leave Game	-
1.35	<i>Beim Fördern wird das ganze Rohstoffmenge aufgefordert (eine Einheit aus dem Rohstoff)</i>	Auswertung	wi	Besch.	Mine Asteroid	-
1.36	<i>Das Spiel soll die Astronauten über einem Solarflair vor dem Solarflair irgendwie berichten.</i>	Präsentation	opt	TDR	-	-

1.37	<i>Solarflairs können den Asteroidzone erreichen</i>	Präsentation	wi	Besch.	-	Nicht immer.
1.38	<i>Roboter können nicht fördern.</i>	Auswertung	wi	Besch.	-	-
1.39	<i>Nur eine Sonne existiert in dem Spiel</i>	Präsentation	wi	TDR	-	wichtig!
1.41	<i>Ein Spieler kann maximal 2 JumpGate mitnehmen</i>	Präsentation	gs	Besch.	-	-
1.42	<i>Um eine JumpGate herzustellen muss man bestimmte Ressourcen bezahlen.</i>	Präsentation	wi	Besch.	Manage JumpGate	2 Stahl, 1 WasserEis und 1 Uran.
1.43	<i>Gibt's ein HomeAsteroid, die größere Speicherkapazität hat.</i>	Präsentation	wi	TDR	-	unendlich e Speicherkapazität (wahrscheinlich)

2.3.2 Anforderungen über Ressourcen

ID	Beschreibung	Kontrolle	Priorität	Quelle	Komment
2.01	<i>Programm wird auf Java geschrieben</i>	Auswertung	gs	Besch.	-
2.02	<i>Existenz von JVM</i>	Auswertung	gs	Besch./ TDR	
2.03	<i>Windows 10, MAC OS X Betriebssysteme empfohlen</i>	Präsentation	opt	TDR	<i>Diese benutzen wir in der Gruppe</i>
2.04	<i>Maus, Tastatur und Bildschirm angeschlossen</i>	Präsentation	wi	TDR	
2.05	<i>Berechtigungen über Dateiverwaltung</i>	Präsentation	wi	TDR	

2.3.3 Anforderungen über Übergabe

ID	Beschreibung	Kontrolle	Priorität	Quelle	Komment
3.01	<i>Grundanforderungen sind erfüllt</i>	<i>Auswertung</i>	<i>gr</i>	<i>TDR</i>	-
3.02	<i>Notwendige Softwares sind installiert</i>	<i>Auswertung</i>	<i>gr</i>	<i>TDR</i>	-

2.3.4 Andere nicht funktionale Anforderungen

ID	Beschreibung	Kontrolle	Priorität	Quelle	Komment
4.01	<i>Ausführbare Datei mit kleinen Größe</i>	<i>Auswertung</i>	<i>opt</i>	<i>TDR</i>	<i>Wegen Portabilität</i>
4.02	<i>Die Software soll fähig sein mehrere Spieler behandeln</i>	<i>Präsentation</i>	<i>wi</i>	<i>Besch. und TDR</i>	-
4.03	<i>Die Software muss auf Windows und MAC OSX laufen</i>	<i>Auswertung</i>	<i>wi</i>	<i>TDR</i>	<i>Unser Team arbeitet auf Win und Mac.</i>

2.4 Wesentliche use-case Diagramme

2.4.1 Use-case Beschreibungen

Name des Use-case	Start game
Kurzbeschreibung	Das Spiel wird gestartet.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt START GAME Taste. 2. Spieler wartet 3. Spiel fängt an.

Name des Use-case	Leave game
Kurzbeschreibung	Der Spieler tritt aus dem Spiel. Das Spiel endet.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt LEAVE GAME Taste. 2. Das Spieler beendet das Spiel.

Name des Use-case	Pause game
Kurzbeschreibung	Der Spieler kann eine Spielpause einlegen. Das Spiel stoppt, die Ergebnisse bleiben erhalten, bis der Spieler die Pause unterbricht.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt PAUSE GAME Taste. 2. Das Spiel stoppt.

Name des Use-case	Move settler
Kurzbeschreibung	Der Spieler leitet einen Siedler in dem Solarsystem.
Aktoren	Player
Drehbuch	Der Siedler bewegt auf dem nächsten benachbarten Asteride.

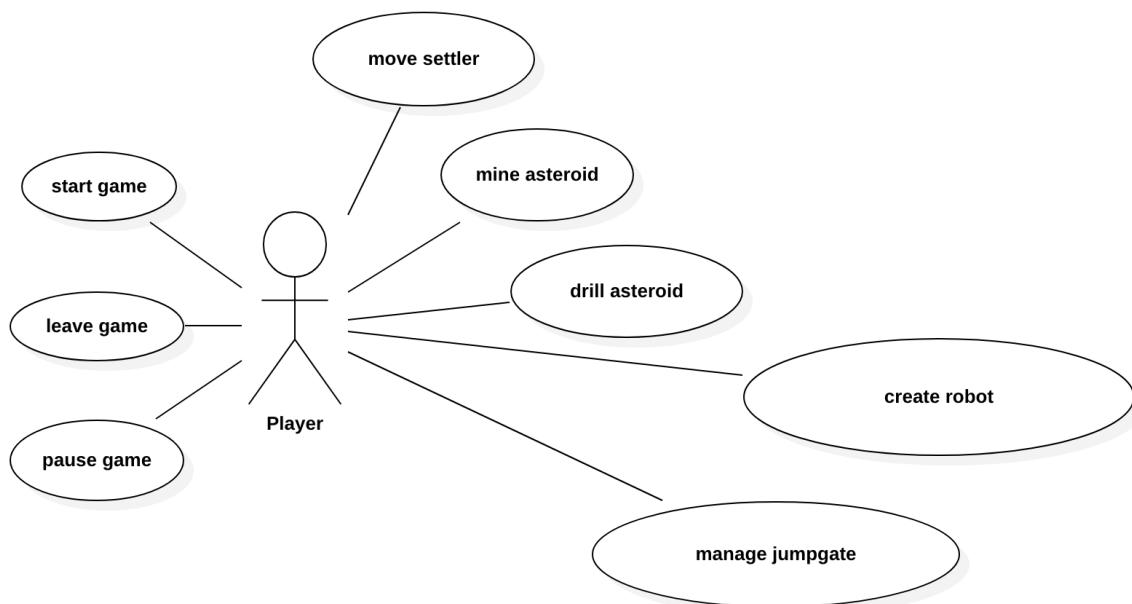
Name des Use-case	Mine asteroid
Kurzbeschreibung	Der Siedler fördert den Rohstoff.
Aktoren	Player
Drehbuch	Der Siedler fördert der Ressource aus dem Kern der Asteroide.

Name des Use-case	Drill asteroid
Kurzbeschreibung	Der Spieler bohrt den Mantel der Asteroide.
Aktoren	Player
Drehbuch	Der Spieler während des Bohrens macht den Mantel der Asteroide 1 Einheit kleiner.

Name des Use-case	Manage jumpgate
Kurzbeschreibung	Der Spieler handhabt das Teleporttor.
Aktoren	Player
Drehbuch	Mit zwei Einheiten Eisen, einer Einheit Wassereis und einer Einheit Uran kann der Spieler ein Paar Teleport Tore herstellen. Jedes Tor kann später vom Spieler in der Nähe des gerade besuchten Asteroiden gestartet werden. Die beiden Mitglieder des Tor Paars sind verbunden und betreten eines, der Spieler befindet sich im anderen. Frisch hergestellte Tore können vom Spieler getragen werden, ein Spieler kann jedoch maximal zwei Tore gleichzeitig haben..

Name des Use-case	Create robot
Kurzbeschreibung	Der Spieler baut einen Roboter
Aktoren	Player
Drehbuch	Siedler können mit einer Einheit Eisen, einer Einheit Kohle und einer Einheit Uran einen autonomen Roboter schaffen, der durch künstliche Intelligenz gesteuert wird. Spieler können die Roboter nicht kontrollieren.

2.4.2 Use-case diagram



2.5 Wörterbuch

Asteroide: Auf einem Besetztes Feld liegende Element, aus denen der Spieler Rohstoffe fördern kann.

Home Asteroide / Basis Asteroide: Ein solcher Asteroide auf den die Rohstoffe gesammelt werden um dort das endgültige Basis aufzubauen.

Basis: Um das Spiel gewinnen zu können muss es aufgebaut werden aus den bestimmten Rohstoffen.

Besetztes Feld: Ein solcher Einheit auf der Spielfeld, auf denen eine Astroide ist.

Bewegen: In einem Schritt kann ein Spieler dieser Tätigkeit machen. In diesem Fall bewegt er sich auf den benachbarten Astroide.

Bohren: In einem Schritt kann ein Spieler dieser Tätigkeit machen. In diesem Fall wird die Manteldicke der Astroide mit 1 verringert. Falls es 0 ist dann kann der Rohstoff gefördert werden.

Einheit: Für die Rohstoffe bedeutet, wie viel aus ihnen zur Verfügung steht.

Explosion: Falls es radioaktiver Rohstoff in Sonnennah gefördert wird wird es explodieren und den Spieler töten.

Fördern: In einem Schritt kann ein Spieler dieser Tätigkeit machen. In diesem Fall wird aus der Kern der Astroide der dort liegende Rohstoff gefördert.

Gewinnen: Falls die Spieler aus allen Rohstoffen 3 Einheiten gesammelt haben dann haben sie gewonnen.

Kern: In der Mitte der Astroide ist der Kern aus dem Rohstoff gefördert werden kann. Es kann sogar leer oder radioaktiv sein.

Leeres Feld: Ein solcher Einheit auf der Spielfeld, auf denen keine Astroide oder andere Sache ist.

Manteldicke: Es zeigt wie dick der Mantel der Astroide ist. Beim Bohren wird es mit 1 verringert.

Nachbarn: Die Asteroide haben nachbarn in bestimmten Radius.

Programm: Das fertig gemachte und entwickelte Endergebnis.

Radioaktiv: Ein Rohstoff kann radioaktiv sein. In diesem Fall wenn die Astroide in Sonnennah ist dann wird es explodieren.

Raumschiff: Eigentlich äquivalent mit dem Spieler. Es hat keine spezielle Bedeutung.

Robot: Ein Robot mit AI kann bewegen, bohren und wird nicht explodieren in Sonnennähe.

Robot bauen: In einem Schritt kann ein Spieler dieser Tätigkeit machen. In diesem Fall kann er aus den bestimmten Rohstoffen ein Robot mit AI bauen.

Rohstoff: Es befindet sich in dem Kern der Asteroiden. Aus denen können Roboter, Portals, Basis gebaut werden.

Schritt: In einem Schritt kann der Spieler Tätigkeiten durchführen. zB: Fördern, Bewegen, Bohren, Portal und Roboter bauen.

Siedler: siehe Spieler.

Sonnenflair/Solarflair: ein Sturm von der Sonne und kann die Spieler töten falls sie nicht in einem leeren Astroide sind.

Sonnennah: Es bedeutet dass die Spieler nah zu der Sonne sind und falls ein radioaktiver Rohstoff aus ihnen gefördert wird dann wird es explodieren.

Spieler: Der Person der den Spiel spielt. Die Siedler leitet und Tätigkeiten durchführt.

Spielfeld : Den ganzen Feld auf denen der Spiel passiert. Auf ihm sind Spieler, Asteroiden, Roboter, Portals.

Sterben: Falls der Spieler explodiert oder ungeschützt von einem Sonnenflair ist dann wird er sterben und nicht mehr spielen.

Teleport Portal: Ein Spieler kann max 2 bei ihm haben. Aber kann mehrere Portal Paare haben. Dadurch kann man teleportieren. Es kann der Spieler für bestimmte Rohstoffe bauen.

Teleport Portal bauen: In einem Schritt kann ein Spieler dieser Tätigkeit machen. In diesem Fall wird ein Teleport Portal gebaut. Siehe Teleport Portal.

Töten: Ein Sonnenflair oder eine Explosion kann die Spieler vernichten/töten also sie können das Spiel nicht fortsetzen.

Verlieren: Falls alle Spieler gestorben sind dann ist der Spiel vorbei und sie haben verloren.

2.6 Projekt Plan

Schritte und ihre Abgabetermine

- 02.10. Informationsvortrag
- 02.17. Anforderungen, Projekt, Funktionalität
- 02.24. Analyse Modell 1.
- 03.03. Analyse Modell 2.
- 03.10. Skeleton planen
- 03.17. Skeleton
- 03.24. Konzeption von Prototyp
- 03.31. Detaillierte Pläne
- 04.07. Frühling Pause
- 04.14. Prototyp
- 04.21. Spezifikation von GUI
- 04.28. GUI erstellen
- 05.05. GUI und Zusammenfassung
- 05.12. Buffer, Wiederholen

Verantwortungen

Ábel Borbély	Dokumentation, Kodierung, Testen des Kodes
Henrietta Domokos	Dokumentation, Kodierung, UML (Use-Case), Drive, PrettyPrinter
Renátó Hrotkó	Dokumentation, Kodierung, Graphische Oberfläche,
Vince Pongrácz	Dokumentation, Kodierung, UML (Class), Repository

Techniken

Google Docs / Google Drive - Es ist verwendet um das Dokumentation zu schreiben. Alle Dokumente sind auf einem Google Drive geteilt zu denen alle zugreifen können. Alle können hier gleichzeitig schreiben und arbeiten und die vorherige Versionen sind auch zurückzustellen.

Slack - Es benutzen wir um das Kommunikation der Mitglieder zu schaffen. Stetige Kommunikation geschieht hier fast jeden Tag wann wir bestimmte Problemen und ihre Lösungsmethoden besprechen und versuchen immer sie Beste zu wählen. Die 2 Gruppen der Deutschsprachigen Ausbildung haben 2 unterschiedliche Channels um dort arbeiten zu können.

Discord - 2-mal pro Woche haben Sitzungen wann wir besprechen wer was machen wird. Wir versuchen alle Aufgaben gerecht verteilen auch gemäß der einzelnen Ansprüche. Auf unserem Discord Server können wir alles und schnell besprechen und die Dokumente verteilen wir in Slack oder in Drive.

Git - Zum Versionskontroll werden wir Git benutzen. Auf GitHub haben wir einen geteilten Repository wohin wir alle Dateien hochladen werden. Da für eine Team sehr wichtig das Zusammenarbeit ist deswegen wenn etwas schief gegangen ist zB in dem Kode dann mit Git können wir das einfach zurückstellen und dann die Fehler korrigieren und dann fortsetzen.

IntelliJ - das Kodieren passiert in diesem Umgebung. Das haben wir in den letzten Semester benutzt und läuft auf Mac OSX und Windows. und mit Git kann das leicht zusammengebunden werden.

Star UML - UML Use-Case-, Klassen-, Sequenzdiagramme herstellen. Vorher haben wir WhiteStar UML benutzt aber das war schlecht zu benutzen und war auch sehr alt. Deswegen benutzen wir Star UML weil es leichter zu benutzen ist und eine viel bessere Benutzeroberfläche hat.

Adobe Photoshop - Um die Bilder des Projektes herzustellen. Bilder über Asteroide, Rohstoffe, Raumschiff, die Felder, Basis usw. werden in Photoshop hergestellt.

2.7 Tagebuch

Anfang	Zeitdauer	Teilnehmer	Beschreibung
2021.02.14. 17:00	5 Stunde	Borbély Domokos Hrotkó Pongrácz	Anfangsmeeting: Aufgaben durchlesen, verteilen und interpretieren, offene Fragen besprechen. Organisationale Fragen beantworten. (Git, Repository, Planungs- und Entwicklungssoftware)
2021.02.14	cirka 3 Stunde	Domokos	Tätigkeit: Ziel aufschreiben, Vorige Diagrammen durchschauen, Use-case herstellen und beschreiben
2021.02.15.: ● 8:30-10:00, ● 16:45-19:00, ● 22:00-23:50	cirka. 5,5 Stunde	Pongrácz	Tätigkeit: Vorige Diagrammen durchschauen, neue UML Klassendiagramm Modell herstellen, Anforderungen klar machen und auf schreiben.
2021.02.15.	insg. 5 Stunde	Hrotkó	Einführung, Projektplan, Wörterbuch schreiben und übersetzen. Besprechen von UML modelle. Gedanken machen über die Realisation des Spiels.
2021.02.16. 17:00	3 Stunden	Borbély	Allgemeiner Überblick, Funktionen umschreiben
2021.02.16. 20:00	1 Stunde	Hrotkó Pongrácz Domokos Borbély	Besprechen und Überprüfen der Aufgabe, letzte Berührungen
2021.02.19 21:10	cirka 0.5 Stunde	Pongrácz	Planung (UML Class), und Fehlerkorrekturen
2021.02.16.	0.5 Stunde	Domokos	Use-case finalisieren
2021.02.16. 21:15	0.5 Stunde	Hrotkó	Beenden der ausgebliebene Teile des Dokumentes und Übersetzen.

3. Entwicklung des Analysemodells

2 – Halooo Matematikeer

Konzulent:
Márton Kovács

Mitglieder

Henrietta Domokos	J0DCPT	domoheni@gmail.com
Ábel Borbély	DRP0DT	babel1122@gmail.com
Renátó Hrotkó	OIT6HD	renatohrotko@gmail.com
Vince Pongrácz	MKMDJO	pongvin@gmail.com

17.02.2021

3. Entwicklung des Analysemodells

3.1 Objekten Kataloge

3.1.1 Asteroid

Daraus können die Spieler Ressourcen fördern nachdem ihre Mantel durchgebohrt wurde. In Sonnennah kann es explodieren falls radioaktive Ressource enthält und der Spieler es versuchen hat durchzubohren.

3.1.2 Ressources

Es enthält die Typen der möglichen Ressourcen die während des Spiels gefördert werden können. Es ist auch gewusst ob es radioaktiv ist oder nicht.

3.1.3 Core

In einem Asteroide ist ein Core/Kern der enthält immer ein Ressource. In dem Kern ist immer nur ein Ressource außer der Basisasteroide der viel mehr speichern kann weil dort wird der Basis aufgebaut.

3.1.4 Layer

Jede Asteroide hat ein Mantel, dessen Dicke veränderlich ist. Beim bohren wird es gebohrt um das Rohstoff in dem Kern erreichen zu können. Beim Bohren wird es immer 1 Einheit kleiner.

3.1.5 Position

Es hat immer eine x und y Koordinate mit denen die Positionen der Entitäten, Asteroiden und der Sonne bestimmt ist. Asteroiden haben auch ein Radius drin mit dem bestimmt ist, wie nah andere Asteroiden sein können.

3.1.6 Entity

Die Spieler haben Entitäten zB: Siedler und AIRoboter. Ein Entität kann sich bewegen zwischen Asteroiden, bohren die Mantel der Asteroiden, sterben bei einer Explosion (außer Roboter) und Sonnenflair, die Nachbarn behandeln.

3.1.7 AIRoboter

Spieler können aus bestimmten Ressourcen Roboter herstellen die beim Bohren helfen können. Sie überleben den Explosion und landen auf einem benachbarten Asteroide. Sie sind von dem GameController kontrolliert also der Spieler kann mit ihnen nichts tun. Gleichzeitig kann ein Spieler mehr besitzen.

3.1.8 Settler

Spieler können sogar mehrere Siedler haben und mit ihnen können sie spielen. Sie können max 10 Ressourcen haben. Sie können noch fördern aus Asteroiden. Portals herstellen und bauen, Ressourcen zurücksetzen in Asteroiden falls sie leer sind.

3.1.9 Player

Die Spieler haben immer ein oder mehr Siedler und können mehrere Roboter besitzen. Sie haben auch einen Namen der vor dem Spiel eingestellt wird.

3.1.10 Sun

In einem Asteroidenzone ist immer eine Sonne. Sie kann ein Solarflair machen, das die Siedler und Roboter nur in einem leeren Asteroide überleben können.

3.1.11 Gate

Spieler können über Portale verfügen um nach einem anderen Ort zu teleportieren. Ein Spieler kann gleichzeitig immer nur ein Portalpaar dabei haben.

3.1.12 AsteroidZone

Es kann die ganze Asteroidenzone herstellen. Also die Spieler, Sonne, Asteroiden positionieren mit zufälliger Verteilung bei den Asteroiden.

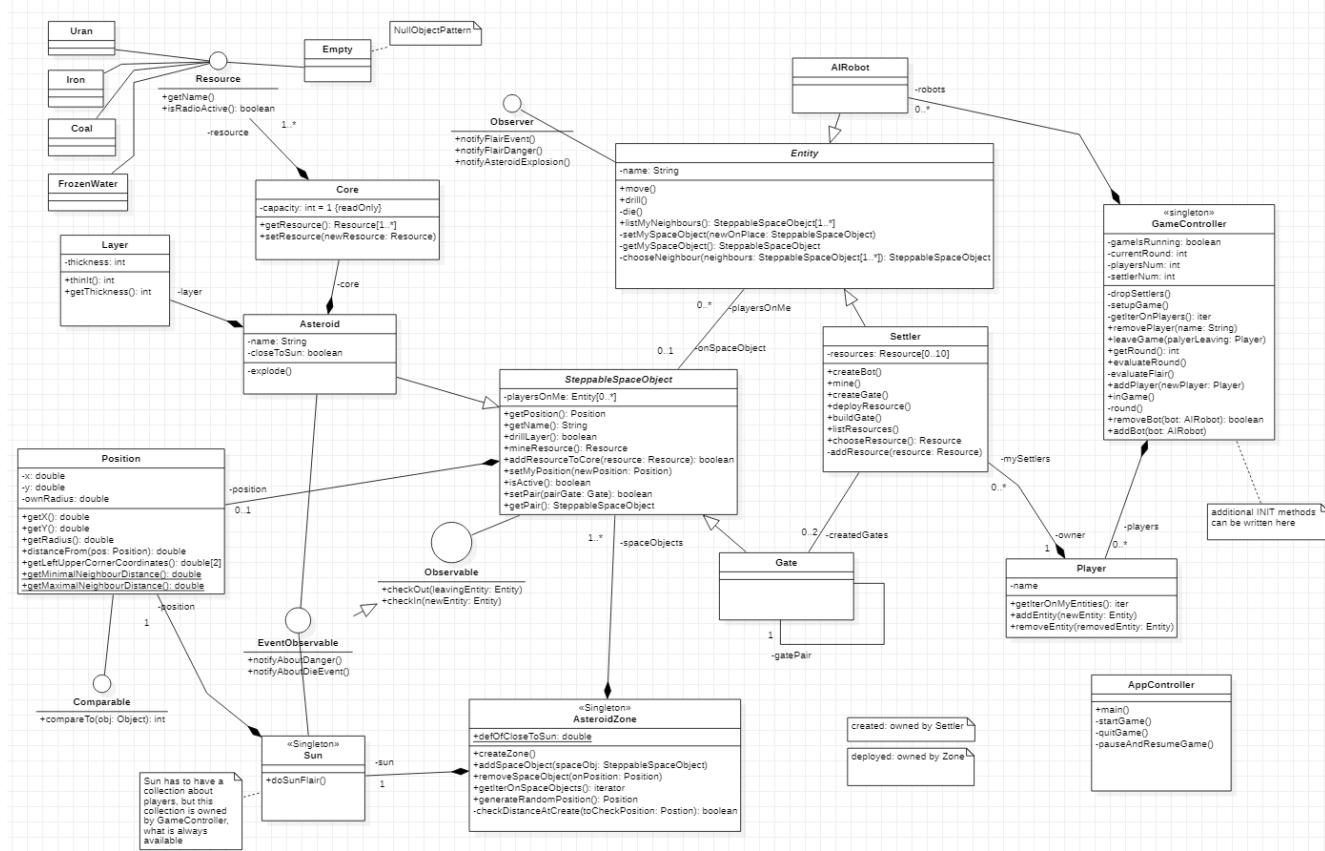
3.1.13 GameController

Es kontrolliert den ganzen Spiel. Die Runde die nacheinander kommen sind hier behandelt und am Ende ausgewertet. Die ARoboter sind auch von ihm kontrolliert. Es kann den Spiel einstellen.

3.1.14 AppController

Ein AppController kann den ganzen Spiel laufen lassen, während dem Spiel pausieren und dann fortsetzen und falls der Spieler möchte dann den ganzen Spiel sofort beenden.

3.2 Statische Struktur Diagramme



3.3 Beschreibung der Klassen

3.3.1 AIRobot

- **Verantwortung**

Der AIRobot bohrt, bewegt sich und hilft den Spielern, ihr Ziel zu erreichen.

- **Basisklasse**

Entity → AIRobot

3.3.2 AppController

- **Verantwortung**

Diese Abteilung ist für die Ausführung des Spiels verantwortlich. Startet, stoppt oder pausiert das Spiel.

- **Methoden**

- **main()**: Ein Programm muss eine globale Funktion namens main enthalten, die den festgelegten Start des Programms darstellt.

3.3.3 Asteroid

- **Verantwortung**

Der Asteroid speichert Rohmaterial in seinem Kern. Er kann Astronauten schützen, wenn sie sich in seinem Kern verstecken. Das Asteroid kann explodieren. Das Rohmaterial des Kerns kann abgebaut werden..

- **Ősosztályok**

SteppableSpaceObject → Asteroid

- **Schnittstellen / Interface**

- EventObservable
- Observable

- **Attribute**

- **String name:** Der Name des Asteroiden.
- **boolean closeToSun:** Ist der Asteroid in der Nähe des Sonne.

- **Methoden**

- noch überschreibende Methoden von der abstrakten Basisklasse (SteppableSpaceObject)
- **explode():** innere Method um Explosion zu behandeln

3.3.4 AsteroidZone

- **Verantwortung**

Verwaltet das Spielfeld.

- **Attribute**

- **int defOfCloseToSun:** Wenn Sie diese Zahl mit einem SteppableSpaceObject vergleichen, können Sie feststellen, ob es sich in der Nähe der Sonne befindet

- **Methoden**

- **createZone():** Es schafft das Spielfeld vor dem Spielstart.
- **addSpaceObject(spaceObj: SteppableSpaceObject):** Fügt der Asteroidenzone einen SteppableSpaceObject hinzu.
- **removeSpaceObject(onPosition: Position):** Entfernt ein Objekt in einer Asteroidenzone von einer bestimmten Position.
- **iterator getIterOnSpaceObjects():** Wenn ich die Nachbarn auflisten möchte, gibt diese Funktion eine Liste aller SpaceObjects und ich kann basierend darauf weiter filtern.
- **Position generateRandomPosition():** Erstellt Positionen durch Generieren von Zufallszahlen. Erforderlich, um die Strecke zu bauen.

3.3.5 Core

- **Verantwortung**

Es ist verantwortlich für die Speicherung der Rohstoffe innerhalb des Asteroiden

- **Attribute**

- **int capacity:** Gibt an, wie viele Rohstoffe sich im Kern befinden können.

- **Methoden**

- **Resource[1...*] getResource:** Es kehrt mit dem Rohmaterial des Kerns zurück.
- **setResource(newResource: Resource):** Setzt den Kernrohstoff auf eine bestimmte Ressource.

3.3.6 Entity

- **Verantwortung**

Abstrakte Basisklasse für Roboter und Siedler.

- **Schnittstellen / Interface**

- Observer → Entity

- **Attribute**

- **String name:** Name des Entity.

- **Methoden**

- **move():** Durch Aufrufen dieser Funktion werden die Entitäten verschoben.
- **drill():** Durch Aufrufen dieser Funktion werden die Entitäten bohren.
- **SteppableSpaceObject[] listMyNeighbours():** Gibt die Nachbarn des Asteroiden zurück, auf dem sich die Entität befindet.

3.3.7 EventObservable

- **Verantwortung**

Wenn ein Asteroid explodiert oder eine Sonneneruption gibt, benachrichtigt SteppableSpaceObject.

- **Methoden**

- **notifyAboutDanger():** Informiert alle aufgeschriebene Entität über die Gefahr von Solarflair.
- **notifyAboutDieEvent():** Informiert alle aufgeschriebene Entität über den Tod, und das Grund darauf.

3.3.8 GameController

- **Verantwortung**

Verwaltet den Verlauf des Spiels. Vom Start zum Ende.

- **Attribute**

- **boolean gameIsRunning:** Sagt dir, ob das Spiel läuft.
- **int currentRound:** Die Nummer der aktuellen Runde des Spiels.
- **int playersNum:** Anzahl der Spieler. Etwas, das während des Setups konfiguriert werden kann.
- **int settlerNum:** Anzahl der Siedler. Etwas, das während des Setups konfiguriert werden kann.

- **Methoden**

- **removePlayer(name: String):** Spieler aus dem Spiel entfernen.
- **leaveGame(playerLeaving: Player):** Wenn ein Spieler aufhören möchte, gibt er auf, tötet im Wesentlichen alle seine Siedler und löscht sie auch aus den Spielern.
- **int getRound():** Gibt das Anzahl der aktuelle Runde im Spiel zurück.
- **evaluateRound():** Wertet den gegebenen Kreis aus. Überprüfen Sie, ob alle Siedler des Spielers gestorben sind, und ob die Spieler gewonnen oder verloren haben.
- **addPlayer(newPlayer: Player):** Fügt dem Spiel einen neuen Spieler hinzu.
- **inGame():** Eine fast unendliche Schleife, in dieser Funktion läuft das Spiel und das Spiel kreist als Endlosschleife. Wir rufen diese Funktion auf, indem wir das Spiel starten und am Ende des Spiels beenden.
- **boolean removeBot(bot: AIRobot):** Entfernt den Roboter. Der Boolesche Wert gibt an, ob Sie ihn entfernt haben.
- **addBot(bot: AIRobot):** Fügt dem Spiel einen neuen AIRoboter hinzu.

3.3.9 Gate

- **Verantwortung**

Kann von Siedlern erstellt und gespeichert werden. Ermöglicht das Reisen zwischen einem Paar von ihnen.

- **Basisklasse**

SteppableSpaceObject → Gate

- **Attribute**

- **Gate gatePair:** Der Paar eines Portals.

- **Methoden**

nur überschreibende Methoden von der abstrakten Basisklasse.

3.3.10 Layer

- **Verantwortung**

Es stellt die Kortikalis/Kruste/Layer des Asteroiden dar, weiß, wie dick er ist und kann ihn reduzieren.

- **Attribute**

- **int thickness:** Die Dicke der Kruste des Asteroiden.

- **Methoden**

- **int thinIt():** Es verdünnt sich. Der DrillLayer ruft Sie an.
- **int getThickness():** Der Asteroid kehrt mit der Dicke seiner Kortikalis zurück.

3.3.11 Player

- **Verantwortung**

Hilft bei der Verwaltung der Entitäten eines Spielers.

- **Attribute**

- **String name:** Der Name des Spielers.
- **Settler[0...*] mySettlers:** Die Kollektion der Siedler, die zu einem Spieler gehören.

- **Methoden**

- **Iterator getIterOnMyEntities():** Erzeugt einen Iterator für die Kollektion der Siedler eines Spielers
- **addEntity(newEntity: Entity):** Weist einem Spieler eine neu Entität zu.
- **removeEntity(removedEntity: Entity):** Hebt die Zuordnung einer Entität zu einem Spieler auf

3.3.12 Position

- **Verantwortung**

Ordnet Objekten im Feld eine bestimmte Position zu.

- **Schnittstellen / Interface**

Comparable

- **Attribute**

- **double x:** Koordinate auf der x-Achse.
- **double y:** Koordinate auf der y-Achse
- **double ownRadius:** Hilft überlappende Objekte zu vermeiden.

- **Methoden**

- **double getX():** Gibt die x-Koordinate zurück.
- **double getY():** Gibt die y-Koordinate zurück.
- **double getRadius():** Gibt den Radius des zugehörigen Objekts zurück.

- **double distanceFrom(pos: Position):** Kalkuliert die Distanz zweier Positionen.
- **double[2] getLeftUpperCornerCoordinate():** Hilft mit dem korrekten Platzieren der Objekte.
- **double getMinimalNeighbourDistance():** Untere Schranke für die Zufallsdistanz, in der die Objekte benachbart sind.
- **double getMaximalNeighbourDistance():** Obere Schranke...

3.3.13 Settler

- **Verantwortung**

Siedler sind die Entitäten, die die Spieler bewegen und über sie mit dem Spielfeld interagieren können.

- **Basisklasse**

Entity → Settler

- **Attribute**

- **Resources[0...10] resources:** Speichert die abgebauten Ressourcen eines Siedlers.
- **Player owner:** Der Spieler, dem die Siedler gehören.
- **Gate[0...2] createdGates:** Die Kollektion der erstellten Portale.

- **Methoden**

- **createBot():** Erschafft einen AI Roboter.
- **mine():** Entfernt die Ressource aus dem Asteroidenkern, und fügt es seiner eigenen Kollektion von Ressourcen zu.
- **createGate():** Erstellt ein Paar Portale, und fügt sie seiner eigenen Kollektion von Portalen zu.
- **deployResource():** Setzt eine Ressource wieder in einen Asteroiden ein.
- **buildGate():** Platziert eines der getragenen Portale.
- **listResources():** Listet die Ressourcen auf, die der Siedler besitzt.
- **Resource chooseResource():** Der Siedler wählt eine Ressource aus seinem Inventar aus.
- **addResource(resource: Resource):** Fügt Ressource zu der Kollektion .

3.3.14 SteppableSpaceObject

- **Verantwortung**

Diese sind Objekte, auf die die Spieler mit ihren Siedlern treten können.

- **Schnittstellen / Interface**

Observable

- **Attribute**

- **Entity[0...*] playersOnMe:** Speichert die Entitäten, die auf einem solchen Objekt stehen.

- **Methoden**

- **getPosition()**: Gibt die Position des Objektes zurück.
- **String getName()**: Gibt den Namen des SteppableSpaceObjectes zurück, falls es existiert.
- **boolean drillLayer()**: Falls es eine Asteroid ist, reduziert die Größe der Kortikalis des Asteroiden um eins. Übergibt die Bohraufgabe an das Layer.
- **Ressource mineResource()**: Falls es eine Asteroid ist, baut es das Innere des Asteroiden ab und setzt die Art des Rohmaterials auf leer. Leitet die Mining-Aufgabe an das Core weiter.
- **addResource(ressource: Resource)**: Übergibt die Aufgabe der Rohstoffzugabe an das Core, falls es eine solche Objekt ist, welches ein Core (Kern) hat.
- **setMyPosition(newPosition: Position)**: Setzt die Position eines Portals, falls solche Operation interpretierbar ist.
- **boolean isActive()**: Gibt an, ob ein Portal, oder Objekt aktiv ist, also ob es wirklich funktioniert.
- **boolean setPair(pairGate: Gate)**: Bindet das aktuelle Objekt mit einem Anderen zu.
- **SteppableSpaceObject getPair()**: Gibt das Paar eines Portals/SteppableSpaceObject zurück.

3.3.15 Sun

- **Verantwortung**

Startet Sunflairs, was gefährlich für Siedler und Roboter sind.

- **Schnittstellen / Interface**

EventObservable

- **Attribute**

- **Position position**: Position der Sonne auf dem Spielfeld.

- **Methoden**

- **doSunFlair()**: Jede Entität auf dem Feld wird überprüft, ob sie in Gefahr ist, und wenn ja, stirbt sie.

3.3.16 Observable

- **Verantwortung**

Diese Schnittstelle hat Operationen, dadurch Entitäten (Siedler und Roboter) zur Kollektion von SteppableSpaceObjects hinzugefügt werden können.

- **Methoden**

- **checkOut(leavingEntity: Entity)**: Das SteppableSpaceObject entfernt die Entität von sich.
- **checkIn(newEntity: Entity)**: Das SteppableSpaceObject registriert eine ankommende Entität an sich.

3.3.16 Observer

- **Verantwortung**

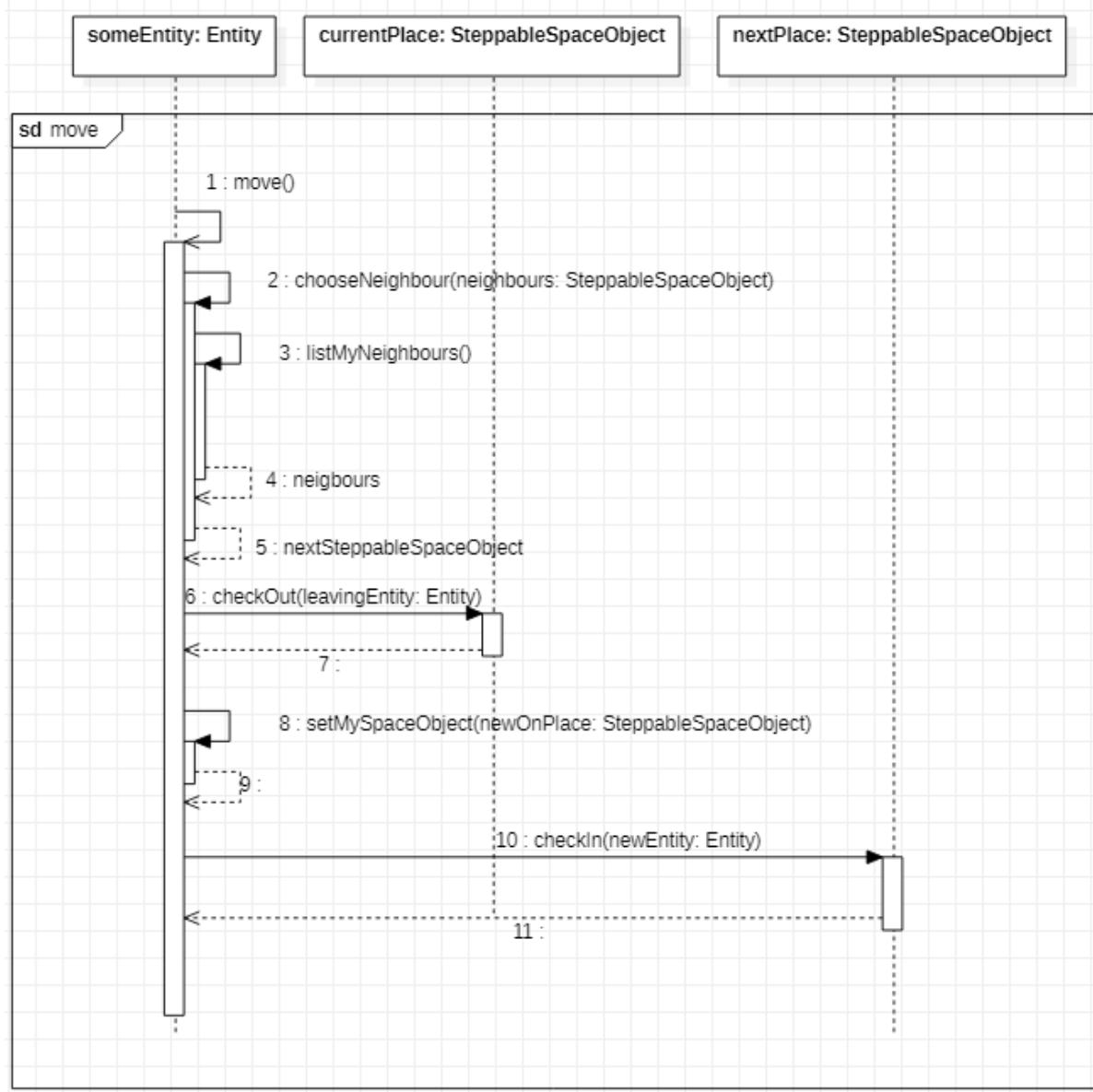
Diese Schnittstelle hat das Aufgabe, dass die Entitäten auf SolarFlair, oder Asteroidexplosion reagieren. Diese Methoden der Interface hat die Kenntnisse, wie das aktuelle Entität das Ereignisse behandelt.

- **Methoden**

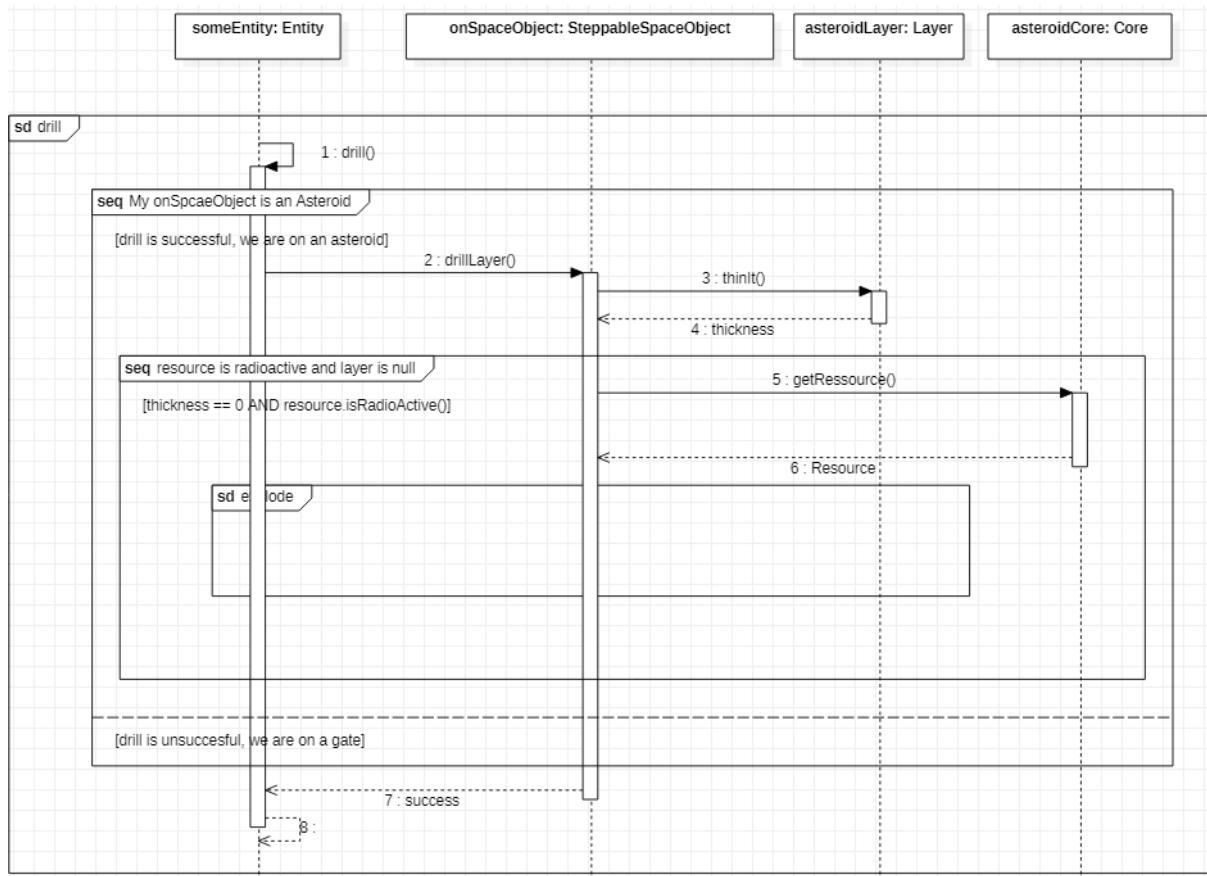
- **notifyFlairEvent()**: Benachrichtigt die Entität (und Spieler auch) über einem FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt.
- **notifyFlairDanger()**: Benachrichtigt die Entität (und Spieler auch) über einem bevorstehende FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt. Often nur ein Message auf dem Bildschirm, über die zukünftige Solarflair.
- **notifyAsteroidExplosion()**: Benachrichtigt die Entität (und Spieler auch), dass ihre Asteroid explodiert hat, und soll sie (die Entität) dieses Ereignis irgendwie abhängig von ihrem Typ behandeln.

3.4 Sequence Diagramme

3.4.1 Move

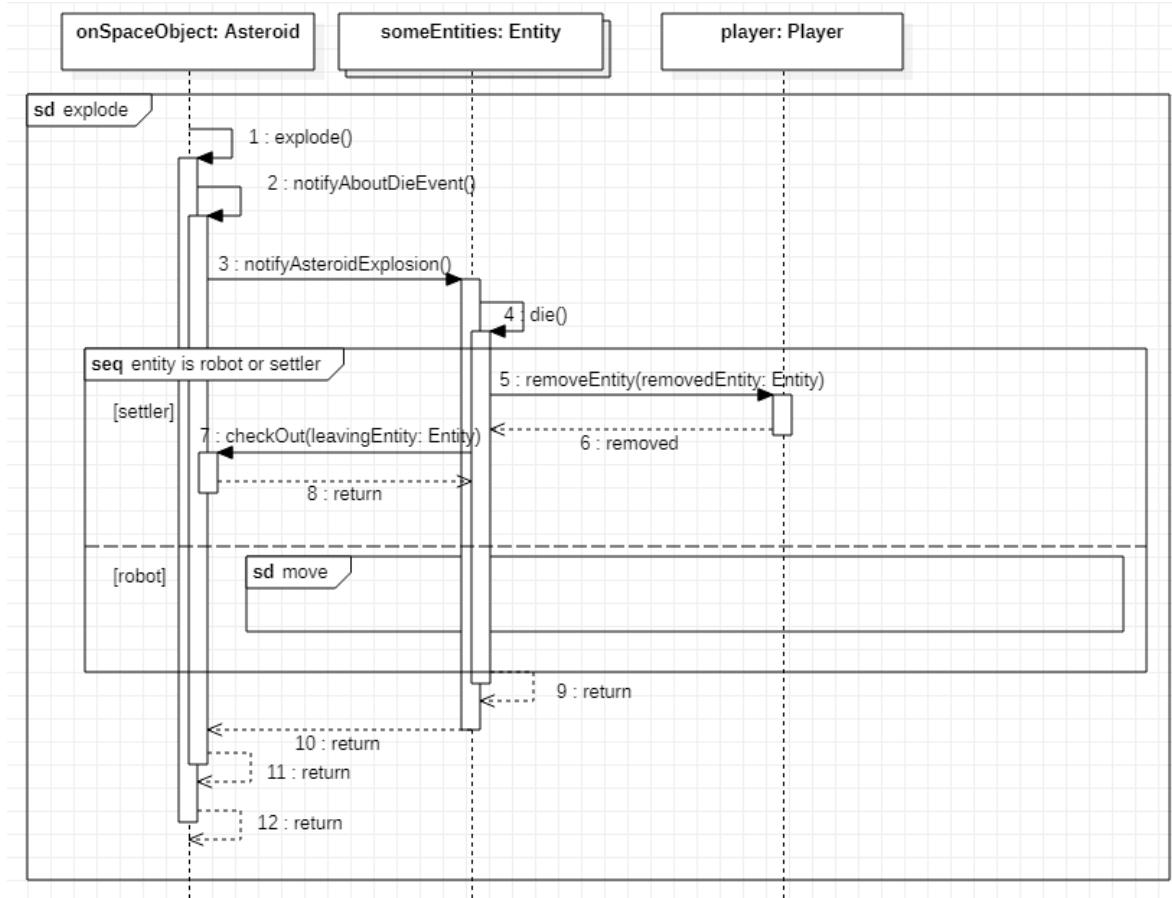


3.4.2 Drill

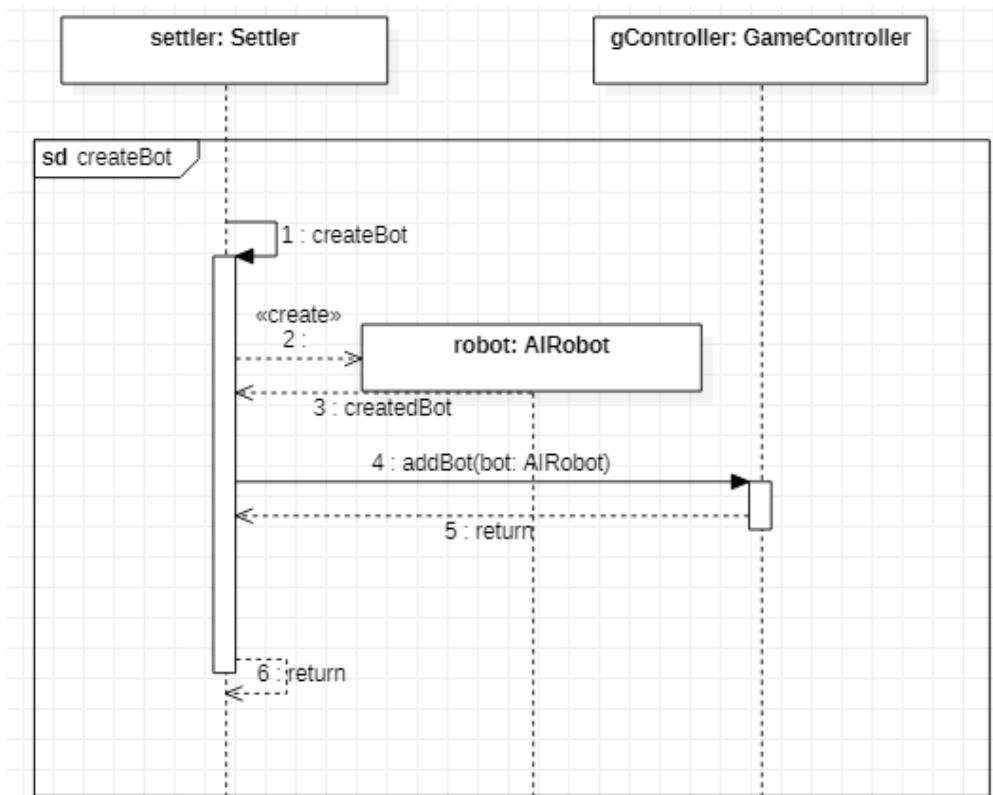


3.4.3

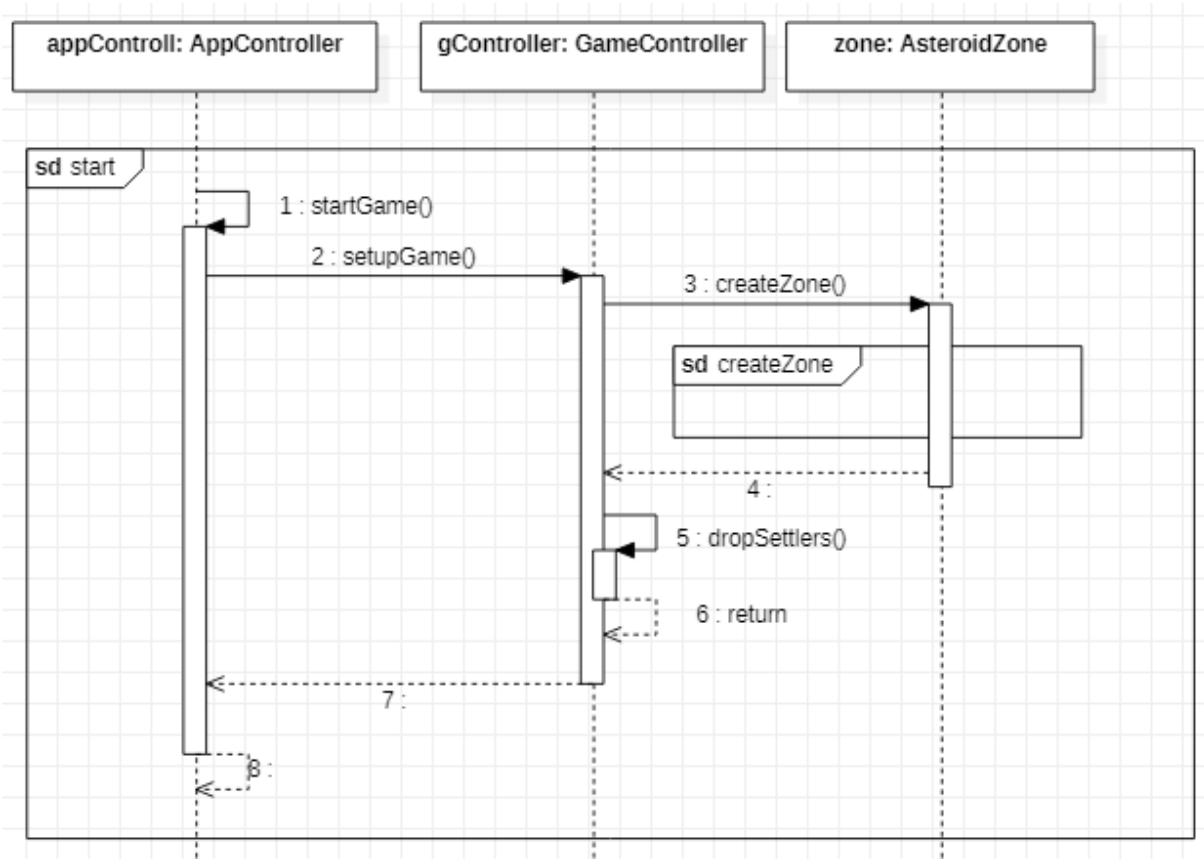
3.4.4 Explode



3.4.5 CreateBot

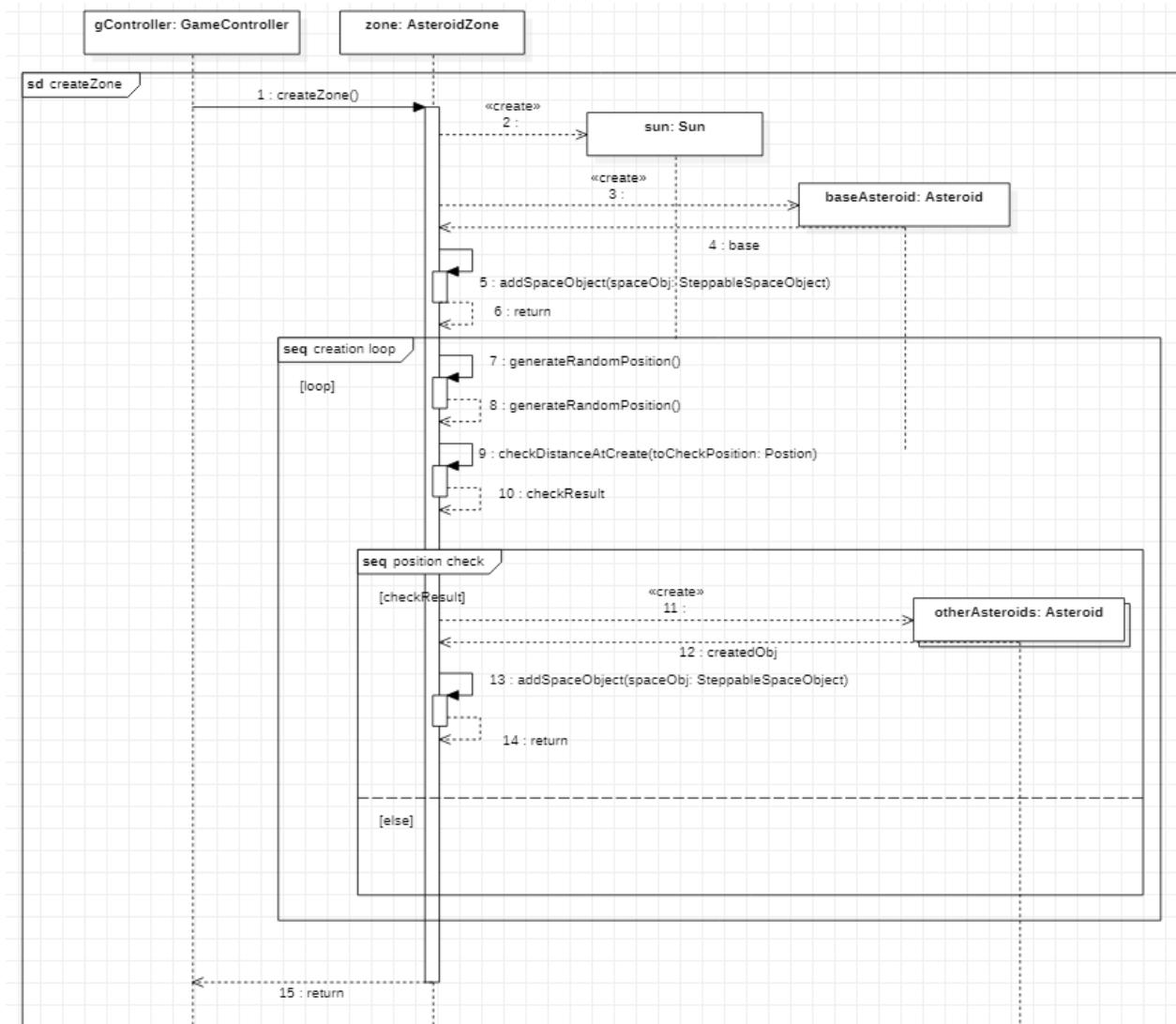


3.4.6 Start

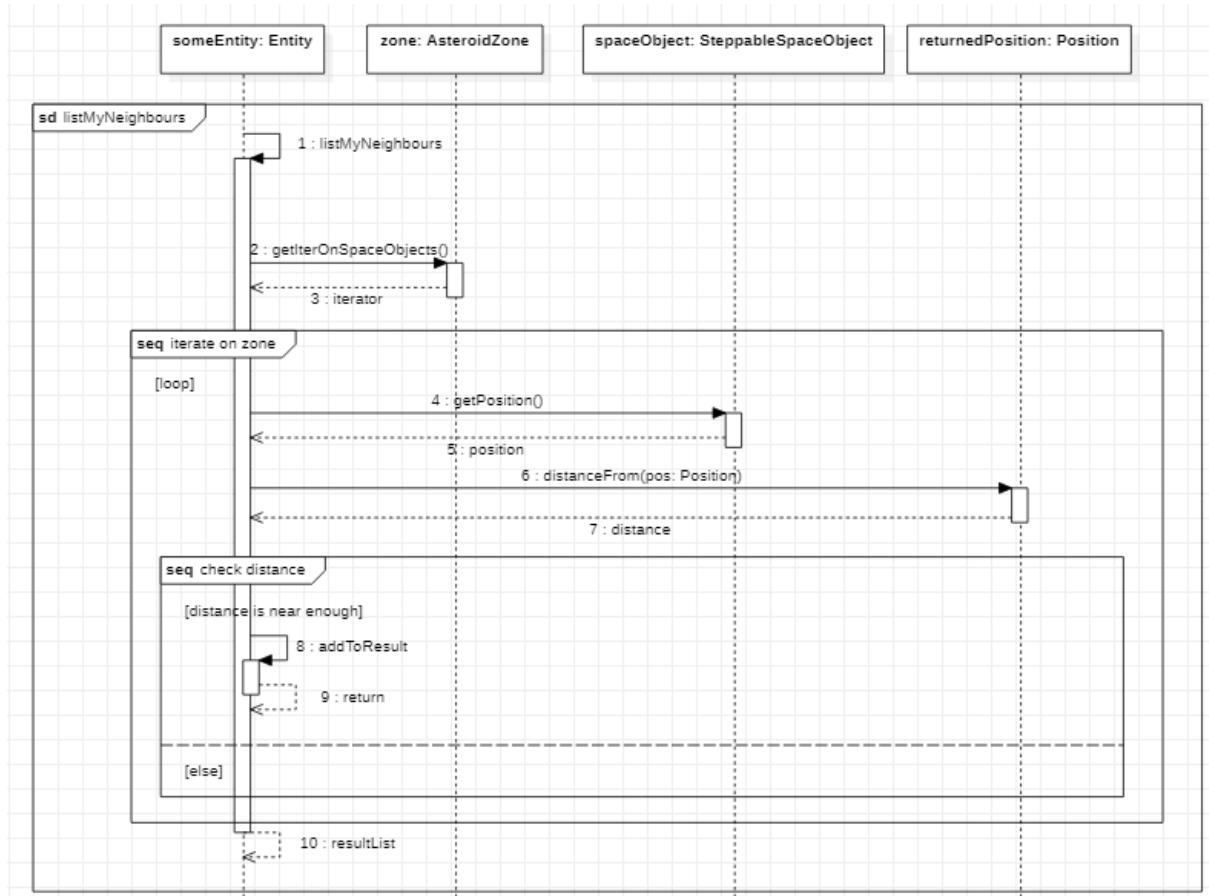


3.4.7

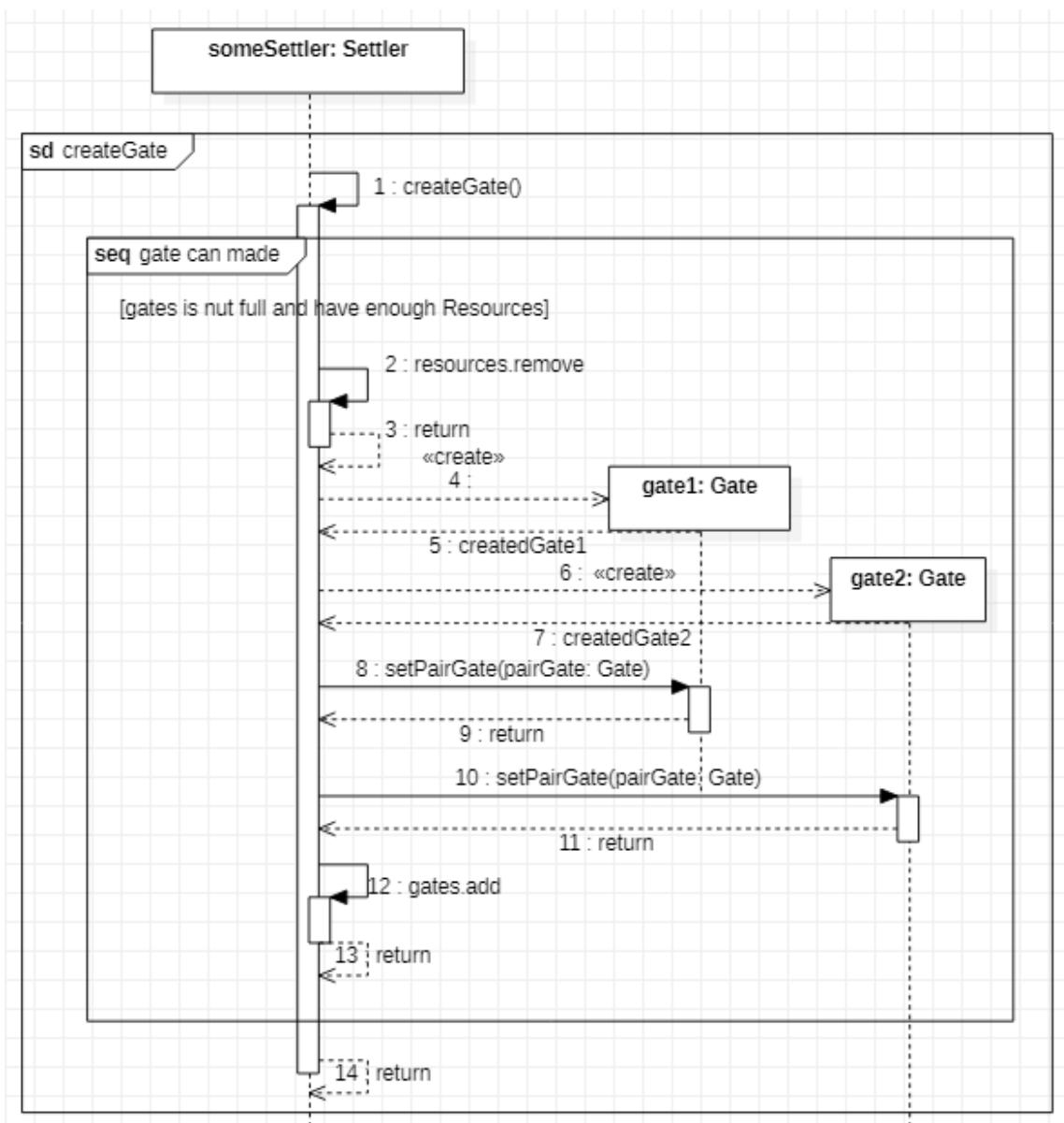
3.4.8 CreateZone



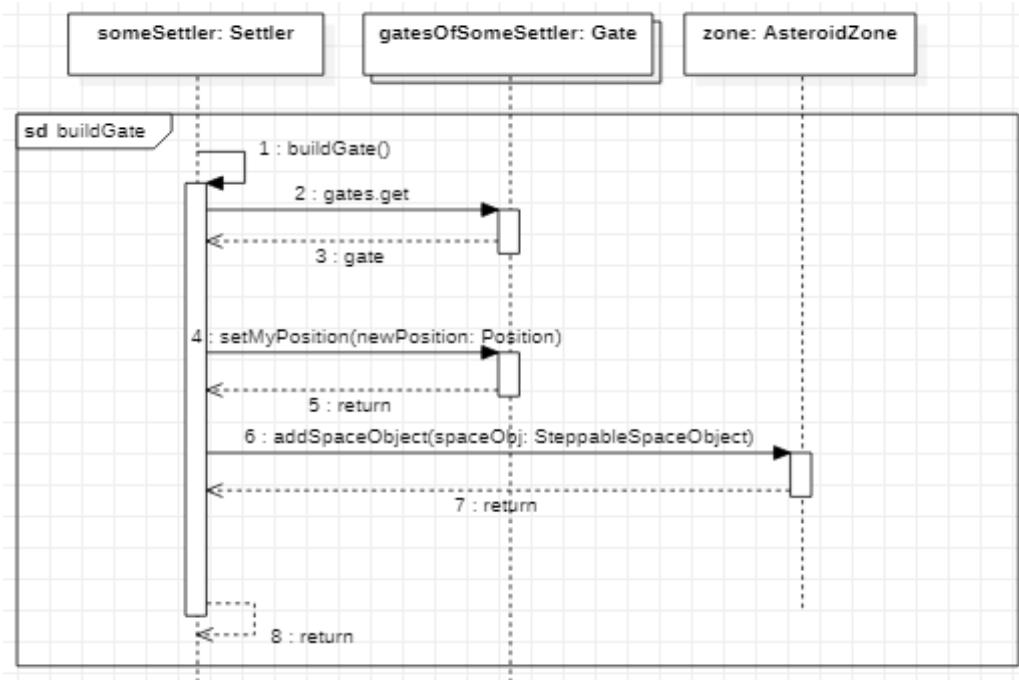
3.4.9 ListMyNeighbours



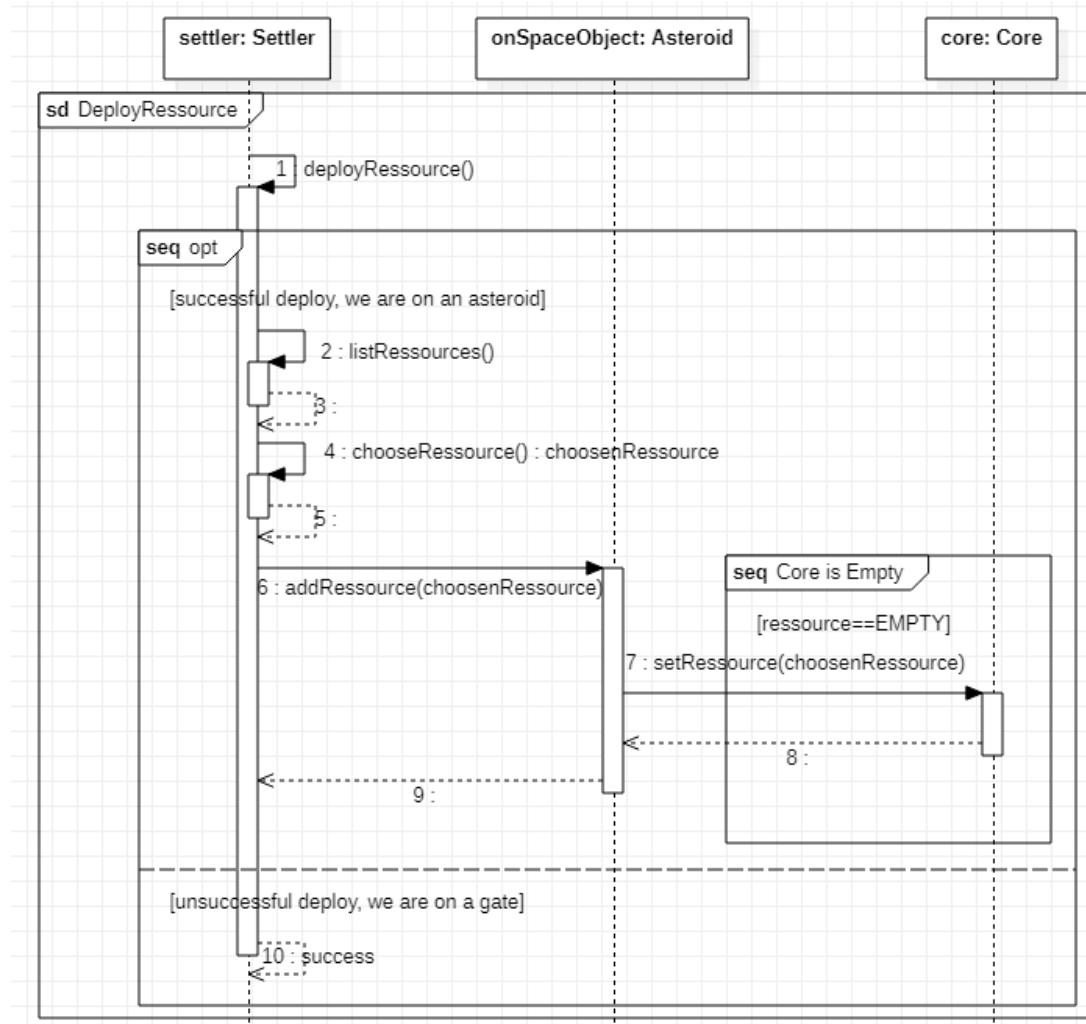
3.4.10 CreateGate



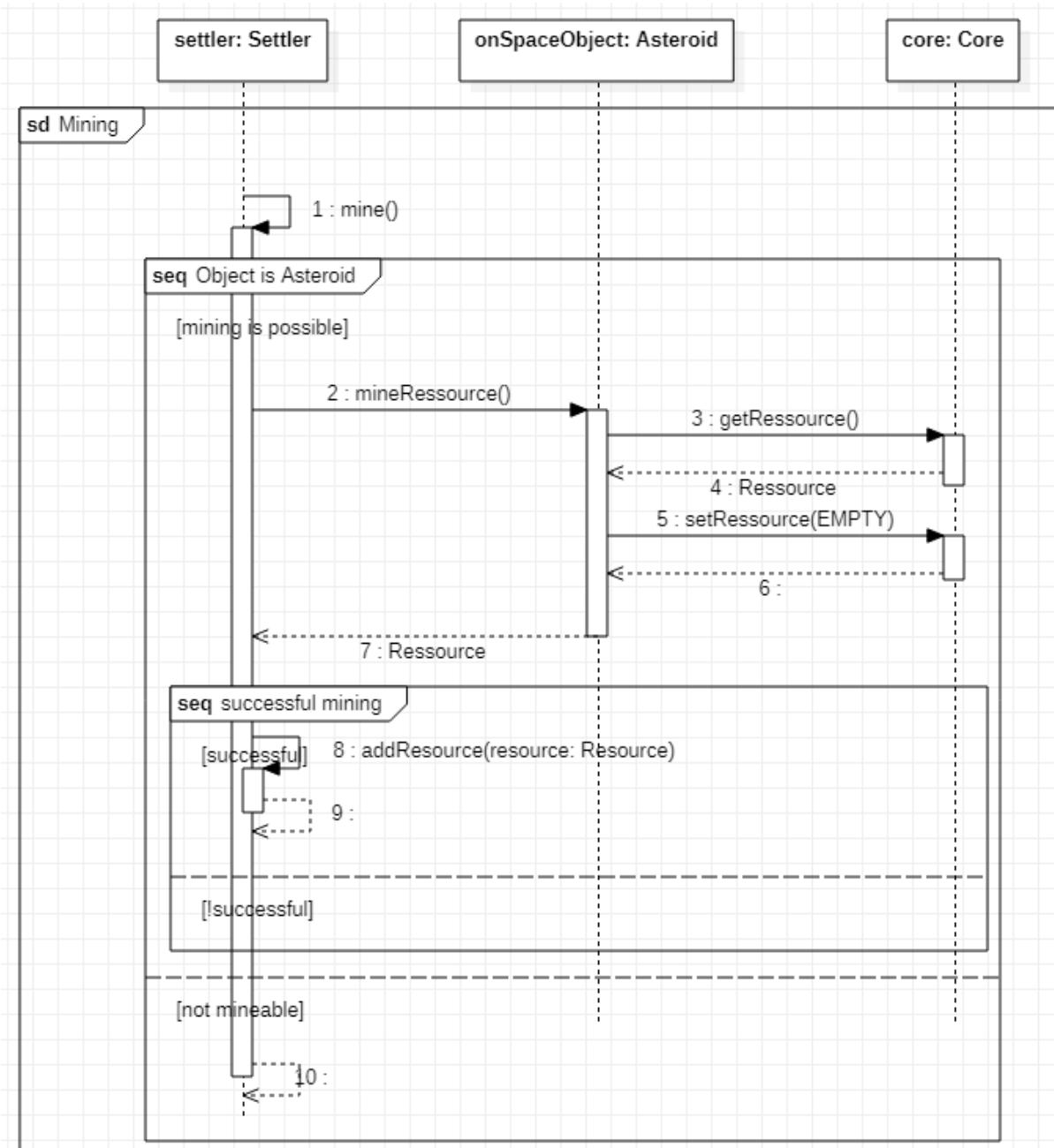
3.4.11 BuildGate



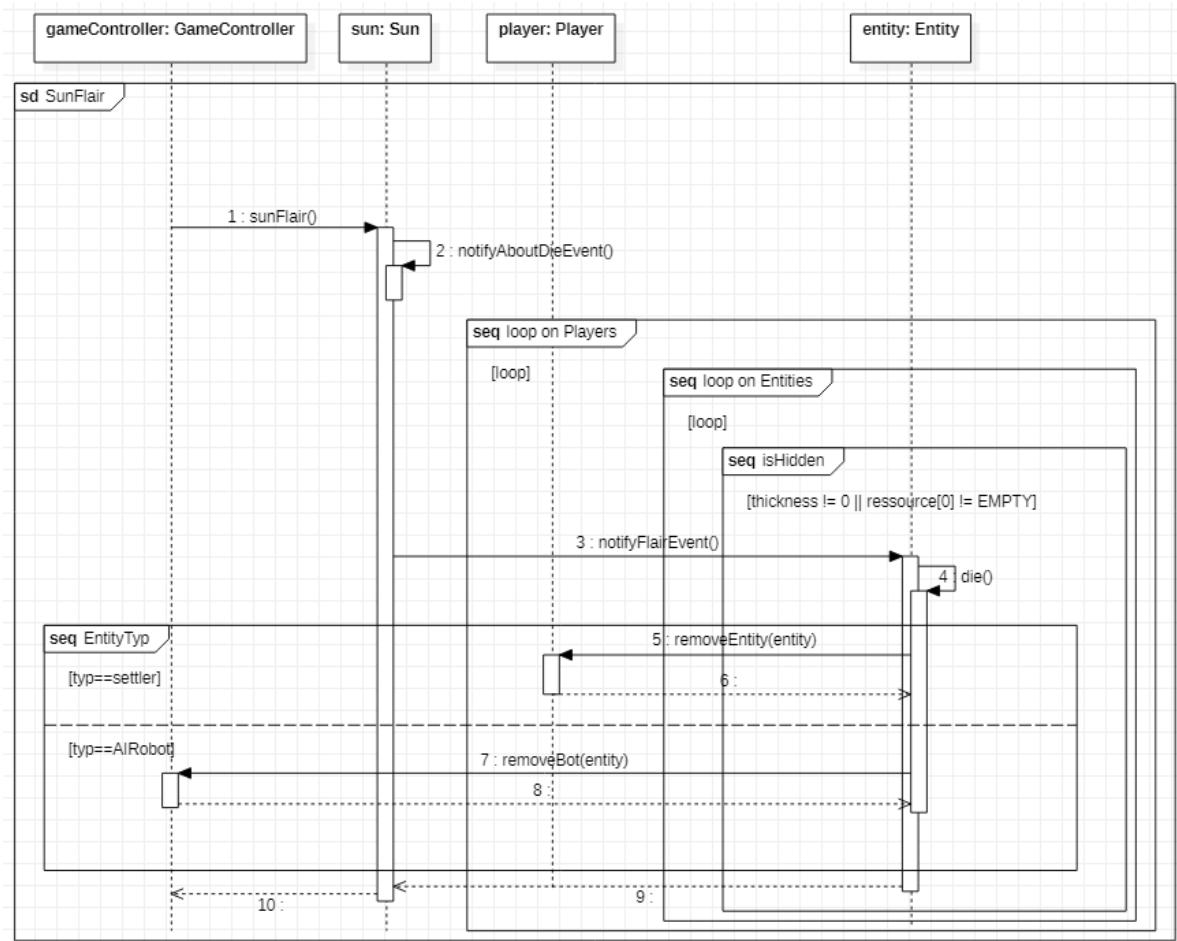
3.4.12 DeployResource



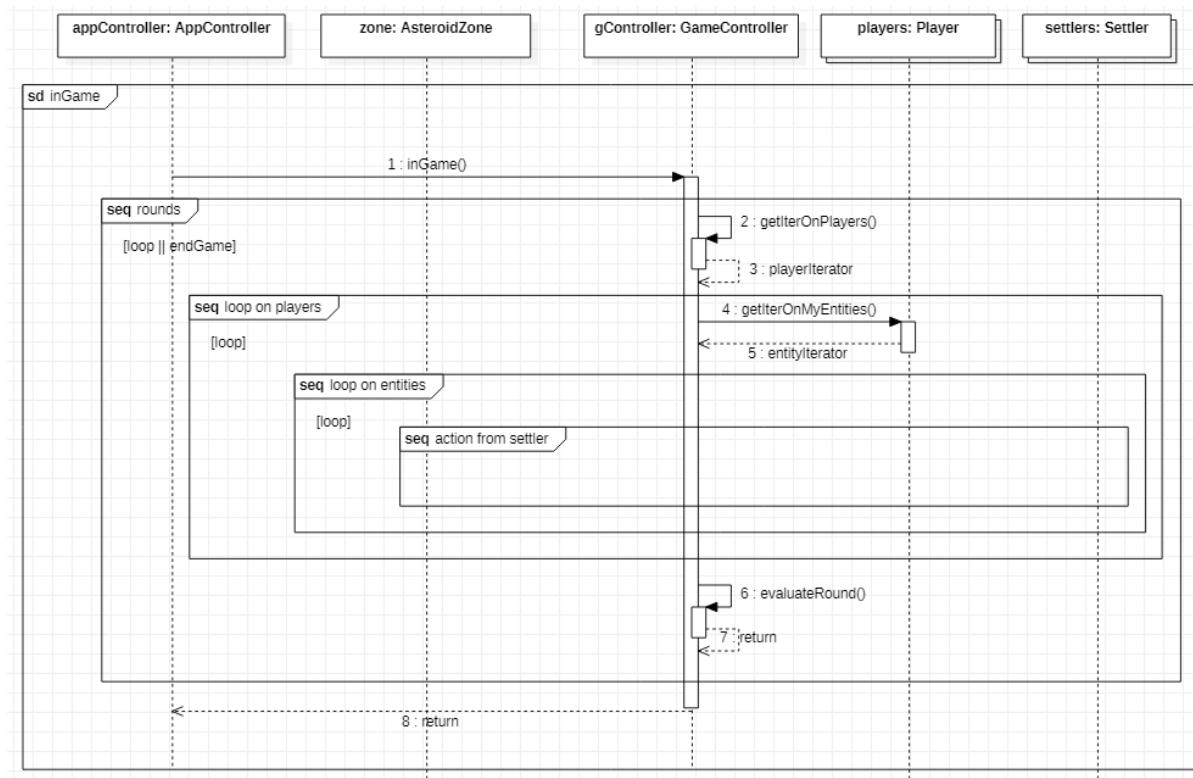
3.4.13 Mining



3.4.14 SunFlair

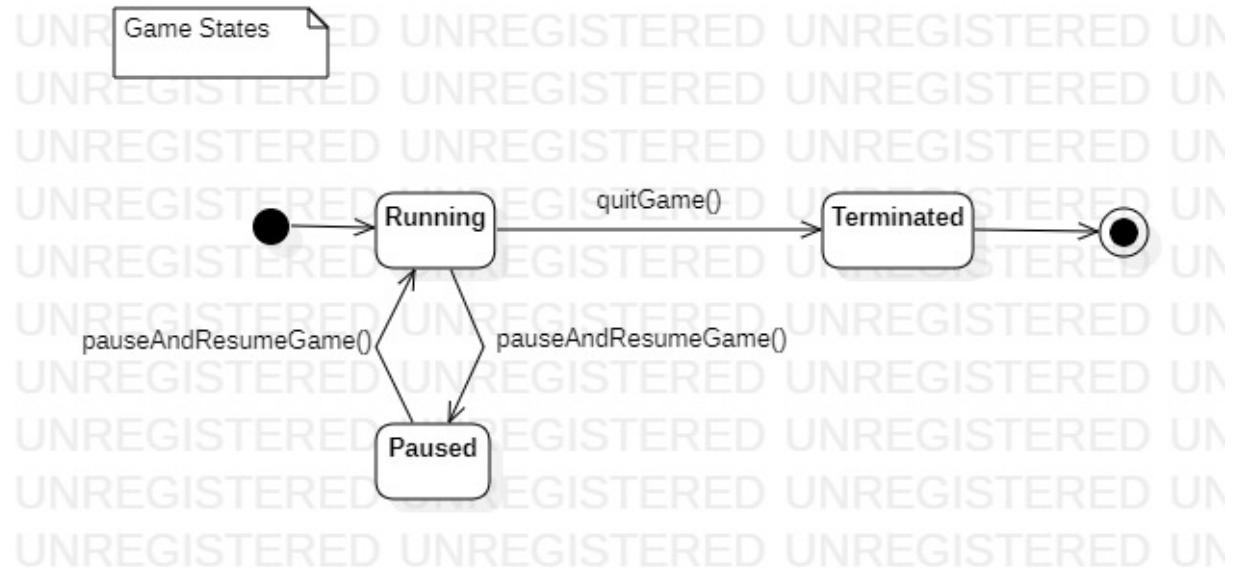


3.4.15 InGame

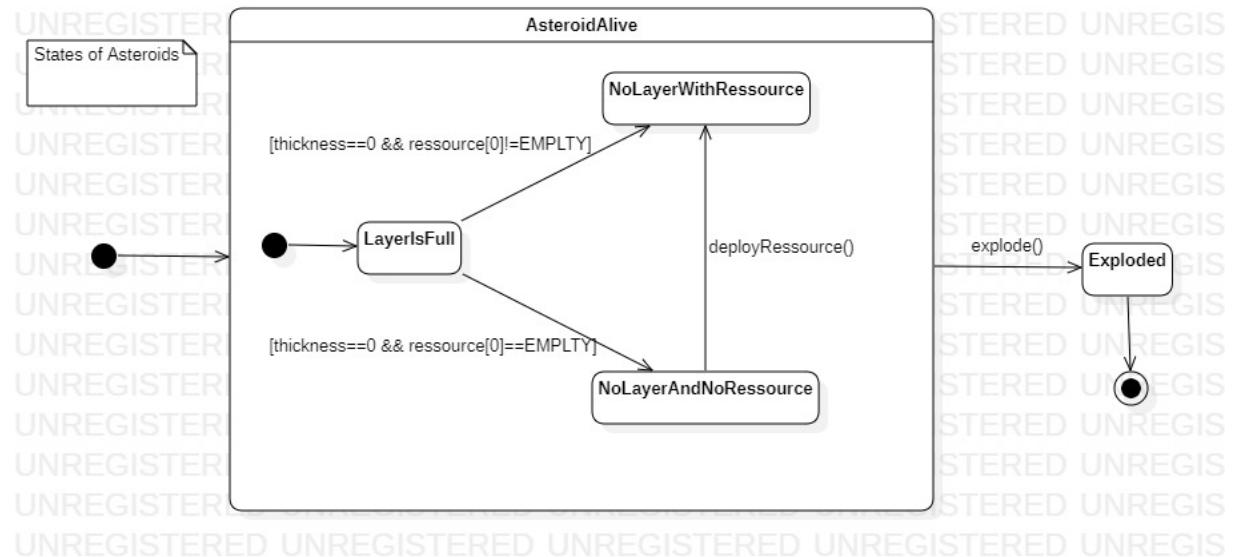


3.5 State-chart Diagramme

3.5.1 Game States



3.5.2 Asteroid States



3.6 Tagebuch

Anfang	Zeitaufwand	Teilnehmer	Beschreibung
2021.02.17. 15:00	2 Stude	Domokos Borbély Hrotkó Pongrácz	Meeting: Aufgabeausteilung auf diesem Etap, Jira eingeführt, über Klassendiagramm sprechen
2021.02.19. 13:00	2 Stunde	Domokos Borbély Hrotkó Pongrácz	Besprechen von Class Diagram
2021.02.20	2 Stunde	Pongrácz	Sequence Diagramme herstellen
2021.02.20	1 Stunde	Hrotkó	Auflistung und allg. Beschr. der Objekten
2021.02.21	2 Stunde	Pongrácz	Sequence Diagramme herstellen, kleine Änderungen in Klassendiagramm
2021.02.21	1,5 Stunde	Hrotkó	Sequenzdiagramme herstellen
2021.02.22.	1 Stunde	Borbély	Beschreibung der Klassen
2021.02.22.	1 Stunde	Domokos	Beschreibung der Klassen
2021.02.23.	1 Stunde	Domokos	Beschreibung der Klassen
2021.02.23.	1 Stunde	Pongrácz	Kleinere Änderungen, Klassenbeschreibung
2021.02.23.	1 Stunde	Borbély	Beschreibung der Klassen
2021.02.23	1 Stunde	Hrotkó	State Chart diagramme hestellen, durchlesen und kontrollieren das ganze Dokument.
2021.02.23	0,5 Stunde	Hrotkó Pongrácz Borbély Domokos	Aufgabenteil abgeschlossen, Diskussion
2021.02.25	1 Stunde	Pongrácz	Kleinere Änderungen

4. Entwicklung des Analysemodells

2 – Halooo Matematikeeer

Konzulent:
Márton Kovács

Mitglieder

Henrietta Domokos	J0DCPT	domoheni@gmail.com
Ábel Borbély	DRP0DT	babel1122@gmail.com
Renátó Hrotkó	OIT6HD	renatohrotko@gmail.com
Vince Pongrácz	MKMDJO	pongvin@gmail.com

17.02.2021

4. Entwicklung des Analysemodells

4.1 Objekten Kataloge

4.1.1.1 Asteroid

Daraus können die Spieler Ressourcen fördern nachdem ihr Mantel durchbohrt wurde. In Sonnennah kann es explodieren falls radioaktive Ressource enthält und der Spieler es versuchen hat durchbohren.

4.1.1.2 HomeAsteroid

Darauf werden die Spieler die Rohstoffe sammeln um am Ende eine endgültige Basis aufzubauen zu können. Das kann nicht explodieren und kann nicht gebohrt werden.

4.1.2 Ressources

Es enthält die Typen der möglichen Ressourcen die während des Spiels gefördert werden können. Es ist auch gewusst ob es radioaktiv ist oder nicht.

4.1.2.1 Uran

Ein Typ von Rohstoff dessen Name Uran und liegt in Asteroiden und außerdem ist es radioaktiv also kann in Sonnennah explodieren.

4.1.2.2 Iron

Ein Typ von Rohstoff dessen Name Eisen und liegt in Asteroiden und außerdem ist es radioaktiv also kann in Sonnennah nicht explodieren.

4.1.2.3 Coal

Ein Typ von Rohstoff dessen Name Kohle und liegt in Asteroiden und außerdem ist es radioaktiv also kann in Sonnennah nicht explodieren.

4.1.2.4 FrozenWater

Ein Typ von Rohstoff dessen Name gefrorenes Wasser und liegt in Asteroiden und außerdem ist es radioaktiv also kann in Sonnennah nicht explodieren.

4.1.2.5 Empty

Ein Typ von Rohstoff dessen Name Leer und liegt in Asteroiden und außerdem ist es radioaktiv also kann in Sonnennah nicht explodieren.

4.1.3 Core

In einem Asteroide ist ein Core/Kern der enthält immer ein Ressource. In dem Kern ist immer nur ein Ressource außer der Basisasteroide der viel mehr speichern kann weil dort wird der Basis aufgebaut.

4.1.4 Layer

Jede Asteroide hat ein Mantel, dessen Dicke veränderlich ist. Beim bohren wird es gebohrt um das Rohstoff in dem Kern erreichen zu können. Beim Bohren wird es immer 1 Einheit kleiner.

4.1.5 Position

Es hat immer eine x und y Koordinate mit denen die Positionen der Entitäten, Asteroiden und der Sonne bestimmt ist. Asteroiden haben auch ein Radius drin mit dem bestimmt ist, wie nah andere Asteroiden sein können.

4.1.6 Entity

Die Spieler haben Entitäten zB: Siedler und AIRoboter. Ein Entität kann sich bewegen zwischen Asteroiden, bohren die Mantel der Asteroiden, sterben bei einer Explosion (außer Roboter) und Sonnenflair, die Nachbarn behandeln.

4.1.7 AIRoboter

Spieler können aus bestimmten Ressourcen Roboter herstellen die beim Bohren helfen können. Sie überleben den Explosion und landen auf einem benachbarten Asteroide. Sie sind von dem GameController kontrolliert also der Spieler kann mit ihnen nichts tun. Gleichzeitig kann ein Spieler mehr besitzen.

4.1.8 Settler

Spieler können sogar mehrere Siedler haben und mit ihnen können sie spielen. Sie können max 10 Ressourcen haben. Sie können noch fördern aus Asteroiden. Portals herstellen und bauen, Ressourcen zurücksetzen in Asteroiden falls sie leer sind.

4.1.9 Player

Die Spieler haben immer ein oder mehr Siedler und können mehrere Roboter besitzen. Sie haben auch ein Name der vor dem Spiel eingestellt wird.

4.1.10 Sun

In einem Asteroidenzone ist immer eine Sonne. Sie kann ein Solarflair machen, das Die Siedler und Roboter nur in einem leeren Asteroide überleben können.

4.1.11 Gate

Spieler können über Portale verfügen um nach einem anderen Ort zu teleportieren. Ein Spieler kann gleichzeitig immer nur ein Portalpaar dabeihaben.

4.1.12 AsteroidZone

Es kann die ganze Asteroidenzone herstellen. Also die Spieler, Sonne, Asteroiden positionieren mit zufälliger Verteilung bei den Asteroiden.

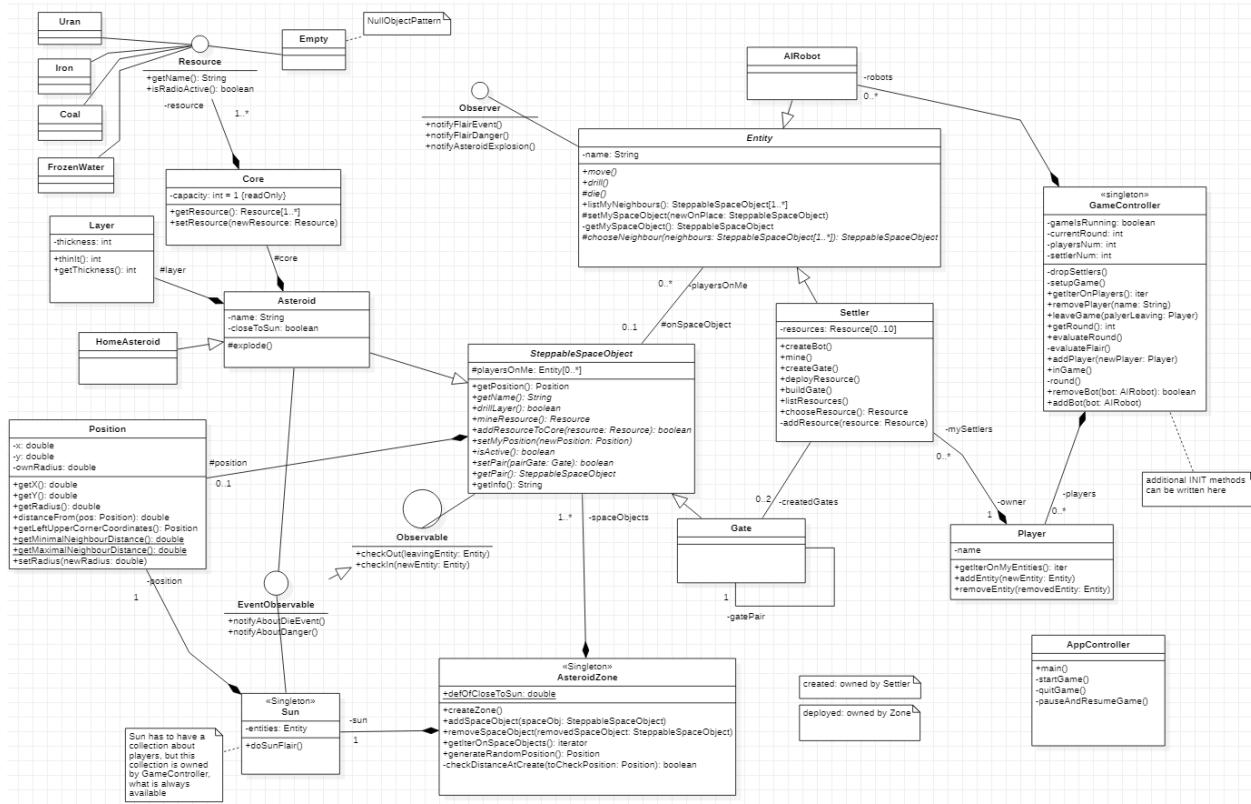
4.1.13 GameController

Es kontrolliert den ganzen Spiel. Die Runde die nacheinander kommen sind hier behandelt und am Ende ausgewertet. Die AIRoboter sind auch von ihm kontrolliert. Es kann den Spiel einstellen.

4.1.14 AppController

Ein AppController kann den ganzen Spiel laufen lassen, während dem Spiel pausieren und dann fortsetzen und falls der Spieler möchte dann den ganzen Spiel sofort beenden.

4.2 Statische Struktur Diagramme



4.3 Beschreibung der Klassen

4.3.1 AIRobot

- **Verantwortung**

Der `AIRobot` bohrt, bewegt sich und hilft den Spielern, ihr Ziel zu erreichen.

- **Basisklasse**

Entity → `AIRobot`

4.3.2 AppController

- **Verantwortung**

Diese Abteilung ist für die Ausführung des Spiels verantwortlich. Startet, stoppt oder pausiert das Spiel.

- **Methoden**

- **main()**: Ein Programm muss eine globale Funktion namens `main` enthalten, die den festgelegten Start des Programms darstellt.

4.3.3 Asteroid

- **Verantwortung**

Der Asteroid speichert Rohmaterial in seinem Kern. Er kann Astronauten schützen, wenn sie sich in seinem Kern verstecken. Das Asteroid kann explodieren. Das Rohmaterial des Kerns kann abgebaut werden..

- **Ősosztályok**

SteppableSpaceObject → Asteroid

- **Schnittstellen / Interface**

- EventObservable
- Observable

- **Attribute**

- **String name:** Der Name des Asteroiden.
- **boolean closeToSun:** Ist der Asteroid in der Nähe des Sonne.

- **Methoden**

- noch überschreibende Methoden von der abstrakten Basisklasse (SteppableSpaceObject)
- **explode():** innere Method um Explosion zu behandeln

4.3.3.2 HomeAsteroid

- **Verantwortung**

Diese Asteroid speichert gleichzeitig mehrere Rohmaterial in seinem Kern. Er kann Astronauten schützen, wenn sie sich in seinem Kern verstecken. Dieser Asteroid kann nicht explodieren, er hat andere Verhalten aus diese Sichtpunkt.

- **Ősosztályok**

SteppableSpaceObject, Asteroid → HomeAsteroid

- **Schnittstellen / Interface**

- EventObservable
- Observable

- **Attribute**

- **String name:** Der Name des Asteroiden. (“Home”)
- **boolean closeToSun:** Ist der Asteroid in der Nähe des Sonne.

- **Methoden**

- noch überschreibende Methoden von der abstrakten Basisklasse (SteppableSpaceObject)
- **explode():** innere Method um Explosion zu behandeln.

4.3.4 AsteroidZone

- **Verantwortung**

Verwaltet das Spielfeld.

- **Attribute**

- **int defOfCloseToSun:** Wenn Sie diese Zahl mit einem SteppableSpaceObject vergleichen, können Sie feststellen, ob es sich in der Nähe der Sonne befindet

- **Methoden**

- **createZone():** Es schafft das Spielfeld vor dem Spielstart.
- **addSpaceObject(spaceObj: SteppableSpaceObject):** Fügt der Asteroidenzone einen SteppableSpaceObject hinzu.
- **removeSpaceObject(onPosition: Position):** Entfernt ein Objekt in einer Asteroidenzone von einer bestimmten Position.
- **iterator getIterOnSpaceObjects():** Wenn ich die Nachbarn auflisten möchte, gibt diese Funktion eine Liste aller SpaceObjects und ich kann basierend darauf weiter filtern.
- **Position generateRandomPosition():** Erstellt Positionen durch Generieren von Zufallszahlen. Erforderlich, um die Strecke zu bauen.

4.3.5 Core

- **Verantwortung**

Es ist verantwortlich für die Speicherung der Rohstoffe innerhalb des Asteroiden

- **Attribute**

- **int capacity:** Gibt an, wie viele Rohstoffe sich im Kern befinden können.

- **Methoden**

- **Resource[1...*] getResource:** Es kehrt mit dem Rohmaterial des Kerns zurück.
- **setResource(newResource: Resource):** Setzt den Kernrohstoff auf eine bestimmte Ressource.

4.3.6 Entity

- **Verantwortung**

Abstrakte Basisklasse für Roboter und Siedler.

- **Schnittstellen / Interface**

- Observer → Entity

- **Attribute**

- **String name:** Name des Entity.

- **Methoden**

- **move():** Durch Aufrufen dieser Funktion werden die Entitäten verschoben.

- **drill()**: Durch Aufrufen dieser Funktion werden die Entitäten bohren.
- **SteppableSpaceObject[] listMyNeighbours()**: Gibt die Nachbarn des Asteroiden zurück, auf dem sich die Entität befindet.

4.3.7 EventObservable

- **Verantwortung**

Wenn ein Asteroid explodiert oder eine Sonneneruption gibt, benachrichtigt SteppableSpaceObject.

- **Methoden**

- **notifyAboutDanger()**: Informiert alle aufgeschriebene Entität über die Gefahr von Solarflair.
- **notifyAboutDieEvent()**: Informiert alle aufgeschriebene Entität über den Tod, und das Grund darauf.

4.3.8 GameController

- **Verantwortung**

Verwaltet den Verlauf des Spiels. Vom Start zum Ende.

- **Attribute**

- **boolean gameIsRunning**: Sagt dir, ob das Spiel läuft.
- **int currentRound**: Die Nummer der aktuellen Runde des Spiels.
- **int playersNum**: Anzahl der Spieler. Etwas, das während des Setups konfiguriert werden kann.
- **int settlerNum**: Anzahl der Siedler. Etwas, das während des Setups konfiguriert werden kann.

- **Methoden**

- **removePlayer(name: String)**: Spieler aus dem Spiel entfernen.
- **leaveGame(playerLeaving: Player)**: Wenn ein Spieler aufhören möchte, gibt er auf, tötet im Wesentlichen alle seine Siedler und löscht sie auch aus den Spielern.
- **int getRound()**: Gibt das Anzahl der aktuelle Runde im Spiel zurück.
- **evaluateRound()**: Wertet den gegebenen Rund aus. Überprüfen Sie, ob alle Siedler des Spielers gestorben sind, und ob die Spieler gewonnen oder verloren haben.
- **addPlayer(newPlayer: Player)**: Fügt dem Spiel einen neuen Spieler hinzu.
- **inGame()**: Eine fast unendliche Schleife, in dieser Funktion läuft das Spiel und das Spiel kreist als Endlosschleife. Wir rufen diese Funktion auf, indem wir das Spiel starten und am Ende des Spiels beenden.
- **boolean removeBot(bot: AIRobot)**: Entfernt den Roboter. Der Boolesche Wert gibt an, ob Sie ihn entfernt haben.
- **addBot(bot: AIRobot)**: Fügt dem Spiel einen neuen AIRoboter hinzu.

4.3.9 Gate

- **Verantwortung**

Kann von Siedlern erstellt und gespeichert werden. Ermöglicht das Reisen zwischen einem Paar von ihnen.

- **Basisklasse**

SteppableSpaceObject → Gate

- **Attribute**

- **Gate gatePair:** Der Paar eines Portals.

- **Methoden**

nur überschreibende Methoden von der abstrakten Basisklasse.

4.3.10 Layer

- **Verantwortung**

Es stellt die Kortikalis/Kruste/Layer des Asteroiden dar; weiß, wie dick er ist und kann ihn reduzieren.

- **Attribute**

- **int thickness:** Die Dicke der Kruste des Asteroiden.

- **Methoden**

- **int thinIt():** Es verdünnt sich. Der DrillLayer ruft Sie an.
- **int getThickness():** Der Asteroid kehrt mit der Dicke seiner Kortikalis zurück.

4.3.11 Player

- **Verantwortung**

Hilft bei der Verwaltung der Entitäten eines Spielers.

- **Attribute**

- **String name:** Der Name des Spielers.
- **Settler[0...*] mySettlers:** Die Kollektion der Siedler, die zu einem Spieler gehören.

- **Methoden**

- **Iterator getIterOnMyEntities():** Erzeugt einen Iterator für die Kollektion der Siedler eines Spielers
- **addEntity(newEntity: Entity):** Weist einem Spieler eine neu Entität zu.
- **removeEntity(removedEntity: Entity):** Hebt die Zuordnung einer Entität zu einem Spieler auf

4.3.12 Position

- **Verantwortung**

Ordnet Objekten im Feld eine bestimmte Position zu.

- **Schnittstellen / Interface**

Comparable

- **Attribute**

- **double x:** Koordinate auf der x-Achse.
- **double y:** Koordinate auf der y-Achse
- **double ownRadius:** Hilft überlappende Objekte zu vermeiden.

- **Methoden**

- **double getX():** Gibt die x-Koordinate zurück.
- **double getY():** Gibt die y-Koordinate zurück.
- **double getRadius():** Gibt den Radius des zugehörigen Objekts zurück.
- **double distanceFrom(pos: Position):** Kalkuliert die Distanz zweier Positionen.
- **double[2] getLeftUpperCornerCoordinate():** Hilft mit dem korrekten Platzieren der Objekte.
- **double getMinimalNeighbourDistance():** Untere Schranke für die Zufallsdistanz, in der die Objekte benachbart sind.
- **double getMaximalNeighbourDistance():** Obere Schranke...

4.3.13 Settler

- **Verantwortung**

Siedler sind die Entitäten, die die Spieler bewegen und über sie mit dem Spielfeld interagieren können.

- **Basisklasse**

Entity → Settler

- **Attribute**

- **Resources[0...10] resources:** Speichert die abgebauten Ressourcen eines Siedlers.
- **Player owner:** Der Spieler, dem die Siedler gehören.
- **Gate[0...2] createdGates:** Die Kollektion der erstellten Portale.

- **Methoden**

- **createBot():** Erschafft einen AI Roboter.
- **mine():** Entfernt die Ressource aus dem Asteroidenkern, und fügt es seiner eigenen Kollektion von Ressourcen zu.
- **createGate():** Erstellt ein Paar Portale, und fügt sie seiner eigenen Kollektion von Portalen zu.
- **deployResource():** Setzt eine Ressource wieder in einen Asteroiden ein.
- **buildGate():** Platziert eines der getragenen Portale.
- **listResources():** Listet die Ressourcen auf, die der Siedler besitzt.
- **Resource chooseResource():** Der Siedler wählt eine Ressource aus seinem Inventar aus.

- **addResource(resource: Resource)**: Fügt Ressource zu der Kollektion .

4.3.14 SteppableSpaceObject

- **Verantwortung**

Diese sind Objekte, auf die die Spieler mit ihren Siedlern treten können.

- **Schnittstellen / Interface**

Observable

- **Attribute**

- **Entity[0...*] playersOnMe**: Speichert die Entitäten, die auf einem solchen Objekt stehen.

- **Methoden**

- **getPosition()**: Gibt die Position des Objektes zurück.
- **String getName()**: Gibt den Namen des SteppableSpaceObjectes zurück, falls es existiert.
- **boolean drillLayer()**: Falls es eine Asteroid ist, reduziert die Größe der Kortikalins des Asteroiden um eins. Übergibt die Bohraufgabe an das Layer.
- **Ressource mineResource()**: Falls es eine Asteroid ist, baut es das Innere des Asteroiden ab und setzt die Art des Rohmaterials auf leer. Leitet die Mining-Aufgabe an das Core weiter.
- **addResource(ressource: Resource)**: Übergibt die Aufgabe der Rohstoffzugabe an das Core, falls es eine solche Objekt ist, welches ein Core (Kern) hat.
- **setMyPosition(newPosition: Position)**: Setzt die Position eines Portals, falls solche Operation interpretierbar ist.
- **boolean isActive()**: Gibt an, ob ein Portal, oder Objekt aktiv ist, also ob es wirklich funktioniert.
- **boolean setPair(pairGate: Gate)**: Bindet das aktuelle Objekt mit einem Anderen zu.
- **SteppableSpaceObject getPair()**: Gibt das Paar eines Portals/SteppableSpaceObject zurück.
- **String getInfo()**: manche Information über das Objekt. Später wird es nützlich, beim GUI.

4.3.15 Sun

- **Verantwortung**

Startet Sunflairs, was gefährlich für Siedler und Roboter sind.

- **Schnittstellen / Interface**

EventObservable

- **Attribute**

- **Position position**: Position der Sonne auf dem Spielfeld.

- **Methoden**

- **doSunFlair()**: Jede Entität auf dem Feld wird überprüft, ob sie in Gefahr ist, und wenn ja, stirbt sie.

4.3.16 Observable

- **Verantwortung**

Diese Schnittstelle hat Operationen, dadurch Entitäten (Siedler und Roboter) zur Kollektion von SteppableSpaceObjects hinzugefügt werden können.

- **Methoden**

- **checkOut(leavingEntity: Entity)**: Das SteppableSpaceObject entfernt die Entität von sich.
- **checkIn(newEntity: Entity)**: Das SteppableSpaceObject registriert eine ankommende Entität an sich.

4.3.17 Observer

- **Verantwortung**

Diese Schnittstelle hat das Aufgabe, dass die Entitäten auf SolarFlair, oder Asteroidexplosion reagieren. Diese Methoden der Interface hat die Kenntnisse, wie das aktuelle Entität das Ereignisse behandelt.

- **Methoden**

- **notifyFlairEvent()**: Benachrichtigt die Entität (und Spieler auch) über einem FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt.
- **notifyFlairDanger()**: Benachrichtigt die Entität (und Spieler auch) über einem bevorstehende FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt. Oft nur ein Message auf dem Bildschirm, über die zukünftige Solarflair.
- **notifyAsteroidExplosion()**: Benachrichtigt die Entität (und Spieler auch), dass ihre Asteroid explodiert hat, und soll sie (die Entität) dieses Ereignis irgendwie abhängig von ihrem Typ behandeln.

4.3.18 Resource

- **Verantwortung**

Diese Schnittstelle wird von den Ressourcen implementiert, die sich im Kern der Asteroiden befinden. Sie hat Operationen, dadurch die Typ der Ressourcen abgefragt werden kann. Es sagt auch, ob es radioaktiv ist.

- **Methoden**

- **String getName()**: Gibt den Namen (Typ) der Ressource zurück.
- **boolean isRadioactive()**: Sagt, ob die Ressource radioaktiv ist.

4.3.19 Coal

- **Verantwortung**

Ressource vom Typ Coal.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Coal" zurück.
- **boolean isRadioactive():** Gibt false zurück.

4.3.20 Empty

- **Verantwortung**

Ressource vom Typ Empty.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Empty" zurück
- **boolean isRadioactive():** Gibt false zurück.

4.3.21 FrozenWater

- **Verantwortung**

Ressource vom Typ FrozenWater.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "FrozenWater" zurück
- **boolean isRadioactive():** Gibt false zurück.

4.3.22 Iron

- **Verantwortung**

Ressource vom Typ Iron

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Iron" zurück
- **boolean isRadioactive():** Gibt false zurück.

4.3.23 Uran

- **Verantwortung**

Ressource vom Typ Uran

- **Schnittstellen / Interface**

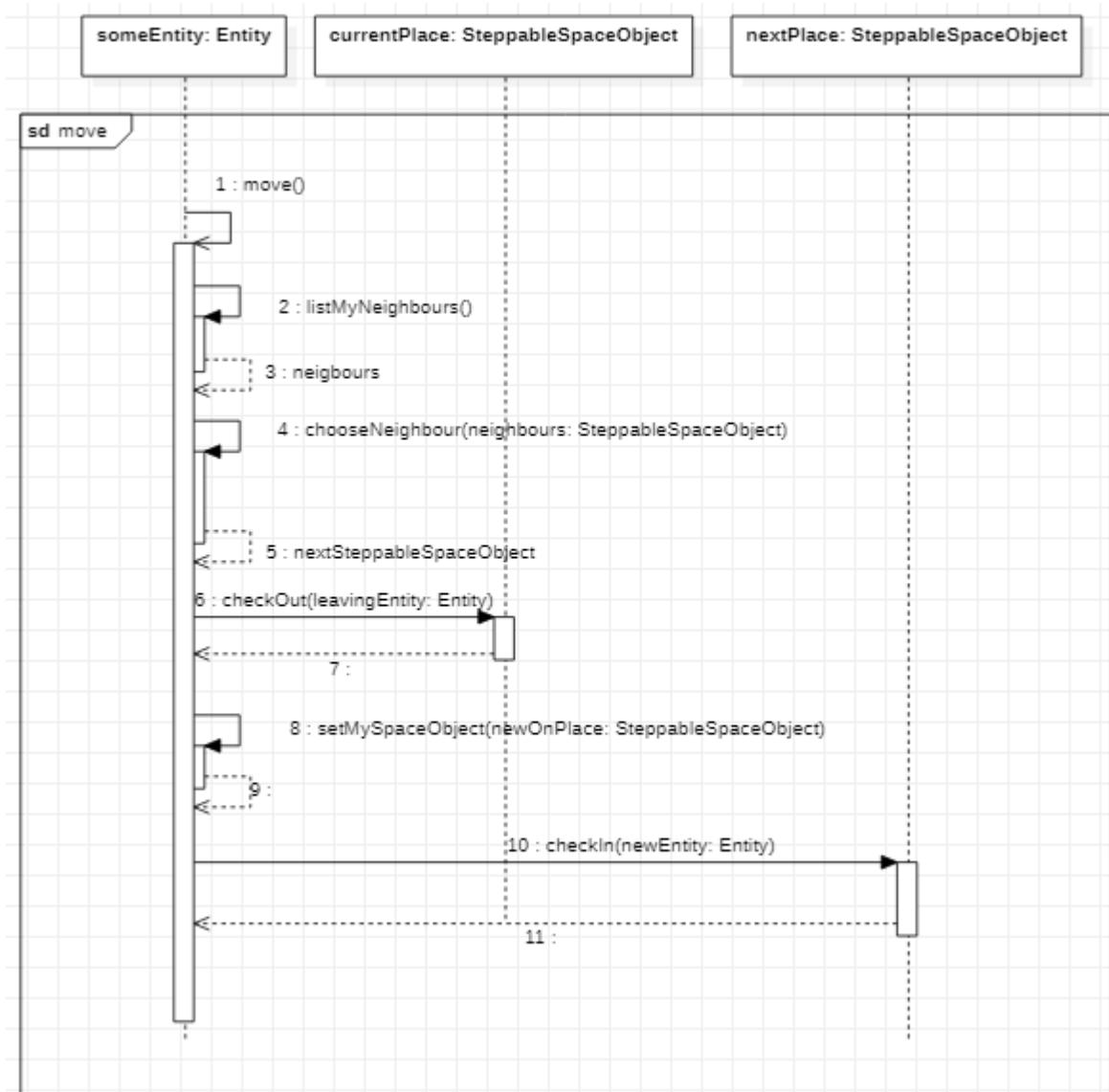
Resource

- **Methoden**

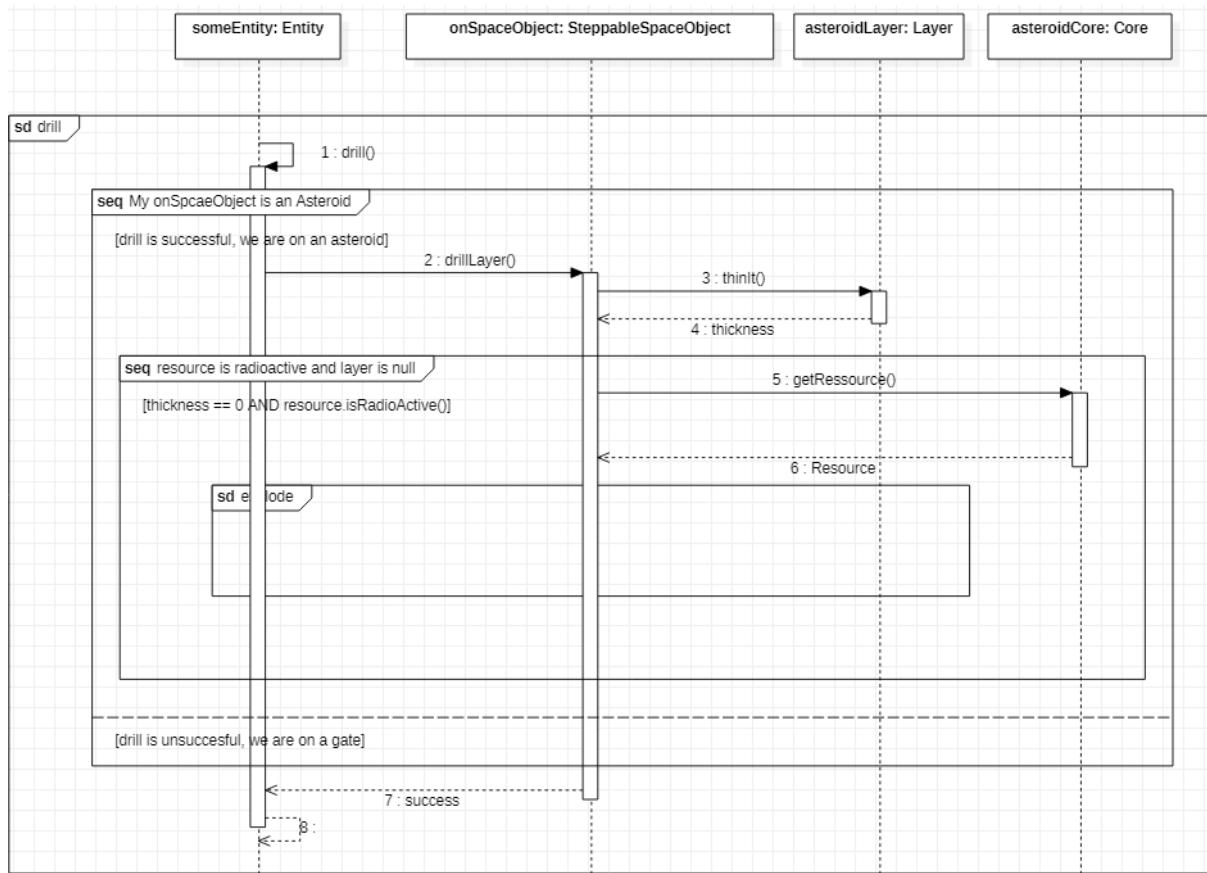
- **String getName():** Gibt "Uran" zurück
- **boolean isRadioactive():** Gibt true zurück.

4.4 Sequence Diagramme

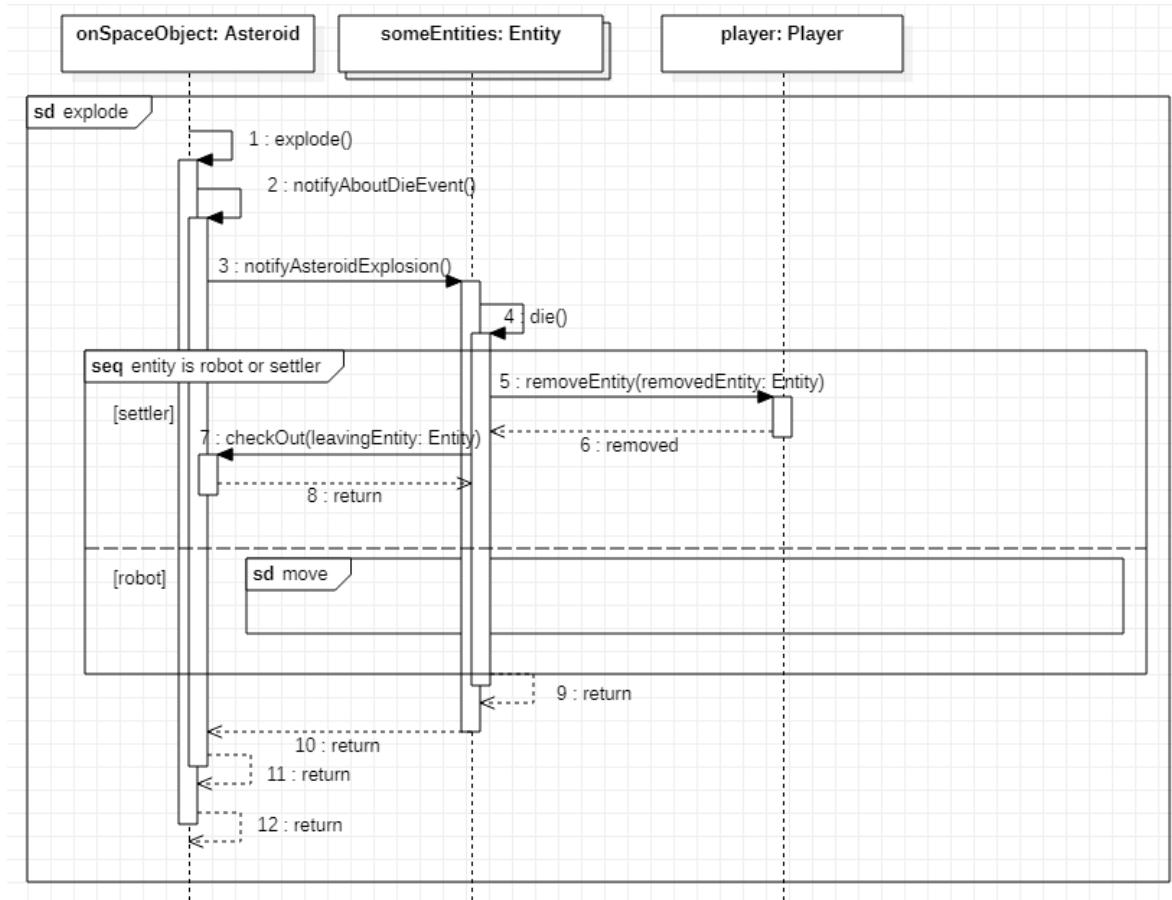
4.4.1 Move



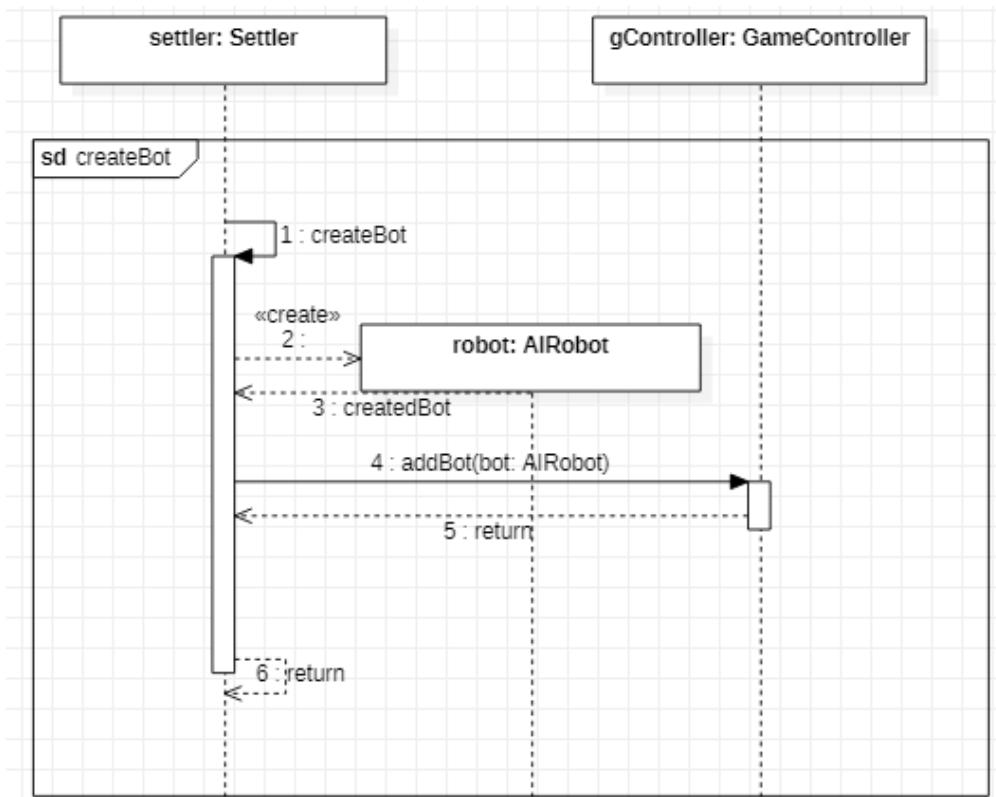
4.4.2 Drill



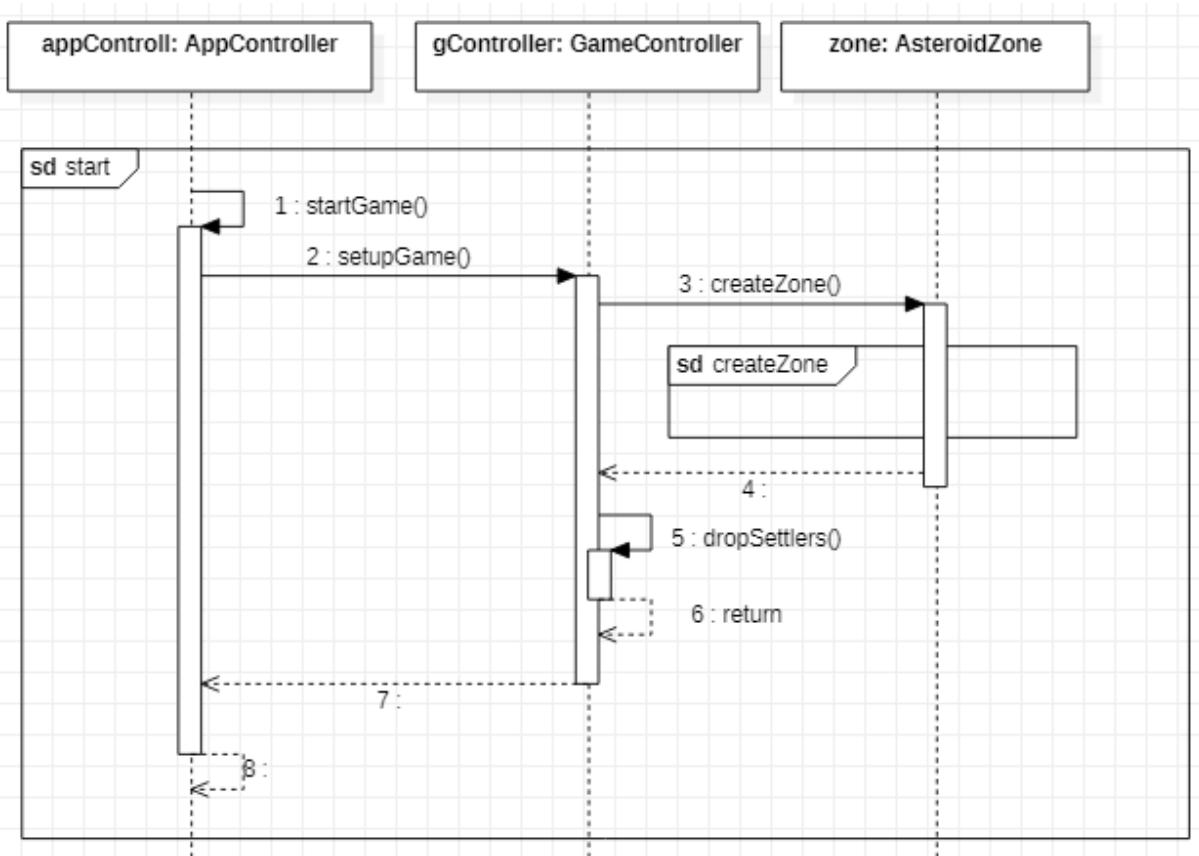
4.4.3 Explode



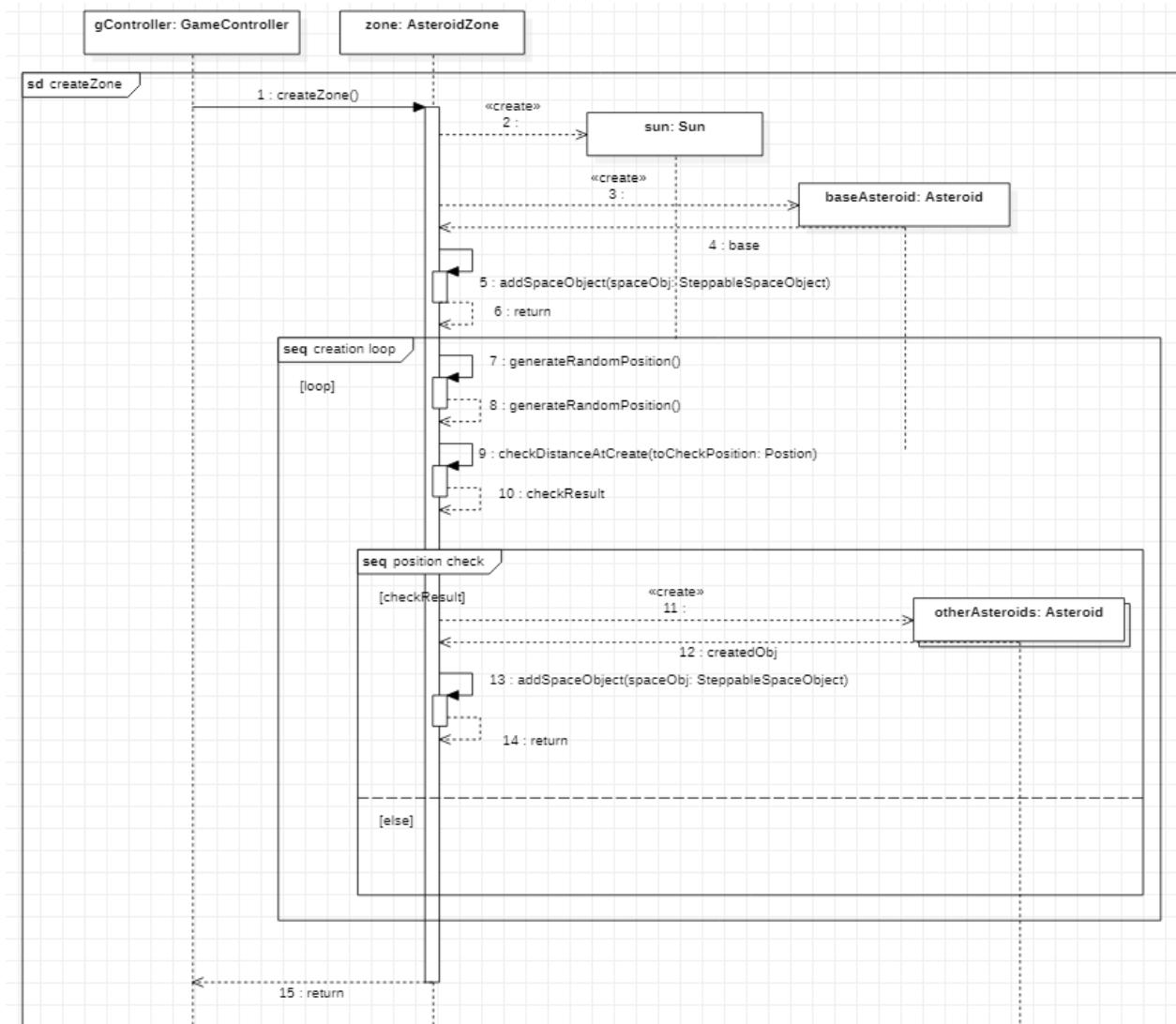
4.4.4 CreateBot



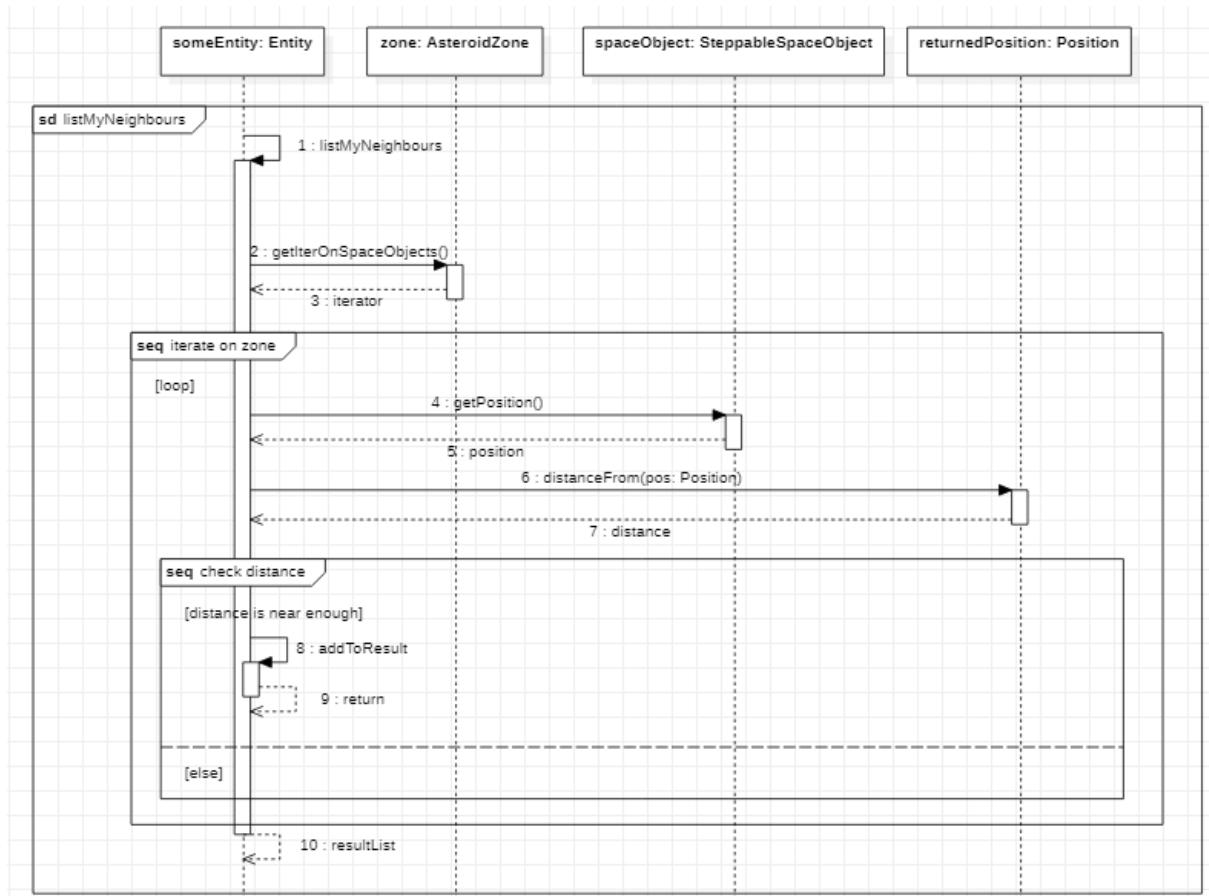
4.4.5 Start



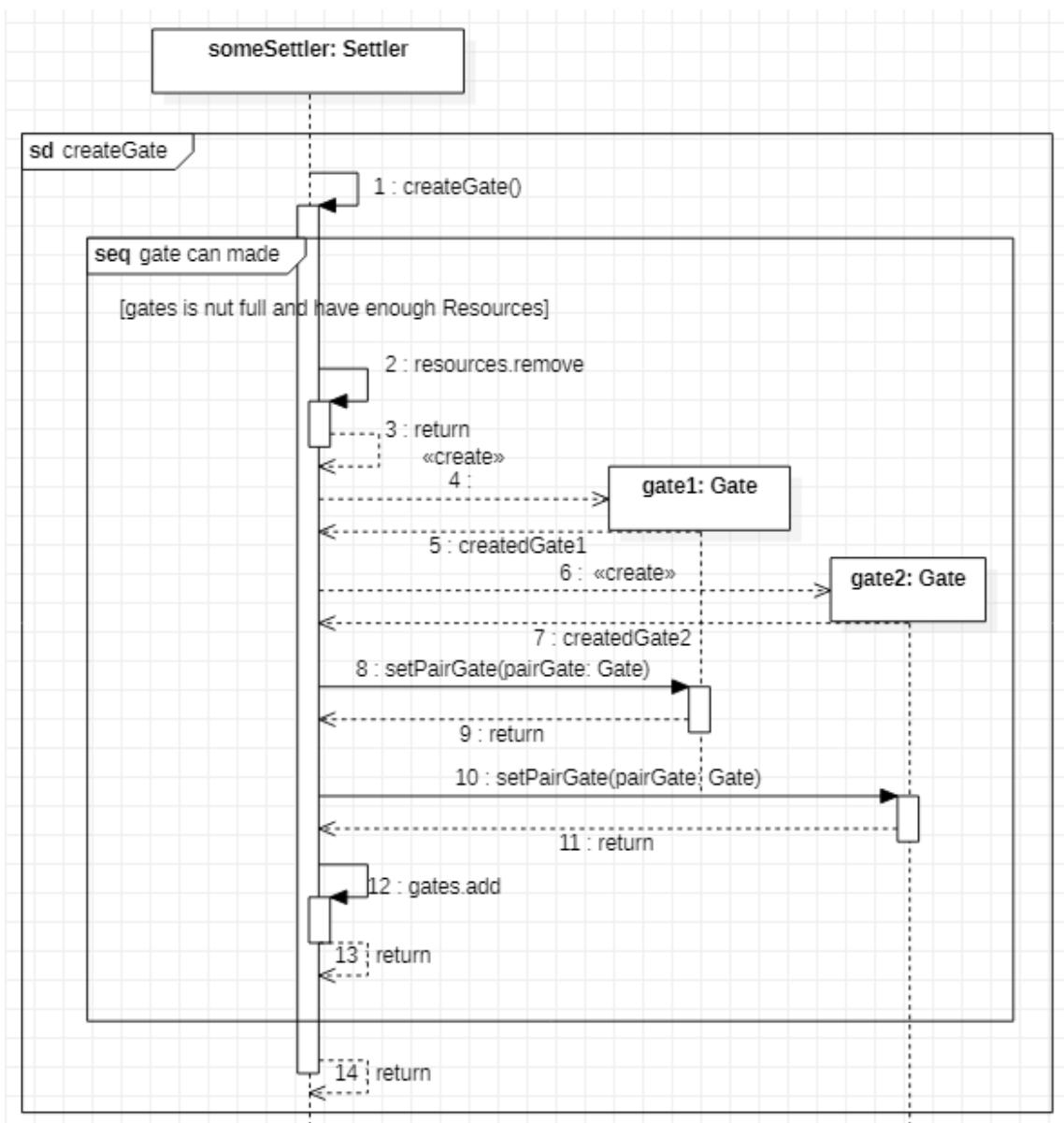
4.4.6 CreateZone



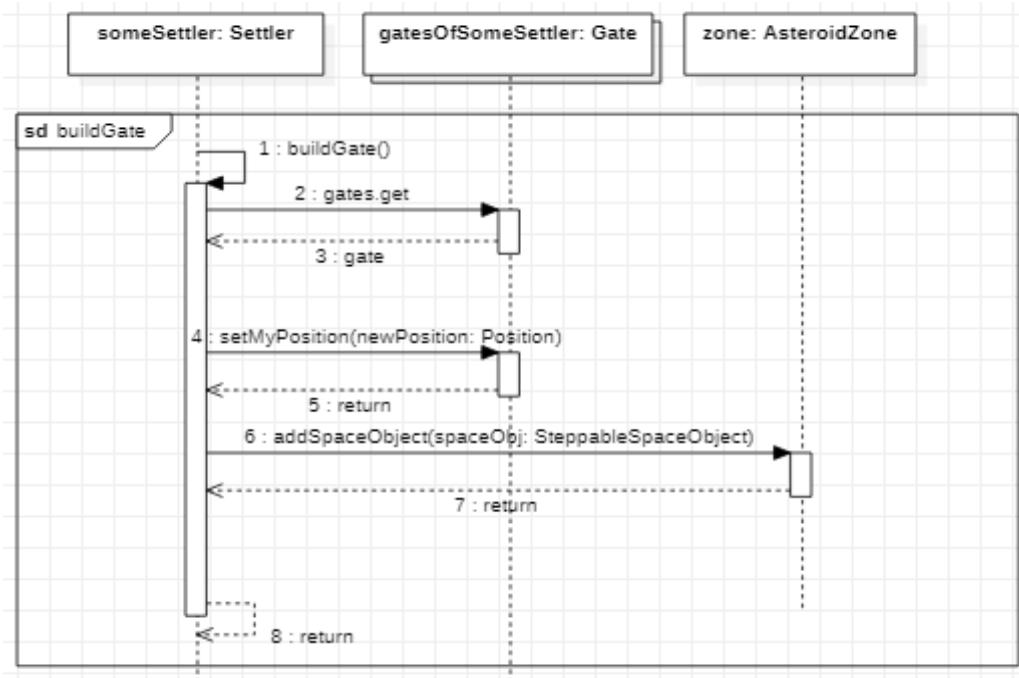
4.4.7 ListMyNeighbours



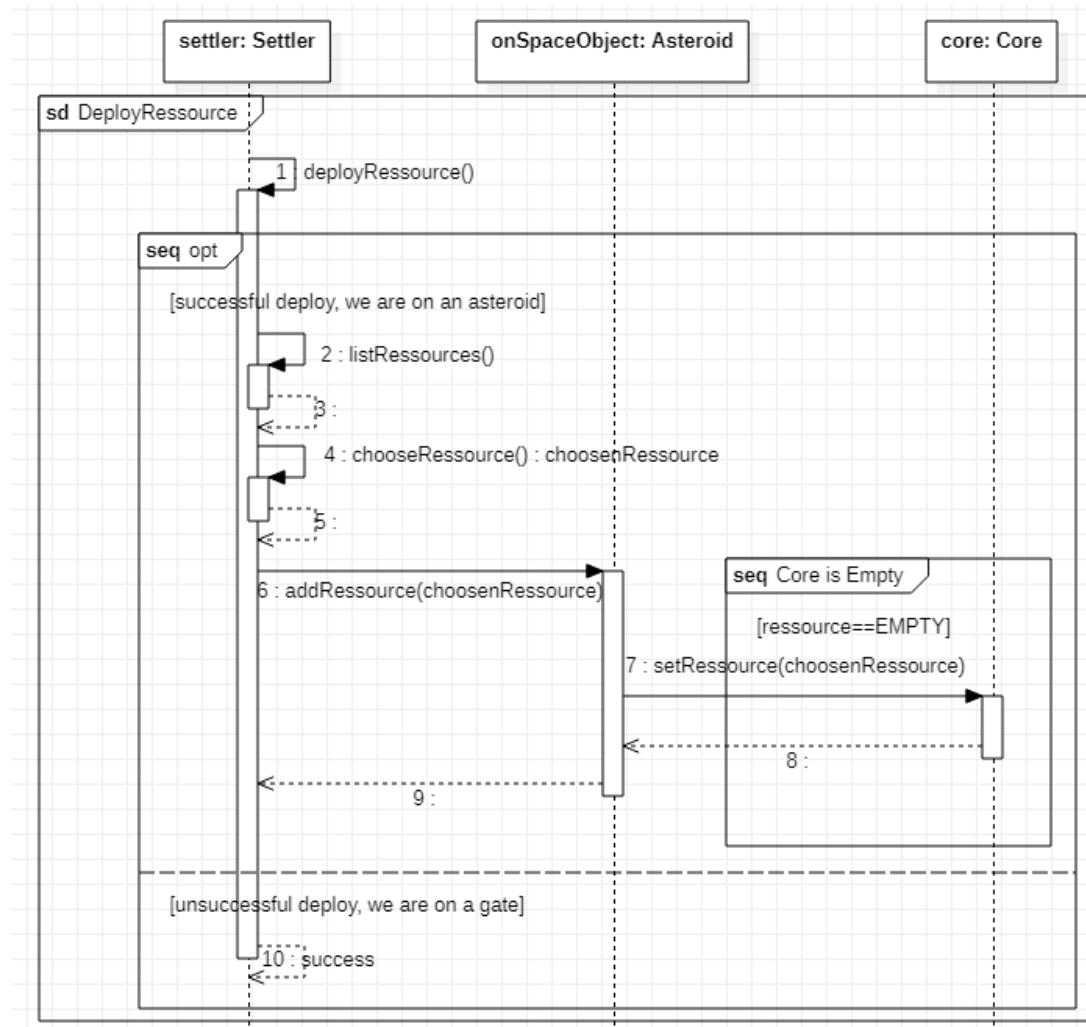
4.4.8 CreateGate



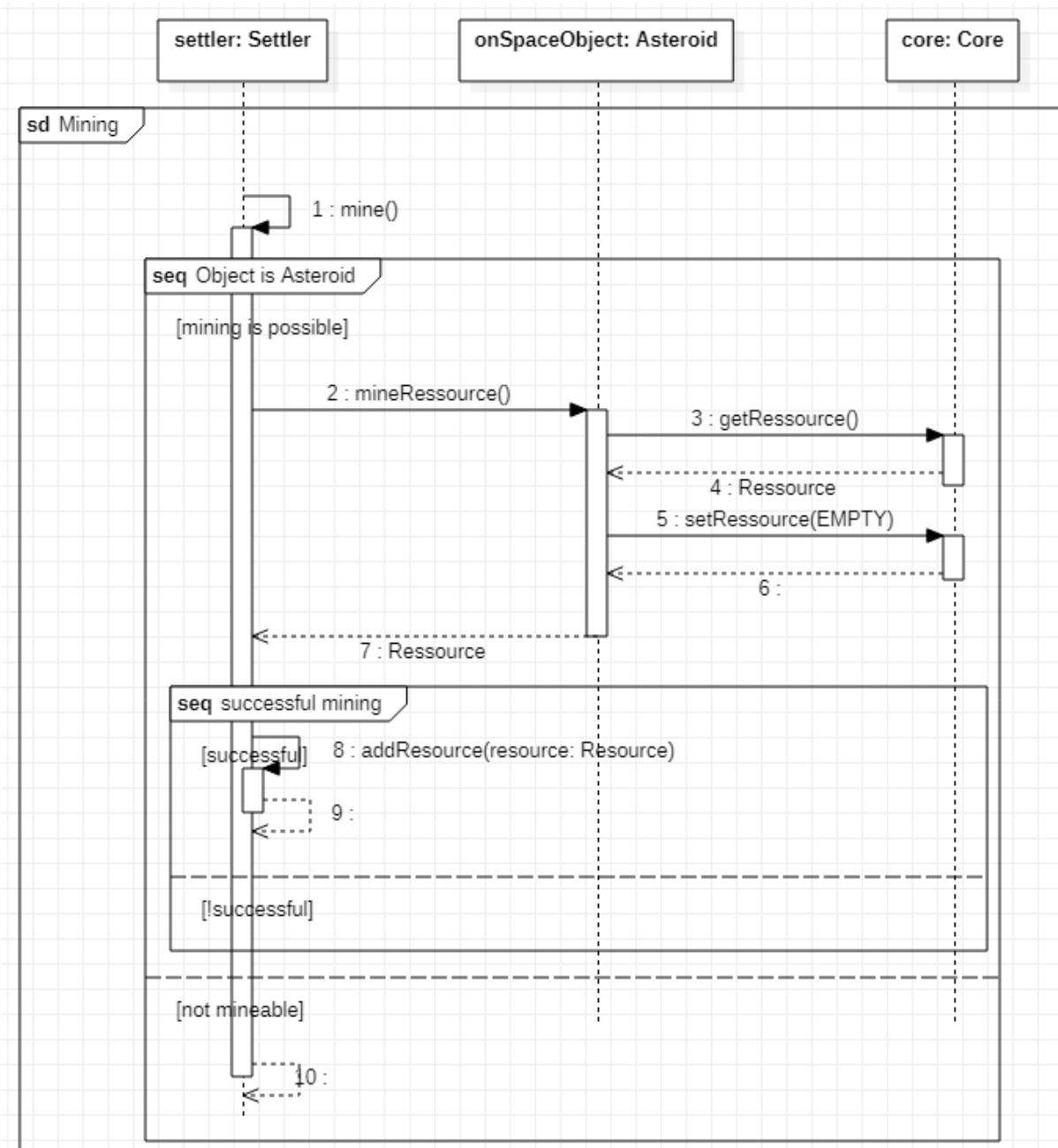
4.4.9 BuildGate



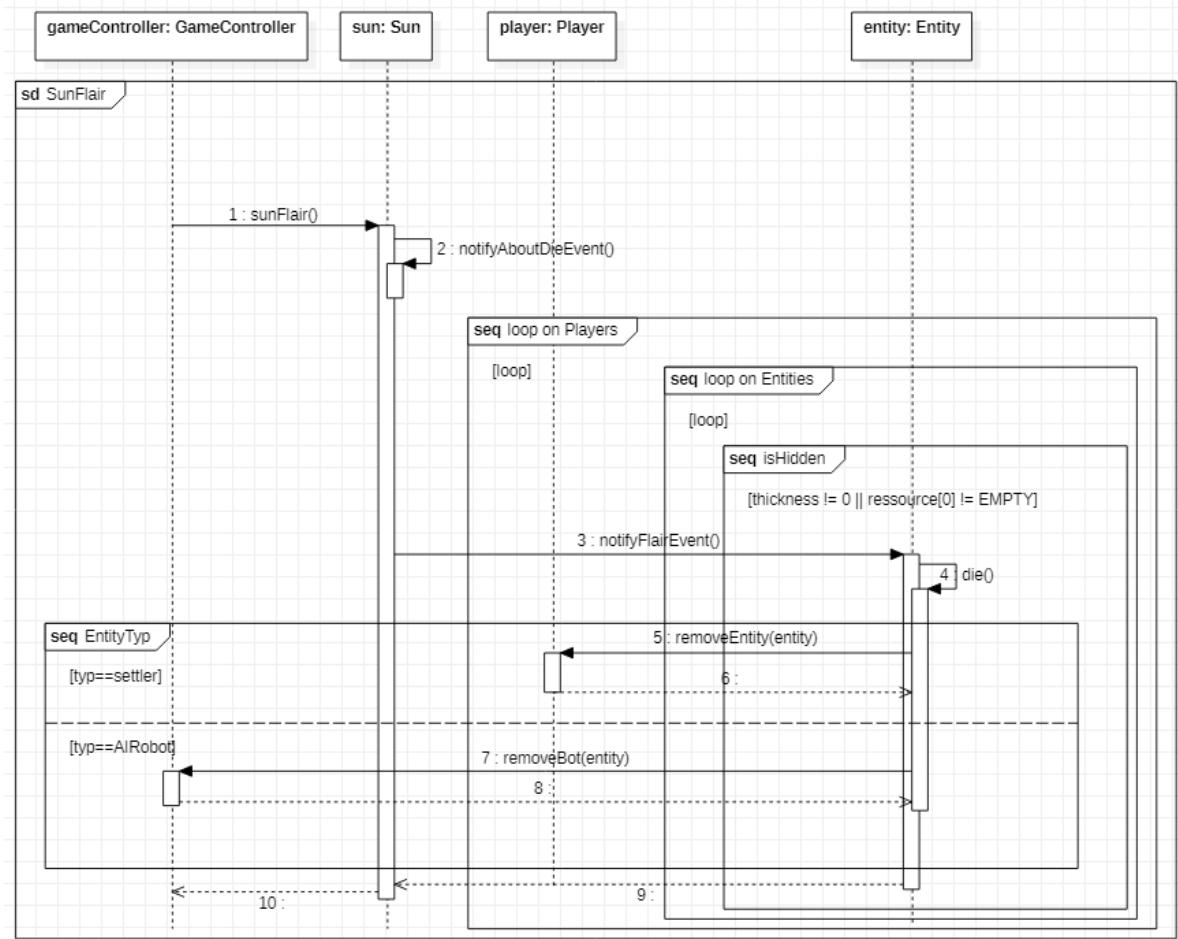
4.4.10 DeployResource



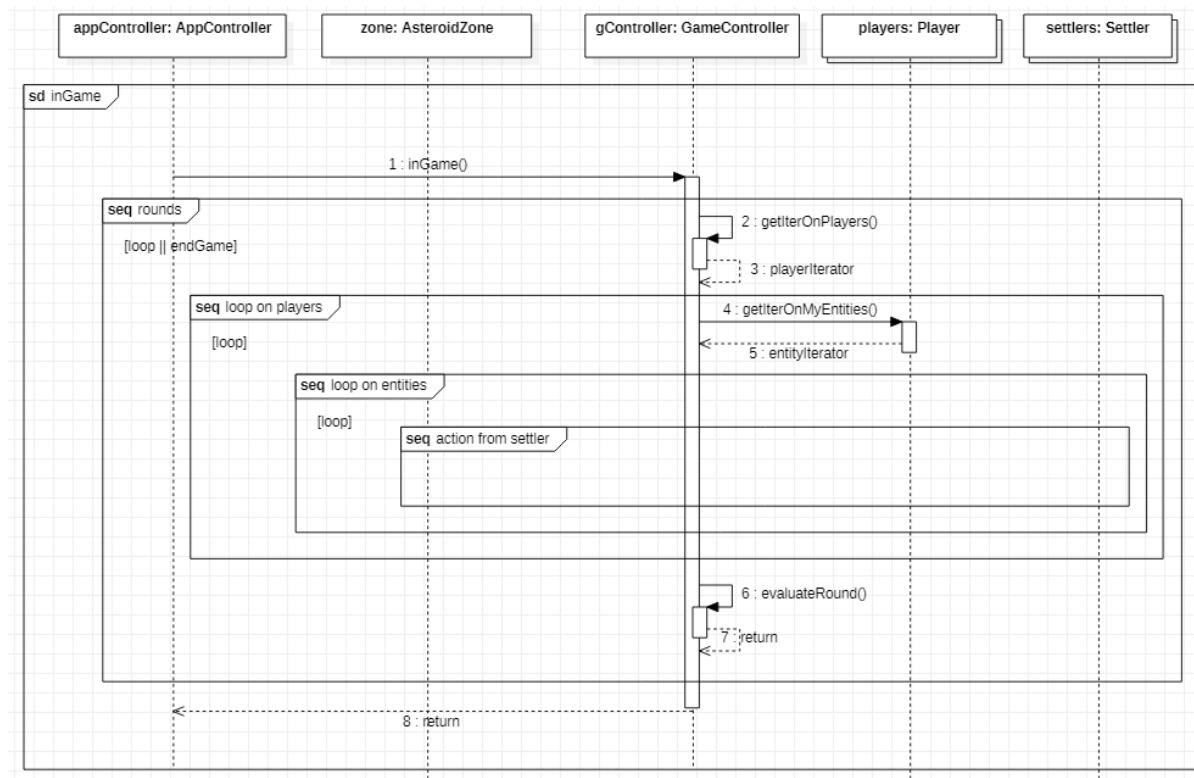
4.4.11 Mining



4.4.12 SunFlair

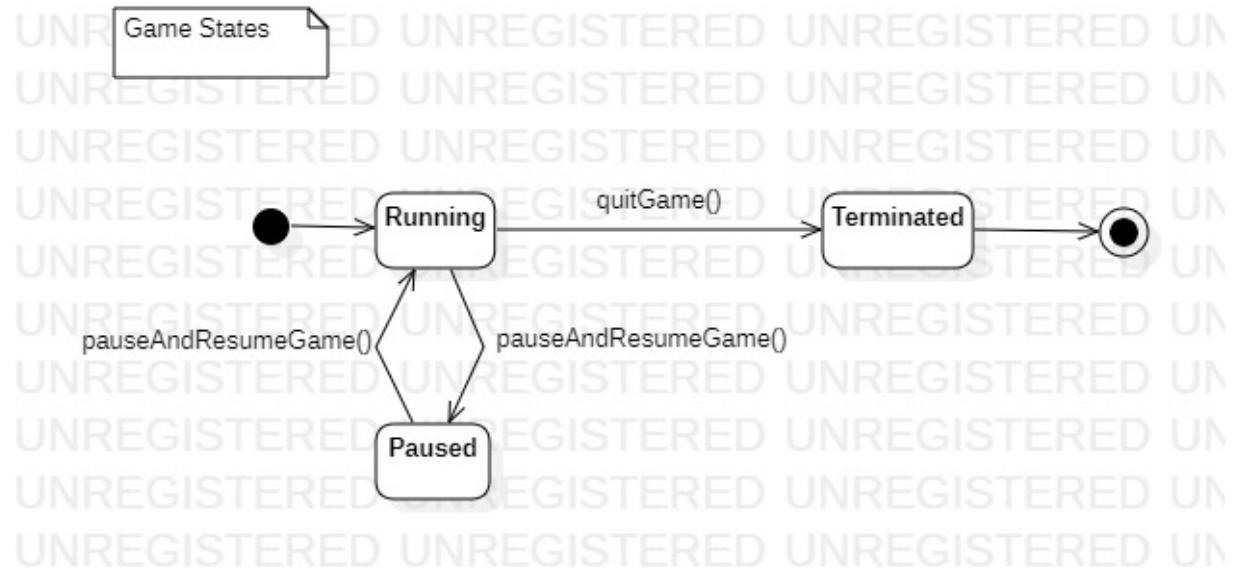


4.4.13 InGame

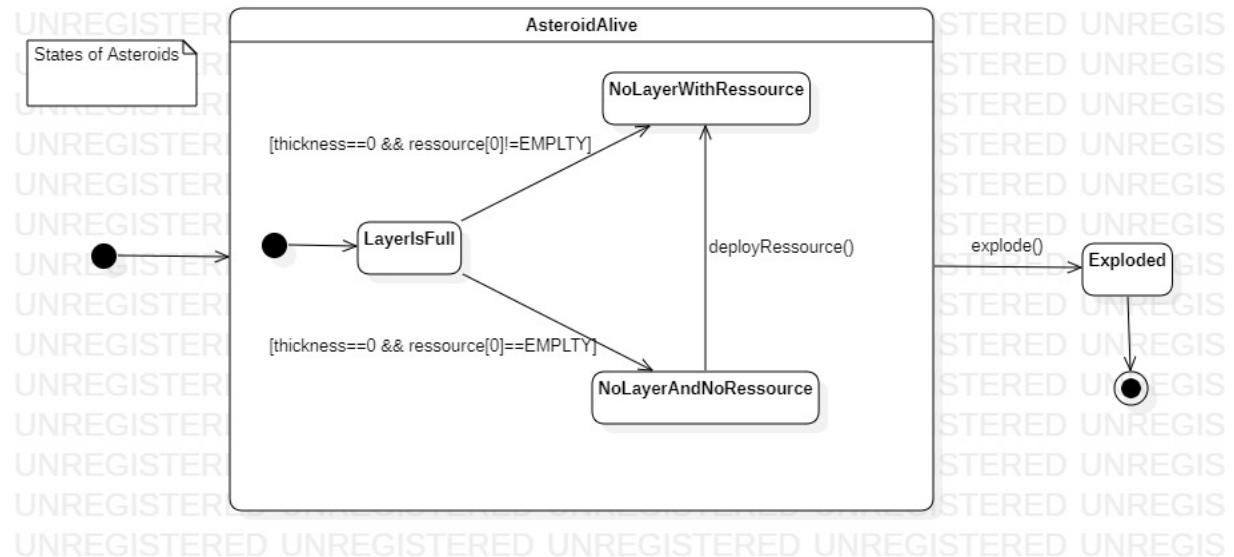


4.5 State-chart Diagramme

4.5.1 Game States



4.5.2 Asteroid States



4.6 Tagebuch

Anfang	Zeitaufwand	Teilnehmer	Beschreibung
2021.02.17. 15:00	2 Stude	Domokos Borbély Hrotkó Pongrácz	Meeting: Aufgabeausteilung auf diesem Etap, Jira eingeführt, über Klassendiagramm sprechen
2021.02.19. 13:00	2 Stunde	Domokos Borbély Hrotkó Pongrácz	Besprechen von Class Diagram
2021.02.20	2 Stunde	Pongrácz	Sequence Diagramme herstellen
2021.02.20	1 Stunde	Hrotkó	Auflistung und allg. Beschr. der Objekten
2021.02.21	2 Stunde	Pongrácz	Sequence Diagramme herstellen, kleine Änderungen in Klassendiagramm
2021.02.21	1,5 Stunde	Hrotkó	Sequenzdiagramme herstellen
2021.02.22.	1 Stunde	Borbély	Beschreibung der Klassen
2021.02.22.	1 Stunde	Domokos	Beschreibung der Klassen
2021.02.23.	1 Stunde	Domokos	Beschreibung der Klassen
2021.02.23.	1 Stunde	Pongrácz	Kleinere Änderungen, Klassenbeschreibung
2021.02.23.	1 Stunde	Borbély	Beschreibung der Klassen
2021.02.23	1 Stunde	Hrotkó	State Chart diagramme hestellen, durchlesen und kontrollieren das ganze Dokument.
2021.02.23	0,5 Stunde	Hrotkó Pongrácz Borbély Domokos	Aufgabenteil abgeschlossen, Diskussion
2021.02.25	1 Stunde	Pongrácz	Kleinere Änderungen
2021.03.02.	0,5 Stunde	Domokos	Kleinere Änderungen (3 → 4)
2021.03.02.	2 Stunde	Pongrácz	IO-Problemen im Skeleton lösen, GitHub Repository reinigen

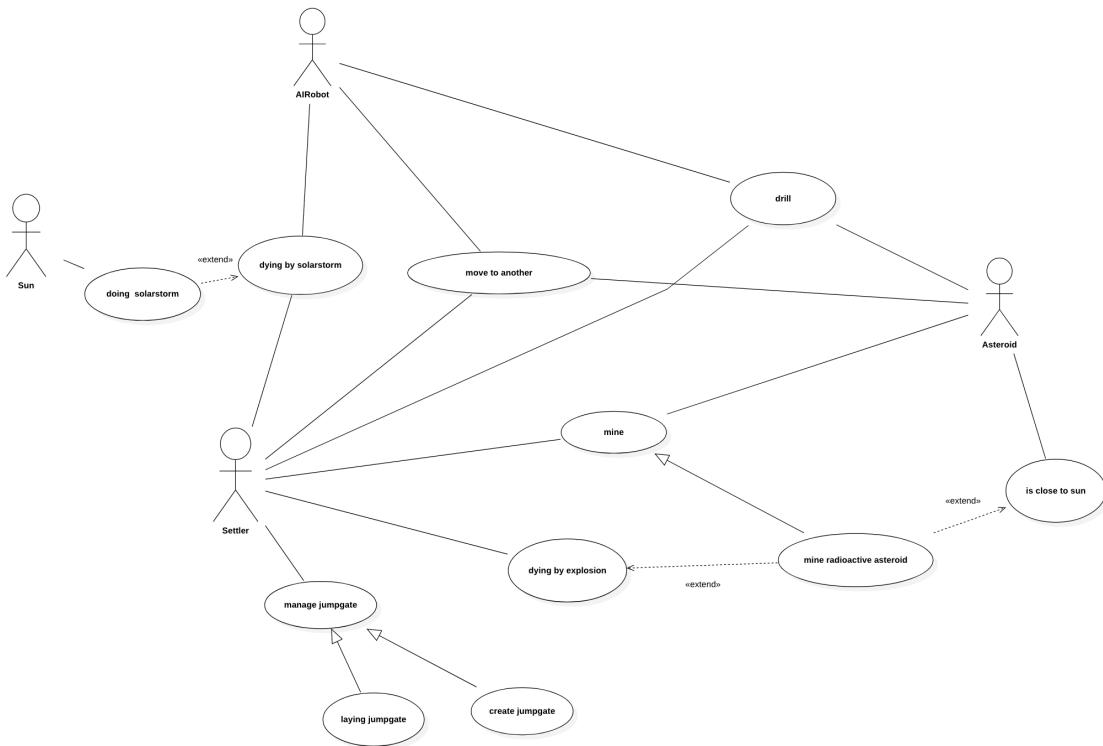
2021.02.23	1 Stunde	Hrotkó Pongrácz Borbély Domokos	Diskussion, Refactoring: über den gegebene Issues, Lösungen und Änderungen nach Skeletonkode diskutieren

5. Planung von Skeleton

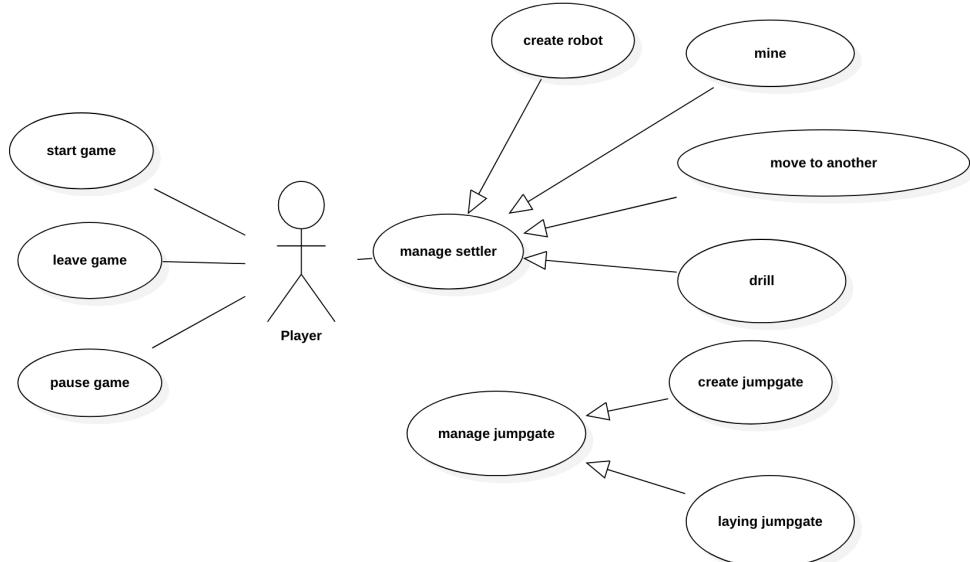
5.1 Die wirkliche Use-case Diagramme des Skeleton Models

5.1.1 Use-case Diagram

InGame Use-Case



User/Player Use-Case



5.1.2 Use-case Beschreibungen

InGame Use-Case

Name des Use-case	Drill
Kurzbeschreibung	Ein Settler oder ein AIRobot bohrt den Mantel der Asteroide.
Aktoren	AIRobot, Settler, Asteroid
Drehbuch	Ein Settler oder ein AIRobot während des Bohrens macht den Mantel der Asteroide 1 Einheit kleiner.

Name des Use-case	Move to another
Kurzbeschreibung	Sie können sich auf einem Asteroiden bewegen.
Aktoren	AIRobot, Settler, Asteroid
Drehbuch	Der Siedler und der Asteroid können sich von einem Asteroiden zum anderen bewegen.

Name des Use-case	Dying by solarstorm
Kurzbeschreibung	Sie können an einem Solarstorm sterben.
Aktoren	AIRobot, Settler
Drehbuch	Es gibt Solarstürme im Spiel, die killen, wenn sie den Siedler oder den Roboter erreichen.

Name des Use-case	Mine
Kurzbeschreibung	Der Siedler baut den Asteroiden ab.
Aktoren	Settler, Asteroid
Drehbuch	Der Siedler fördert die Ressource aus dem Kern der Asteroide.

Name des Use-case	Mine radioactive asteroid
Kurzbeschreibung	Der Siedler baut einen radioaktiven Asteroiden ab.
Aktoren	AIRobot, Settler, Asteroid
Drehbuch	Der Siedler kann einen radioaktiven Asteroiden abbauen, der explodiert, wenn er sich in der Nähe der Sonne befindet.

Name des Use-case	Dying by explosion
Kurzbeschreibung	Der Siedler stirbt bei der Explosion.
Aktoren	Settler
Drehbuch	Wenn ein Siedler versucht, einen radioaktiven Asteroiden in der Nähe der Sonne abzubauen, stirbt er bei seiner Explosion.

Name des Use-case	Manage jumpgate
Kurzbeschreibung	Der Settler handhabt das Teleporttor.
Aktoren	Settler
Drehbuch	Mit zwei Einheiten Eisen, einer Einheit Wassereis und einer Einheit Uran kann der Settler ein Paar Teleport Tore herstellen. Jedes Tor kann später vom Settler in der Nähe des gerade besuchten Asteroiden gestartet werden. Die beiden Mitglieder des Tor Paars sind verbunden und betreten eines, der Settler befindet sich im anderen. Frisch hergestellte Tore können vom Settler getragen werden, ein Settler kann jedoch maximal zwei Tore gleichzeitig haben.

Name des Use-case	Create jumpgate
Kurzbeschreibung	Der Settler baut einen Teleport Tor
Aktoren	Settler
Drehbuch	Mit zwei Einheiten Eisen, einer Einheit Wassereis und einer Einheit Uran kann der Settler ein Paar Teleport Tore herstellen.

Name des Use-case	Laying jumpgate
Kurzbeschreibung	Der Settler legt das Teleporttor ab.
Aktoren	Settler
Drehbuch	Der Settler kann ein Stück des Teleport-Gate-Paars neben einen Asteroiden legen.

Name des Use-case	Is close to sun
Kurzbeschreibung	Der Asteroid ist in der Nähe der Sonne.
Aktoren	Asteroid
Drehbuch	Der Asteroid kann in der Nähe der Sonne sein. Wenn ein Siedler versucht, einen radioaktiven Asteroiden in der Nähe der Sonne abzubauen, stirbt er bei seiner Explosion.

Name des Use-case	Doing solarstorm
Kurzbeschreibung	Die Sonne kann einen Solarstorm verursachen
Aktoren	Sun
Drehbuch	Die Sonne verursacht manchmal eine Solarstorm, die stirbt, wenn sie den AIRobot oder den Siedler erreichen.

User/Player Use-Case

Name des Use-case	Start game
Kurzbeschreibung	Das Spiel wird gestartet.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt START GAME Taste. 2. Spieler wartet 3. Spiel fängt an.

Name des Use-case	Leave game
Kurzbeschreibung	Der Spieler tritt aus dem Spiel. Das Spiel endet.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt LEAVE GAME Taste. 2. Das Spieler beendet das Spiel.

Name des Use-case	Pause game
Kurzbeschreibung	Der Spieler kann eine Spielpause einlegen. Das Spiel stoppt, die Ergebnisse bleiben erhalten, bis der Spieler die Pause unterbricht.
Aktoren	Player
Drehbuch	<ol style="list-style-type: none"> 1. Benutzer drückt PAUSE GAME Taste. 2. Das Spiel stoppt.

Name des Use-case	Create robot
Kurzbeschreibung	Der Spieler baut einen Roboter
Aktoren	Player
Drehbuch	Siedler können mit einer Einheit Eisen, einer Einheit Kohle und einer Einheit Uran einen autonomen Roboter schaffen, der durch künstliche Intelligenz gesteuert wird. Spieler können die Roboter nicht kontrollieren.

Name des Use-case	Manage jumpgate
Kurzbeschreibung	Der Spieler handhabt das Teleporttor.
Aktoren	Player
Drehbuch	Mit zwei Einheiten Eisen, einer Einheit Wassereis und einer Einheit Uran kann der Spieler ein Paar Teleport Tore herstellen. Jedes Tor kann später vom Spieler in der Nähe des gerade besuchten Asteroiden gestartet werden. Die beiden Mitglieder des Tor Paars sind verbunden und betreten eines, der Spieler befindet sich im anderen. Frisch hergestellte Tore können vom Spieler getragen werden, ein Spieler kann jedoch maximal zwei Tore gleichzeitig haben.

Name des Use-case	Create jumpgate
Kurzbeschreibung	Der Spieler baut einen Teleport Tor
Aktoren	Player
Drehbuch	Mit zwei Einheiten Eisen, einer Einheit Wassereis und einer Einheit Uran kann der Spieler ein Paar Teleport Tore herstellen.

Name des Use-case	Laying jumpgate
Kurzbeschreibung	Der Spieler legt das Teleporttor ab.
Aktoren	Player
Drehbuch	Der Spieler kann ein Stück des Teleport-Gate-Paars neben einen Asteroiden legen.

Name des Use-case	Manage Settler
Kurzbeschreibung	Der Spieler kontrolliert die Siedler.
Aktoren	Player
Drehbuch	Der Spieler hat mehrere Siedler. Sie können sie zu einem anderen Asteroiden oder einen JumpGate bewegen, den Asteroiden abbauen und den Asteroiden bohren.

Name des Use-case	Move Settler
Kurzbeschreibung	Der Spieler bewegt die Siedler.
Aktoren	Player
Drehbuch	Der Siedler kann entweder den Asteroiden oder durch einen Teleport Tor bewegen.

Name des Use-case	Drill Asteroid
Kurzbeschreibung	Der Siedler bohrt den Asteroiden.
Aktoren	Player
Drehbuch	Der Spieler während des Bohrens macht den Mantel der Asteroide 1 Einheit kleiner.

Name des Use-case	Mine Asteroid
Kurzbeschreibung	Der Siedler baut den Asteroiden ab.
Aktoren	Player
Drehbuch	Der Siedler fördert der Ressource aus dem Kern der Asteroide.

5.2 Planung der Oberfläche des Skeletons und Dialoge

In unserem Fall wird diese Skeleton Menügesteuert sein. Es gibt immer eine Liste aus denn wir die ausführbare Tätigkeiten auswählen können.

Am Anfang ist es nur das Start() Funktion. Danach kann nur das createZone kommen. Nachdem wir eine voll aufgebaute Asteroidenzone haben, können die andere Szenarien folgen:

1.1	Start	Das Start des Spiels und es ruft den CreateZone Sequence Folge.
1.2	CreateZone	<i>Nach dem Start wird aufgerufen und es herstellt das ganze Spielfeld mit Asteroiden und die Sonne.</i>
3.1	Move	In diesem Fall wird sich der Settler auf eine benachbarte Asteroide bewegen.
4.1	Drill	Der Mantel der Asteroide wird mit ein Einheit verringert. Falls es radioaktiv ist dann kommt die Explode Sequence
4.2	Explode	<i>Nachdem der Mantel durchbohrt wurde, und falls der Rohstoff in Kern radioaktiv und Sonnennah ist dann kommt die Explosion Sequence</i>
4.3	(6 – Move)	<i>Nach dem Explosion wird der automatische Move Sequence kommen falls der Entität ein Roboter war.</i>
5.1	CreateBot	In diesem Sequence wird gezeigt wie die Herstellung eines Roboters geht.
6.1	ListMyNeighbours	Es zeigt wie die Auslistung der benachbarten Asteroiden folgt.
7.1	CreateGate	In diesem Sequence wird gezeigt wie die Herstellung einer Portale geht.
8.1	BuildGate	Hier wird es gezeigt wie ein Portal aufgebaut werden kann.
9.1	DeployRessource	In diesem Szenario kann man sehen wie das Zurückstecken des Rohstoffes geschieht.

10.1	SunFlair	<i>Es stellt vor wie sieht das aus falls ein SunFlair Event kommt.</i>
------	----------	--

Der Benutzer soll die Zahl der Befehl ausgeben zB. Bewegen -1, Fördern – 2 usw... Oder in bestimmten Fällen können auch richtig oder falsch Fragen vorkommen zB die Funktionen die etwas kontrollieren (Ende der Runde). Beim Auswählen einer solchen Tätigkeit wird der Name der Szenario ausgeschrieben und dann die Name der aufgerufene Funktion und Objekt folgenderweise:

{der Name des Objektes} {der Name der Funktion} {nächste Reihe Karakter}

Eine Beispiel folgt hier: Mining

Settler - mine()

,,Ist die Manteldicke zero (Also der Kortikalis wurde durchbohrt)?

true

Asteroid - mineRessource()

Core - getRessource()

Core - setRessource()

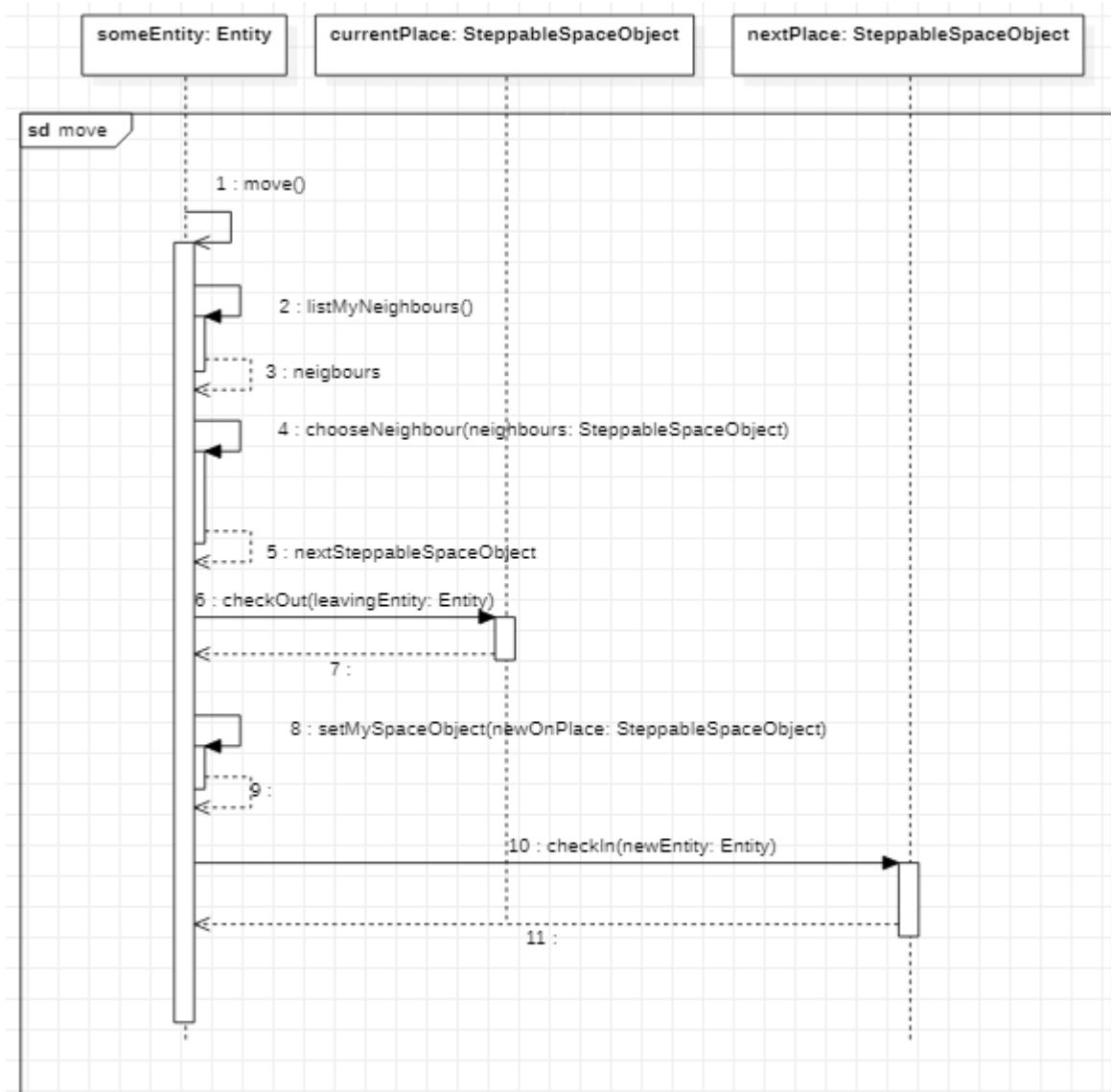
,,Gab es in dem Kern ein Rohstoff, den der Siedler fördern konnte? (also es war nicht leer)''

true

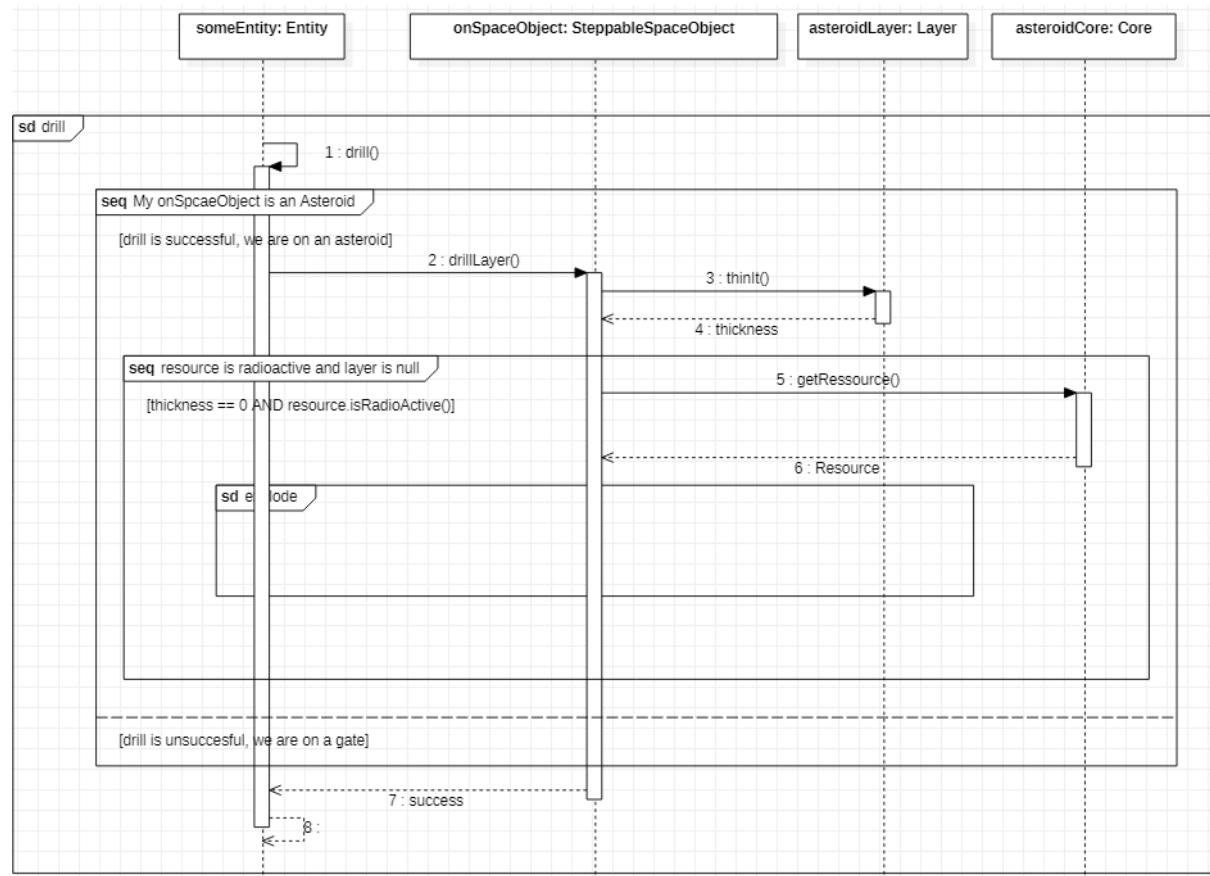
Settler - addRessource()

5.3 Sequence Diagramme für die innere Funktionierung

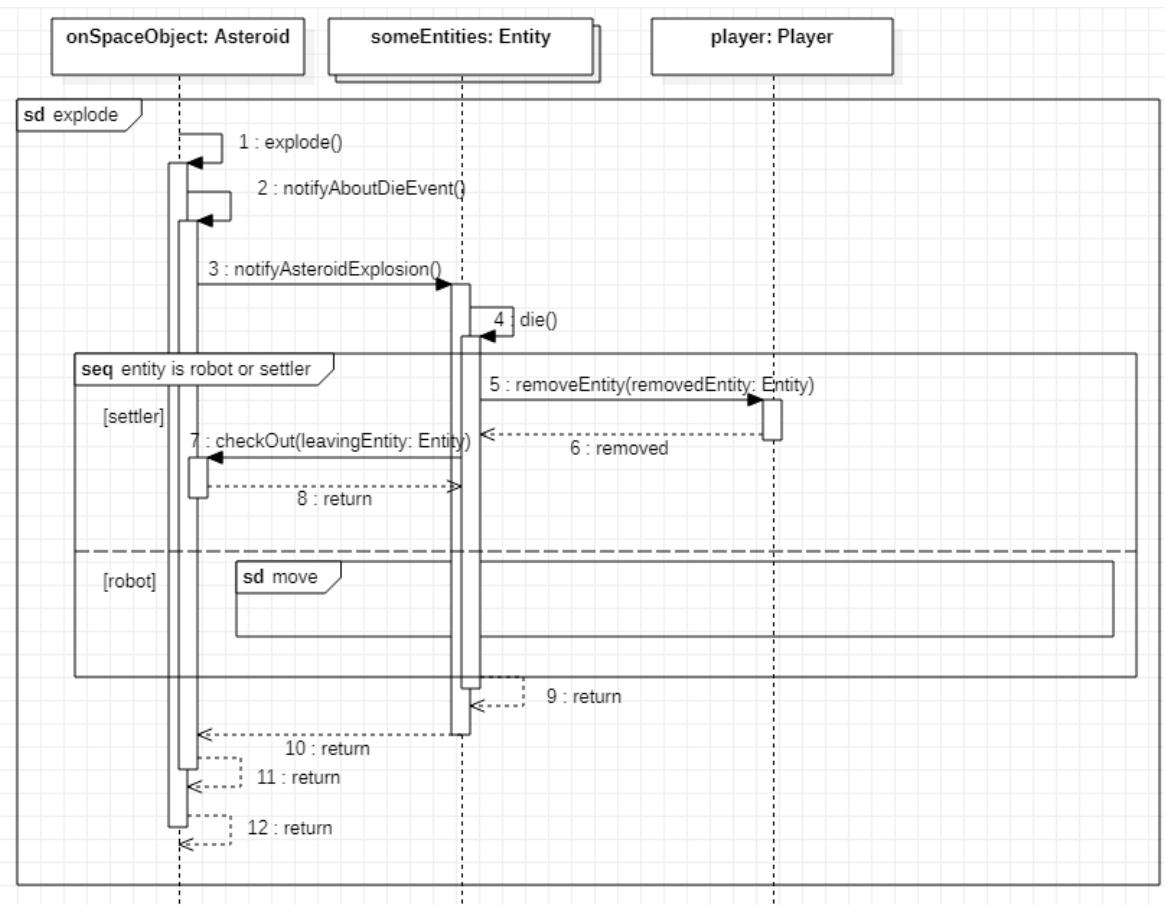
5.3.1 Move



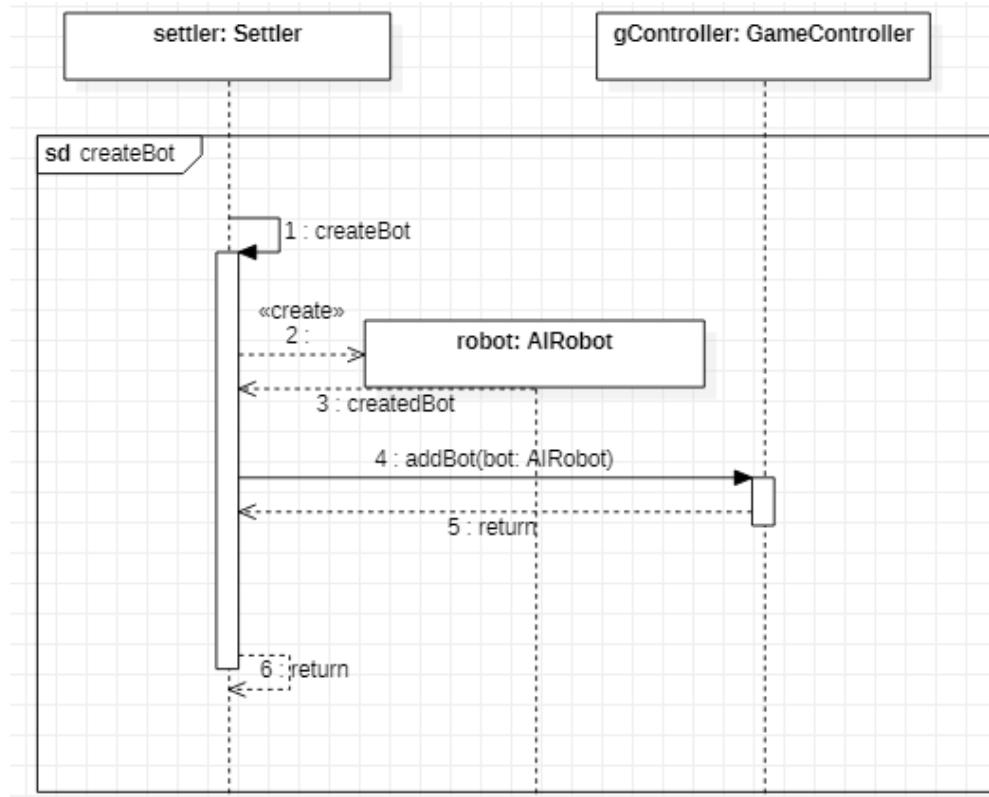
5.3.2 Drill



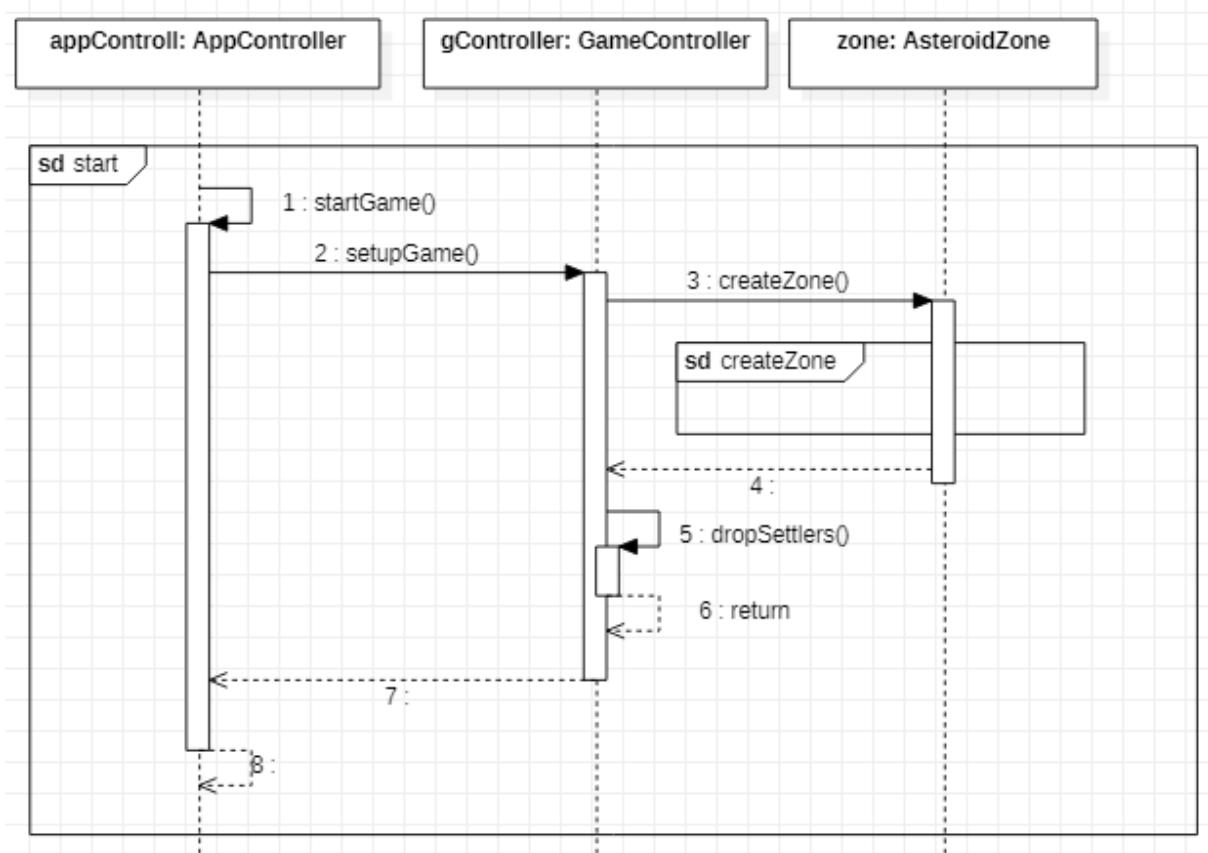
5.3.3 Explode



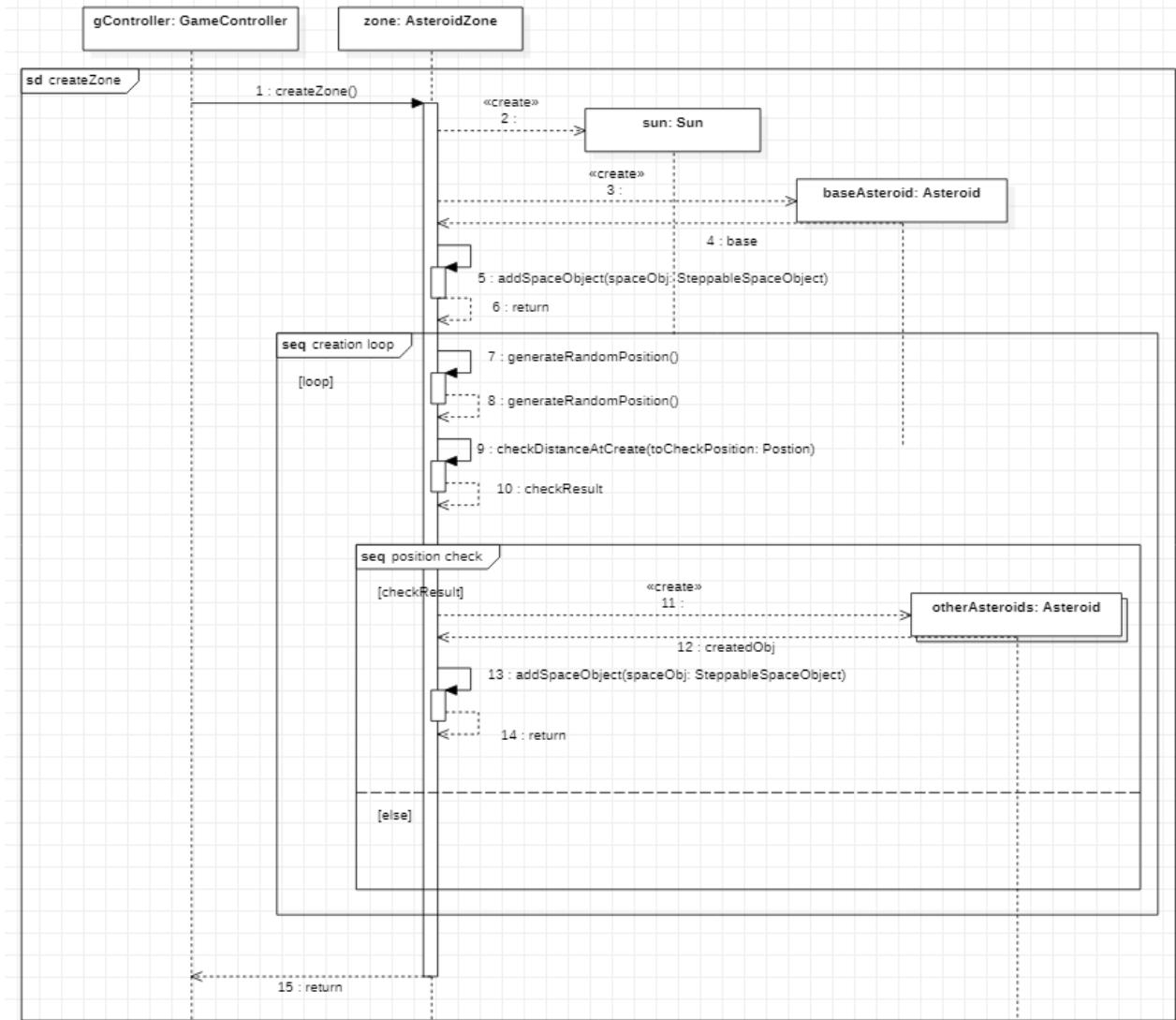
5.3.4 CreateBot



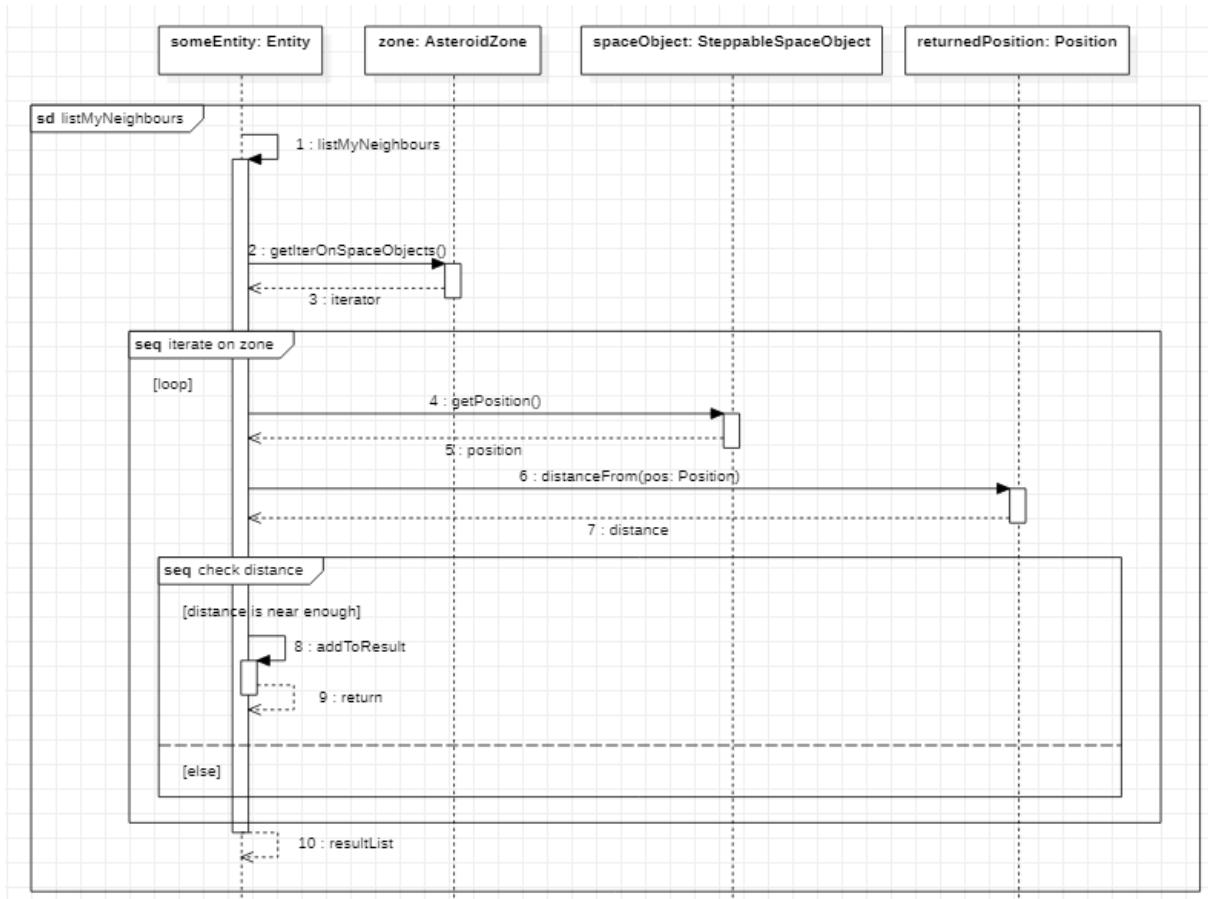
5.3.5 Start



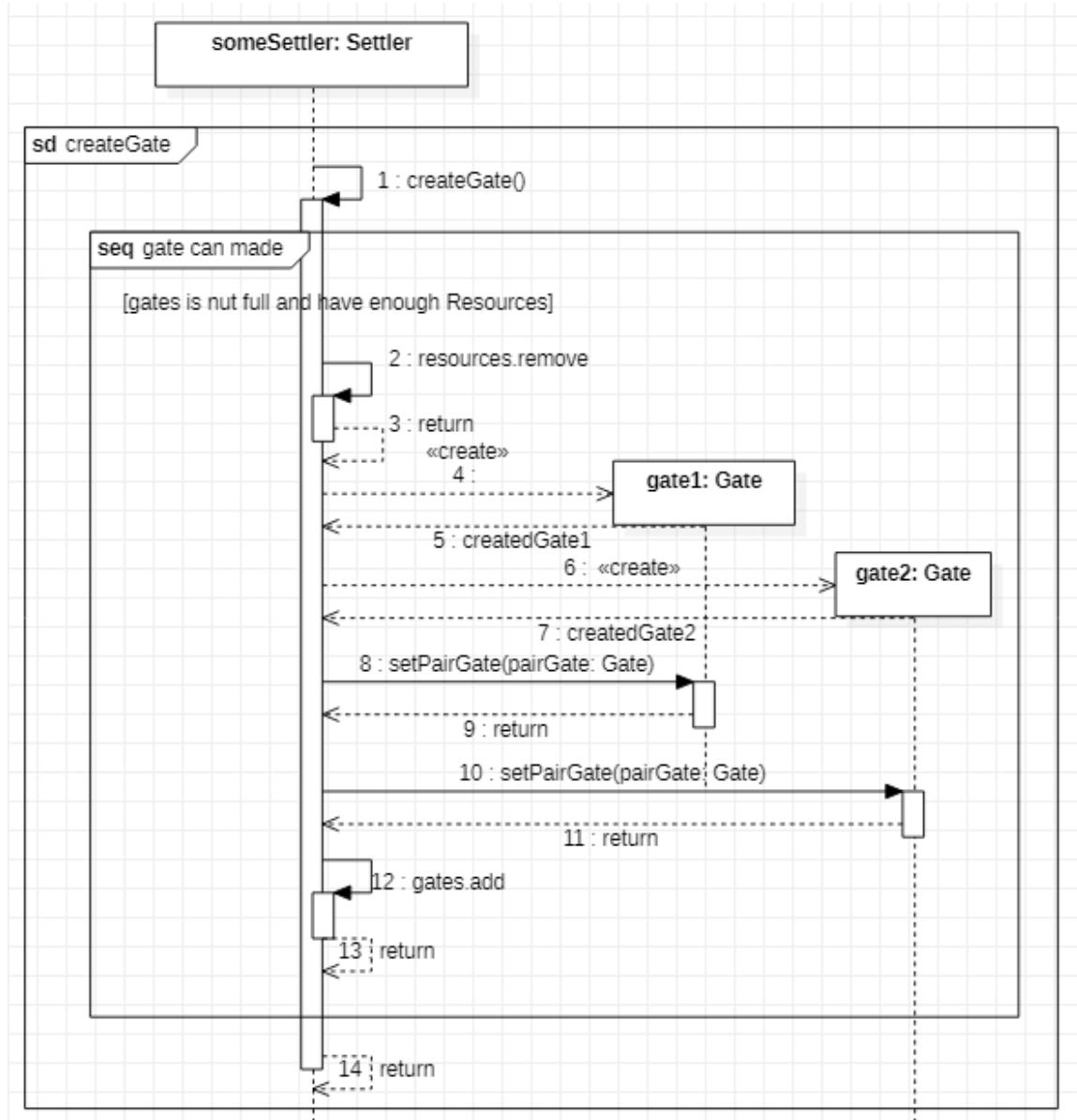
5.3.6 CreateZone



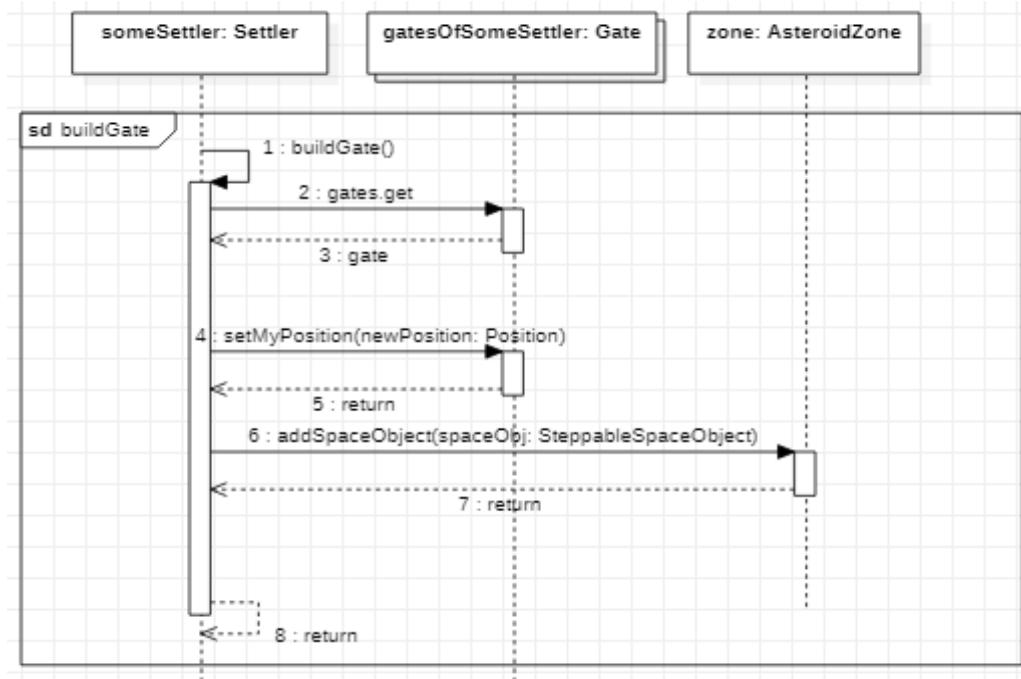
5.3.7 ListMyNeighbours



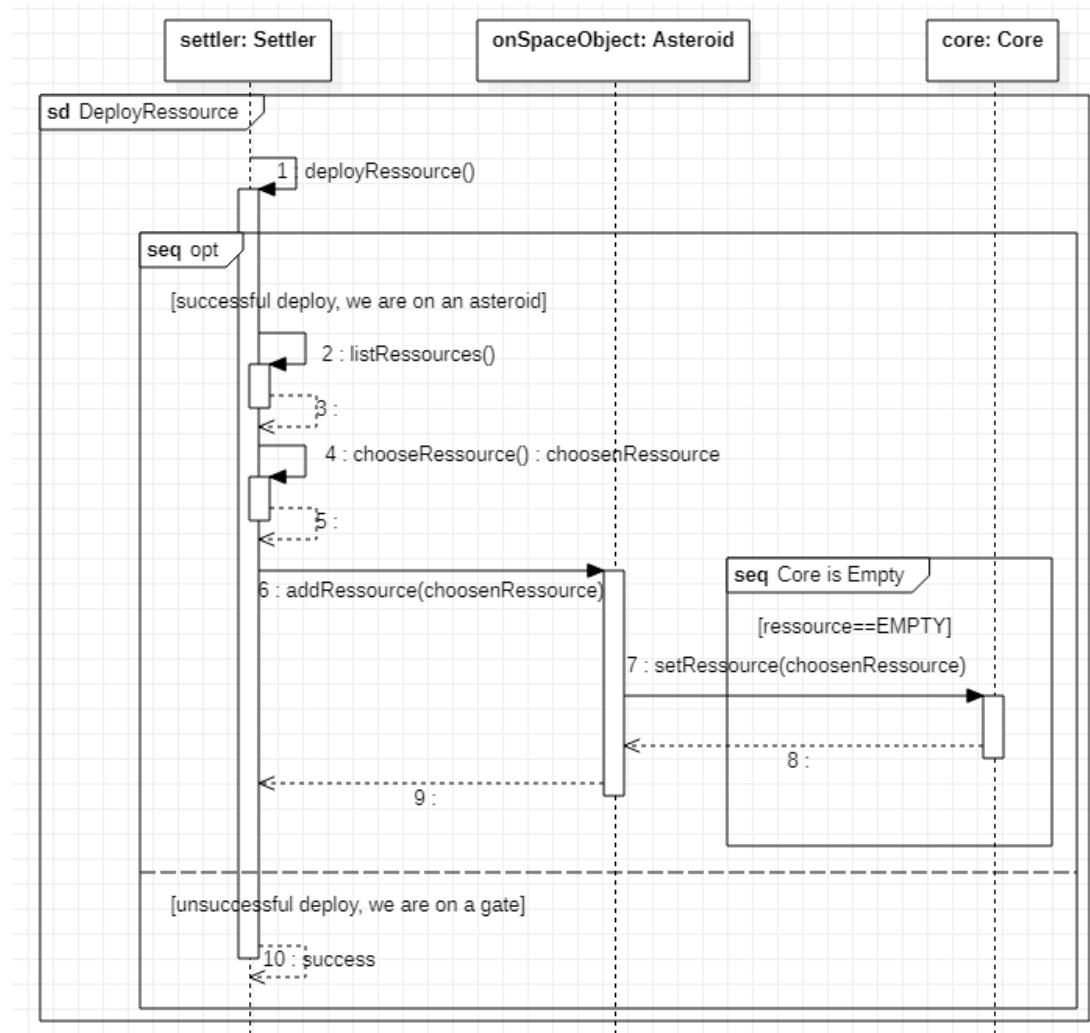
5.3.8 CreateGate



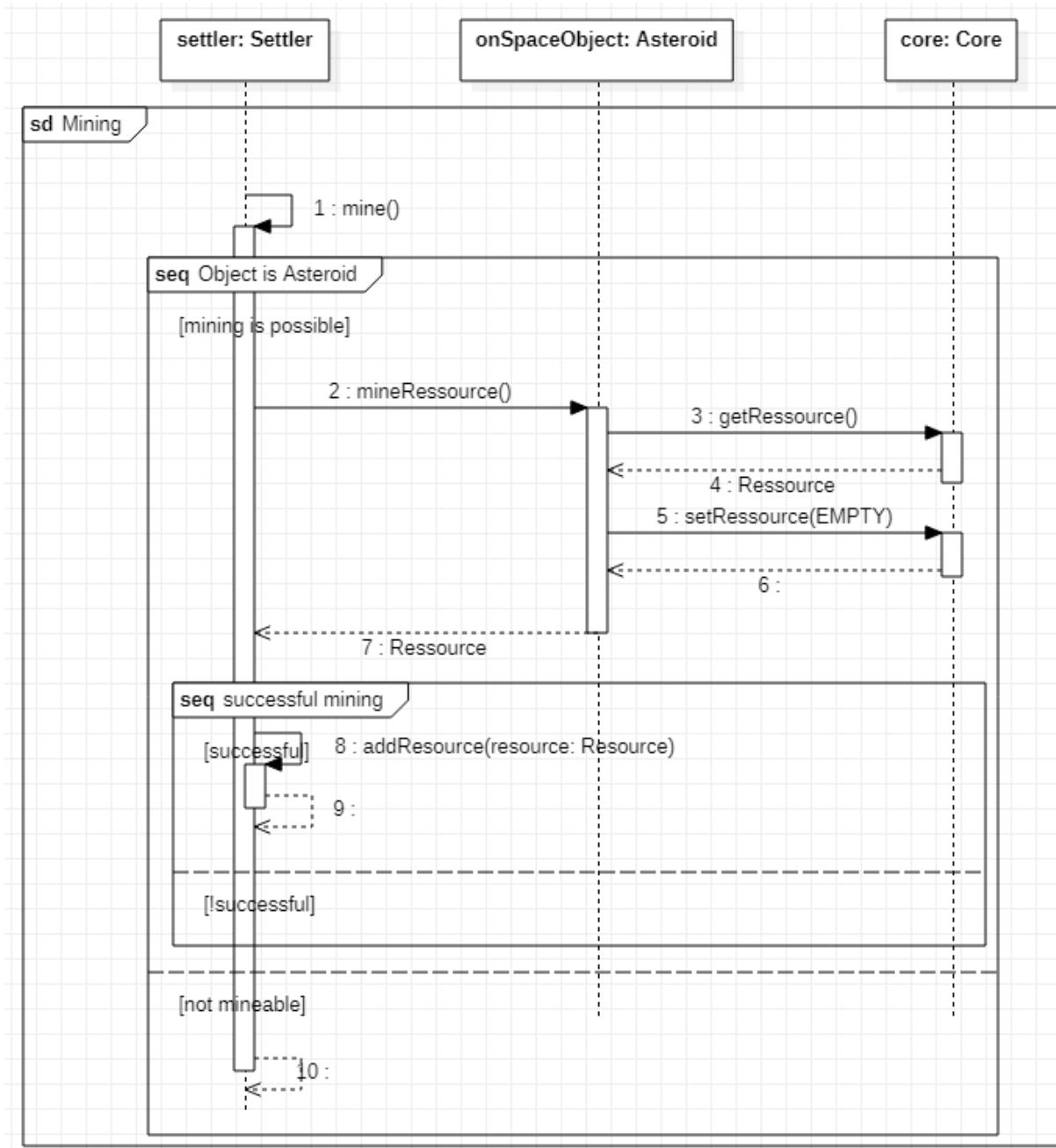
5.3.9 BuildGate



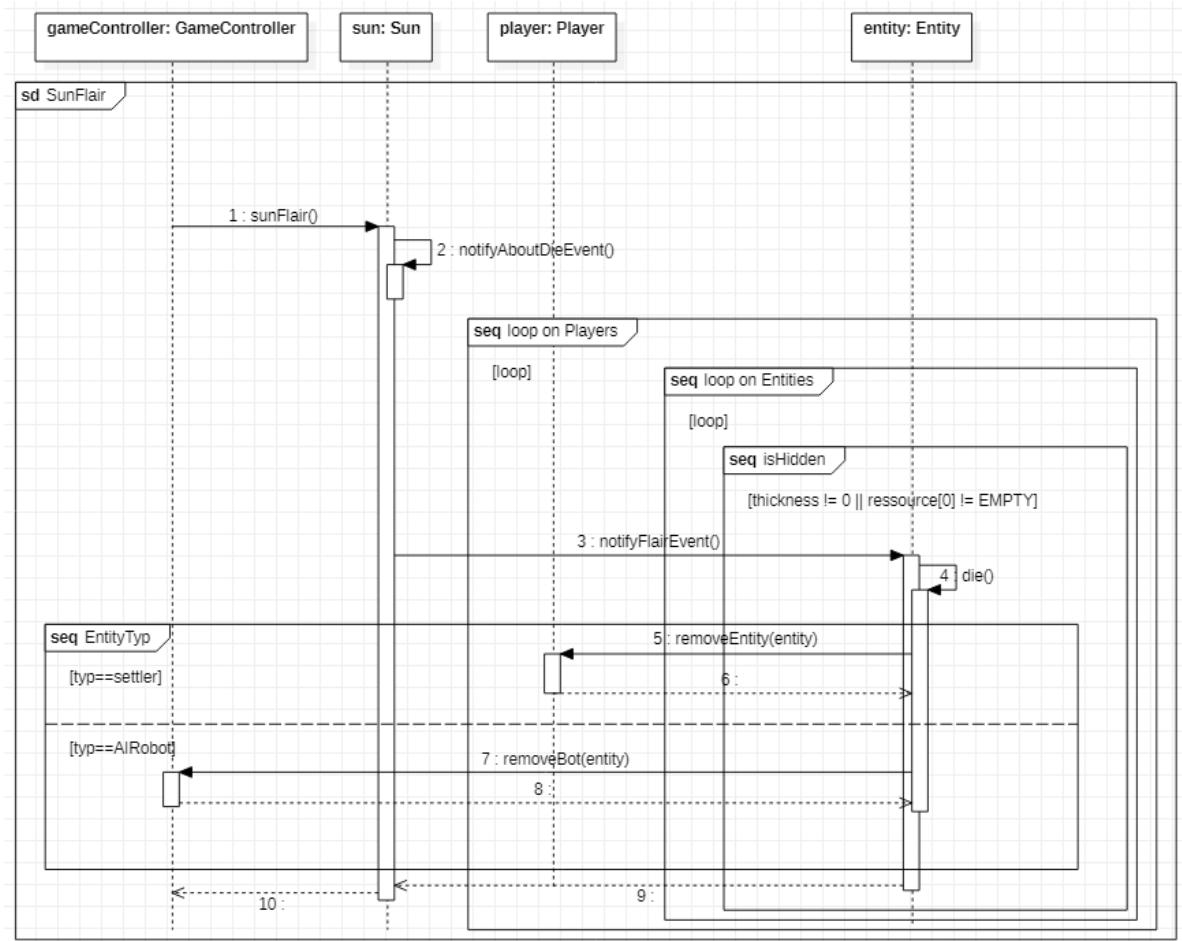
5.3.10 DeployResource



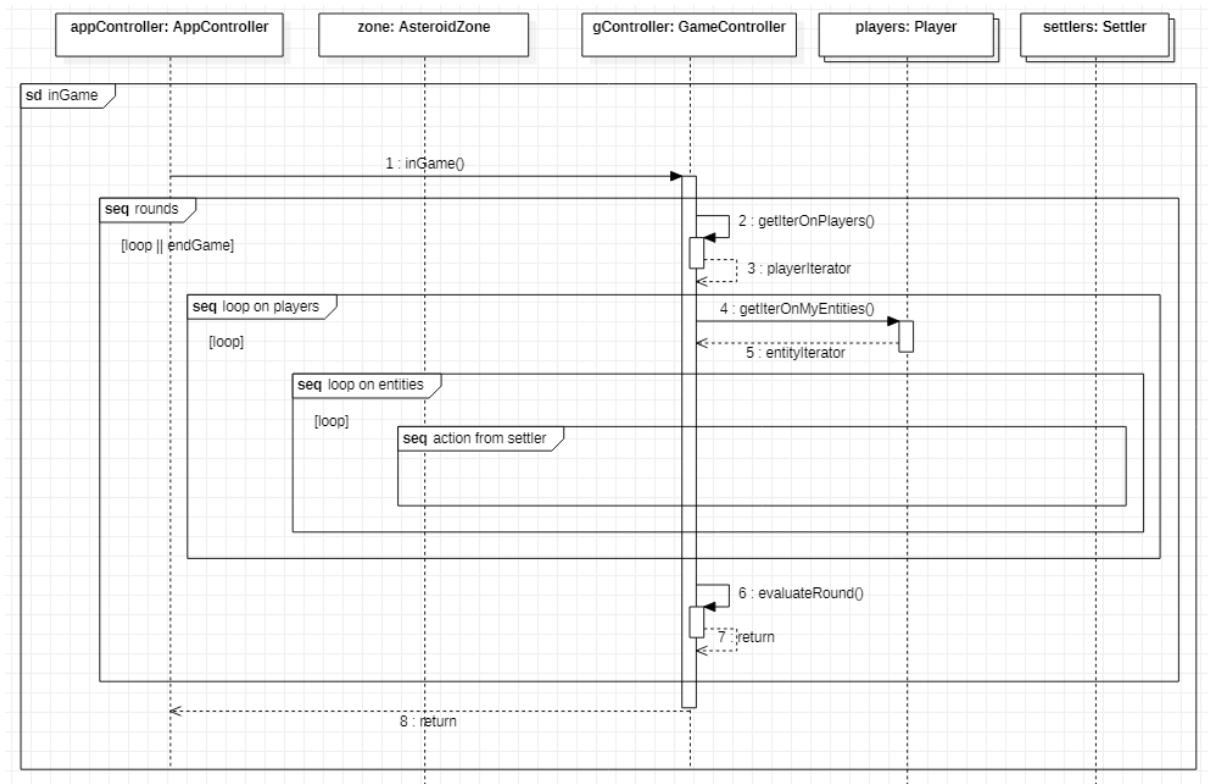
5.3.11 Mining



5.3.12 SunFlair

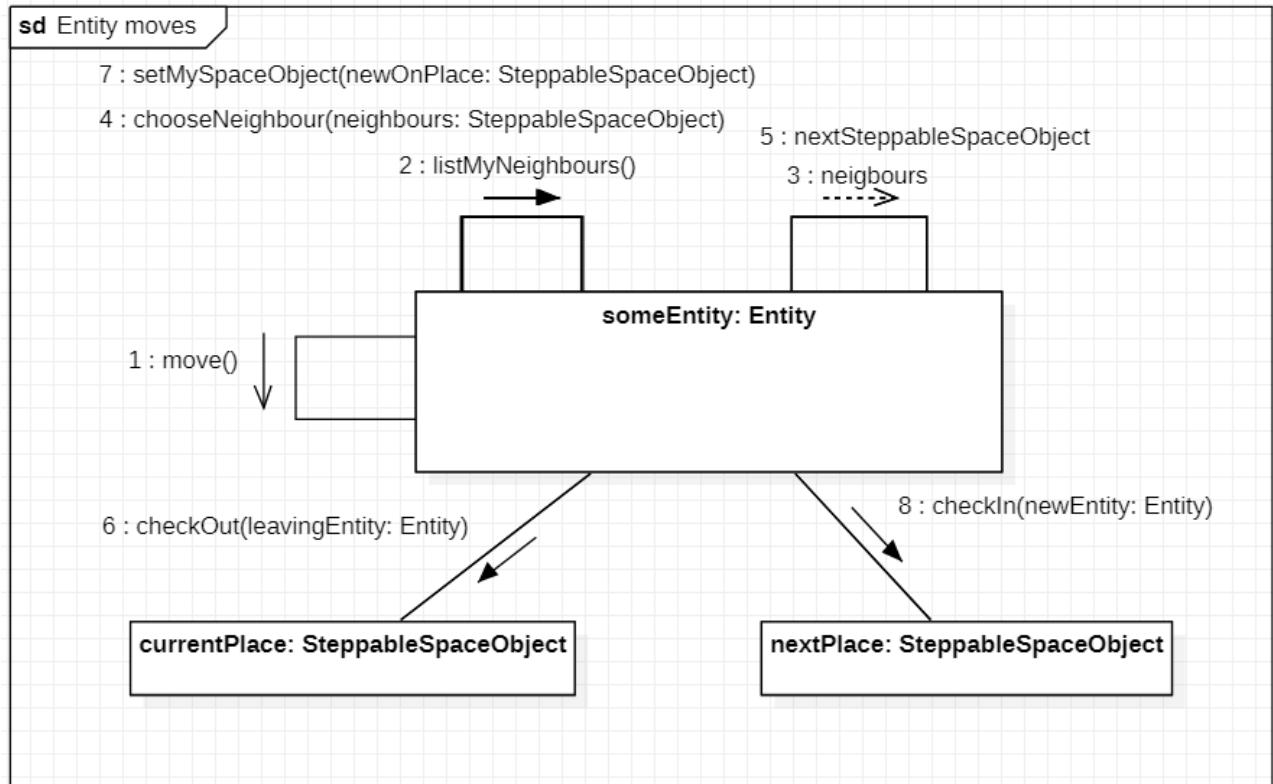


5.3.13 InGame

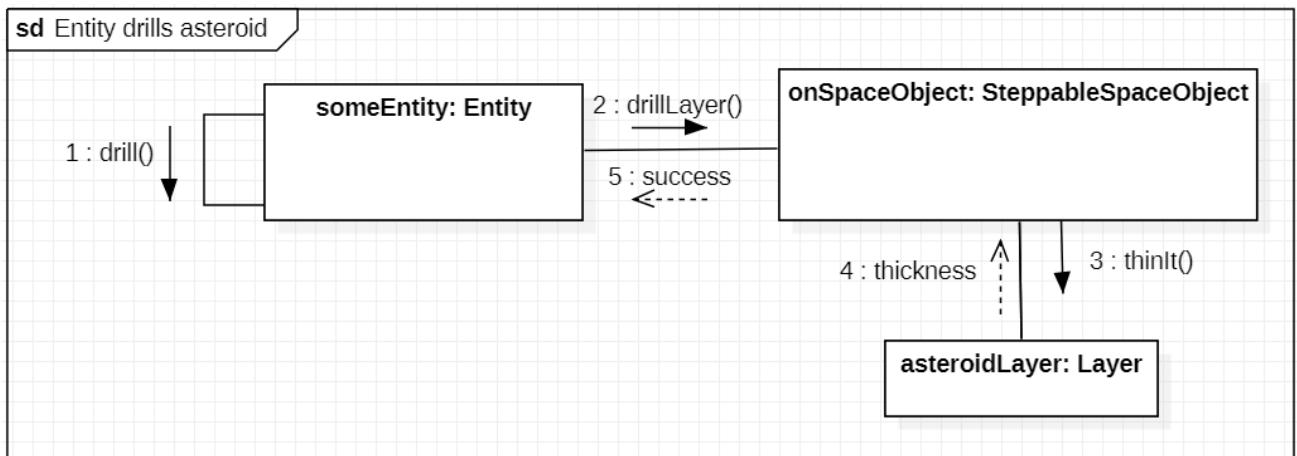


5.4 Kommunikation Diagramme

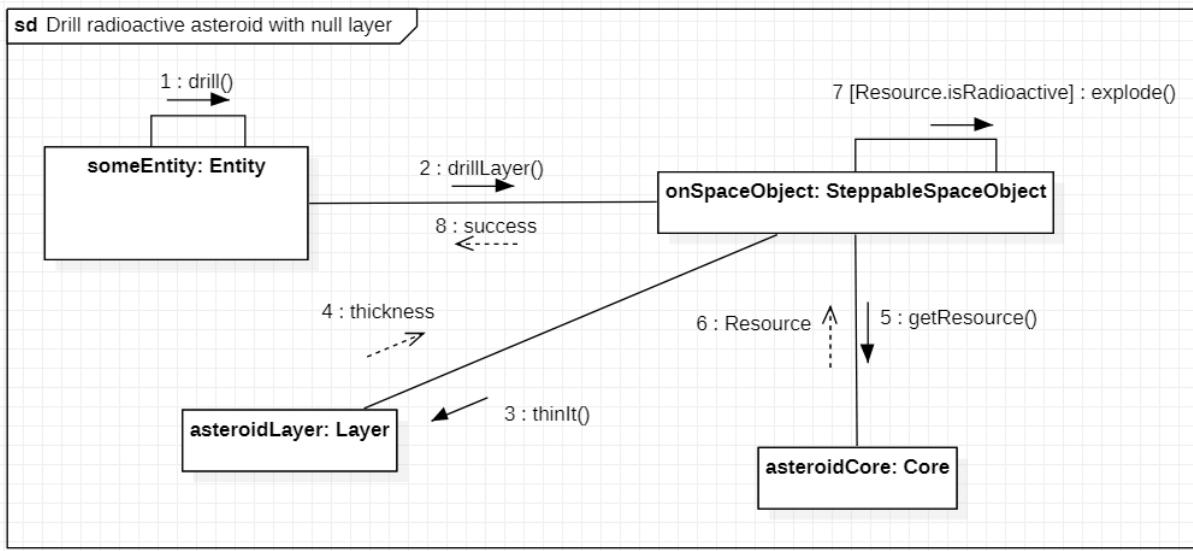
5.4.1 Entity moves



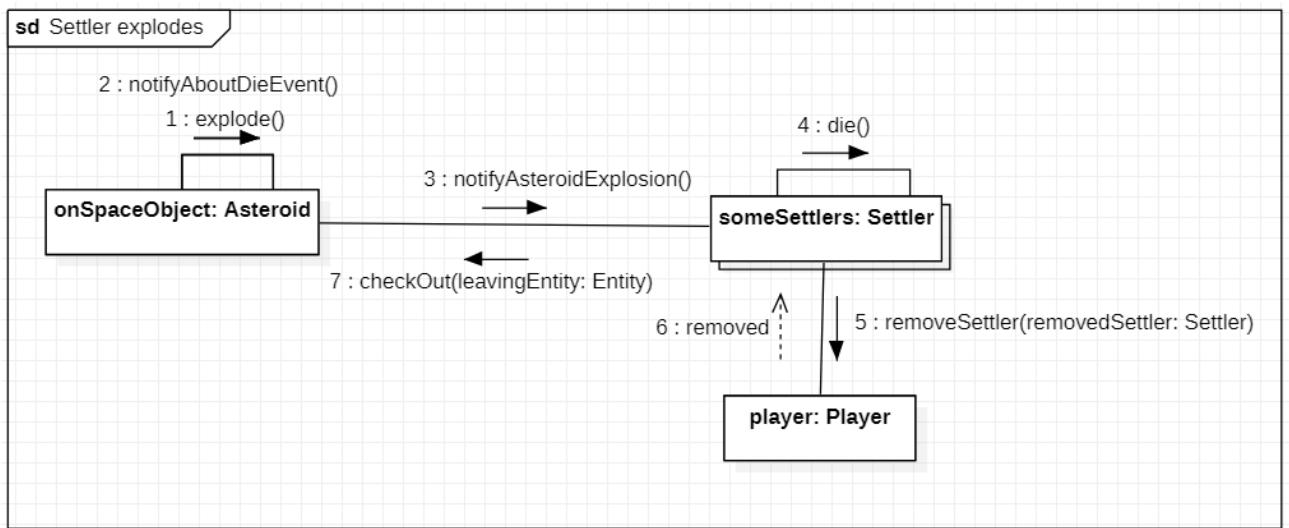
5.4.2 Entity drills asteroid



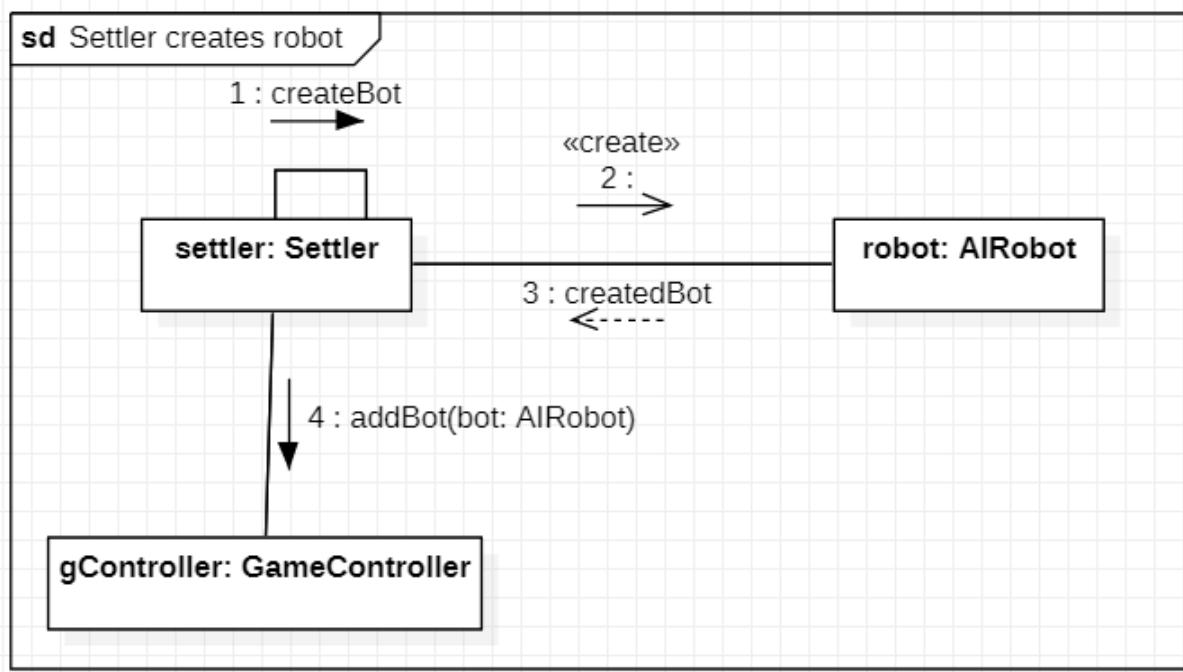
5.4.3 Entity drills asteroid with no layer near the Sun



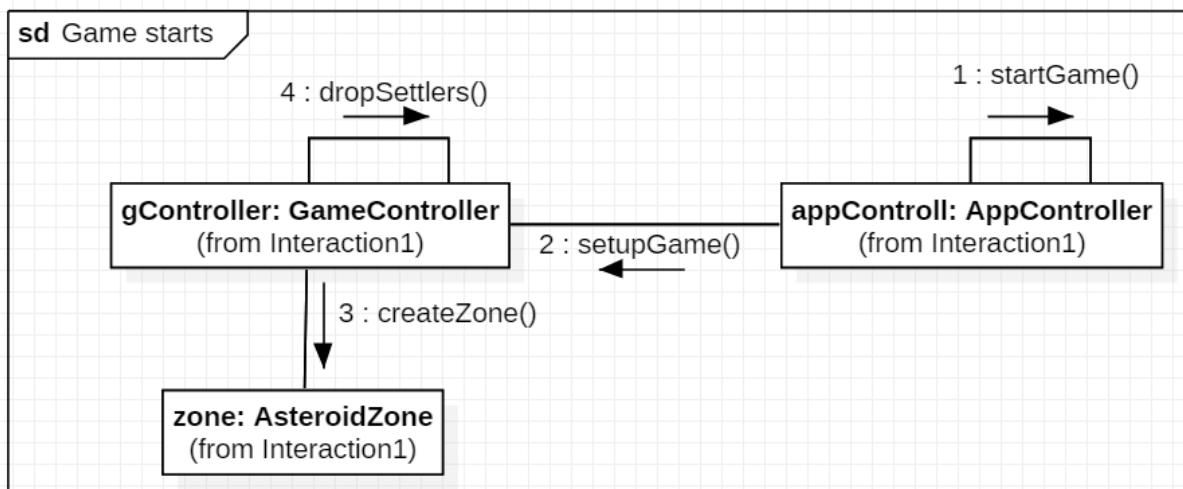
5.4.4 Settler explodes



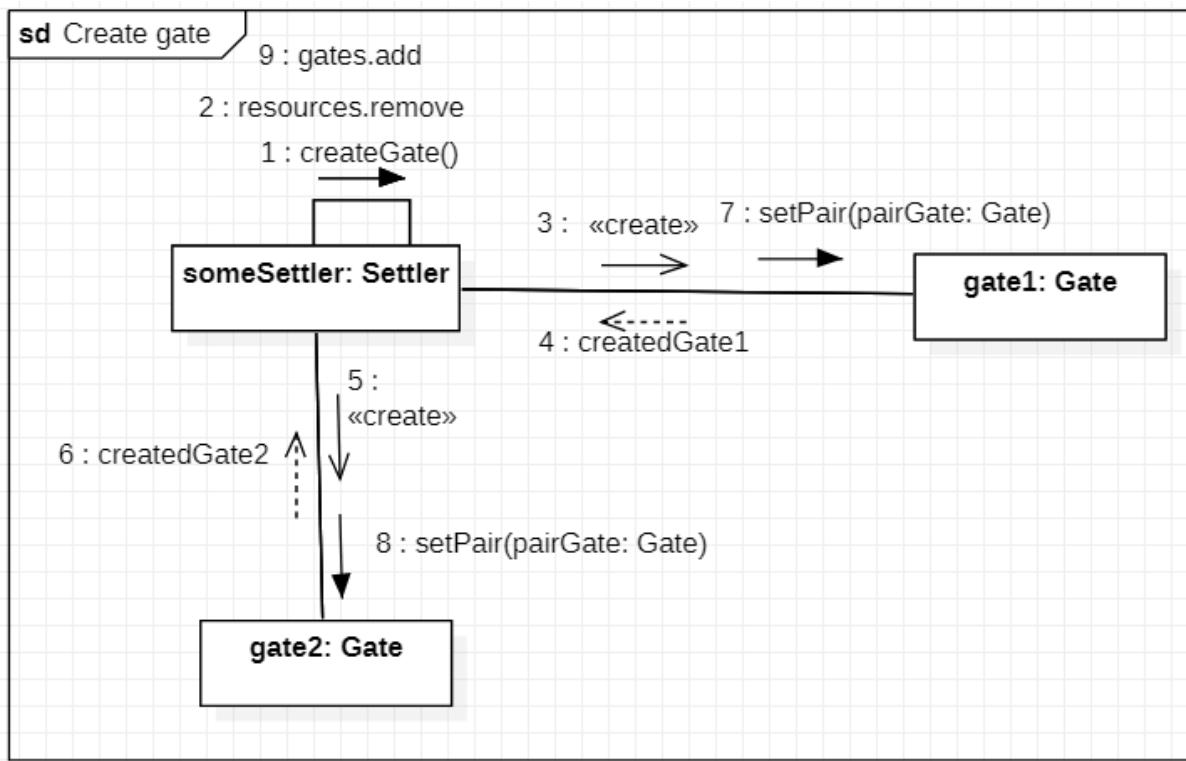
5.4.5 Settler creates robot



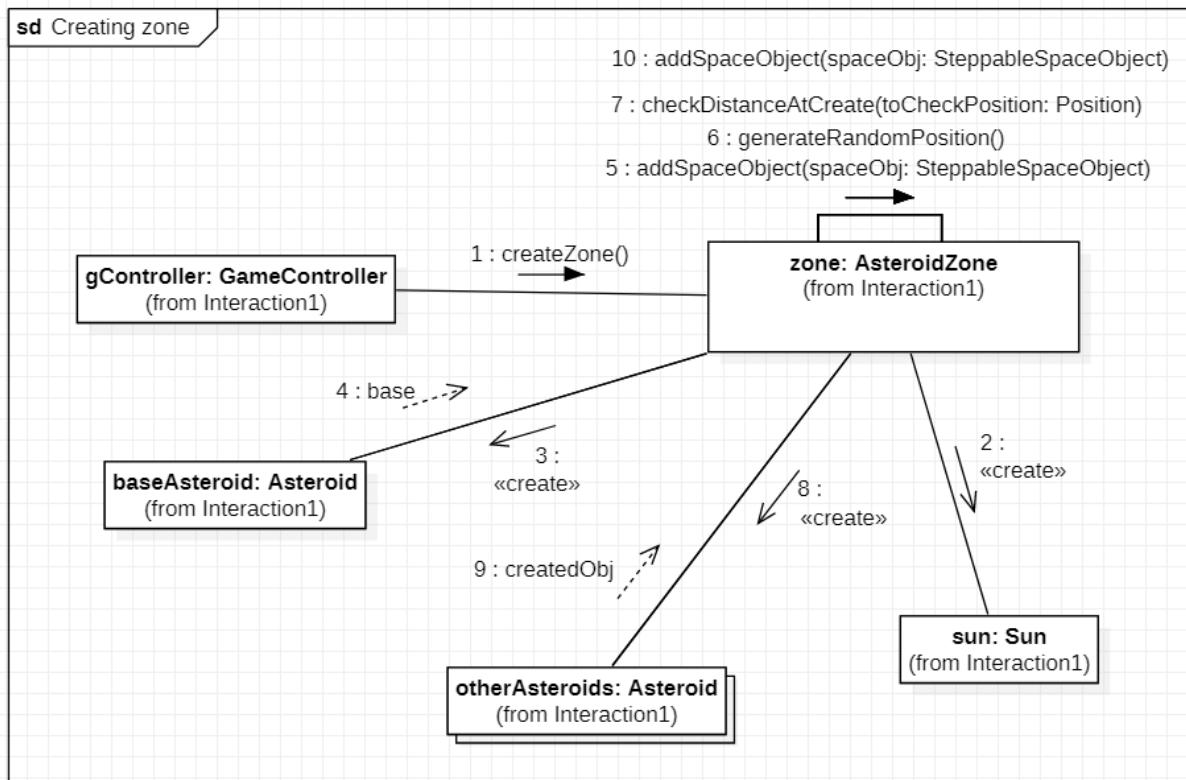
5.4.6 Game starts



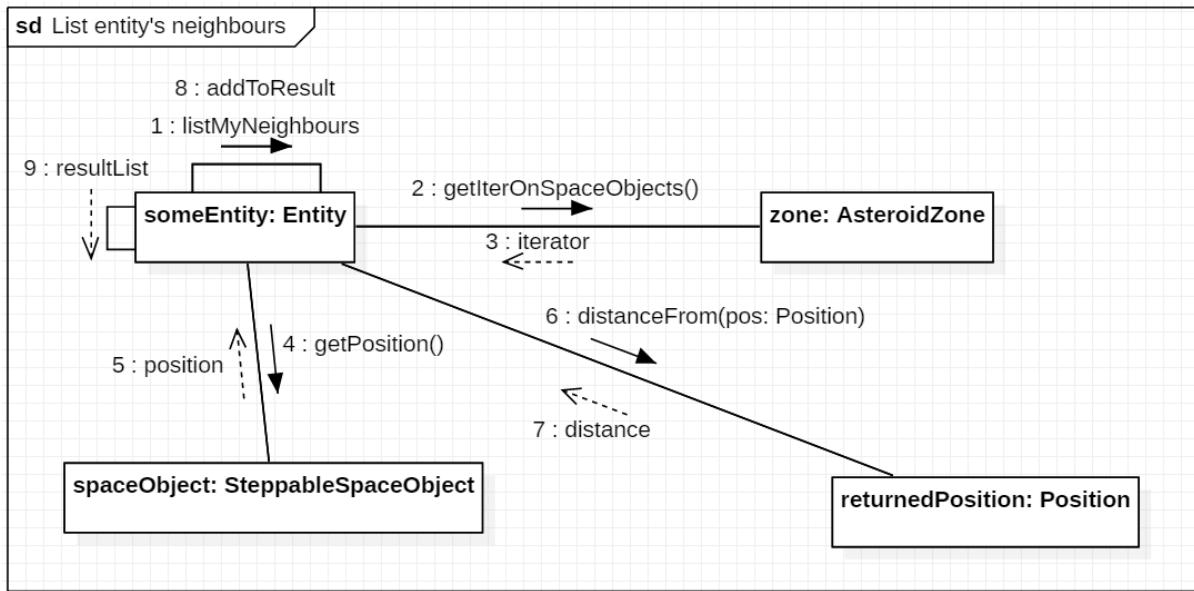
5.4.7 Settler creates gate



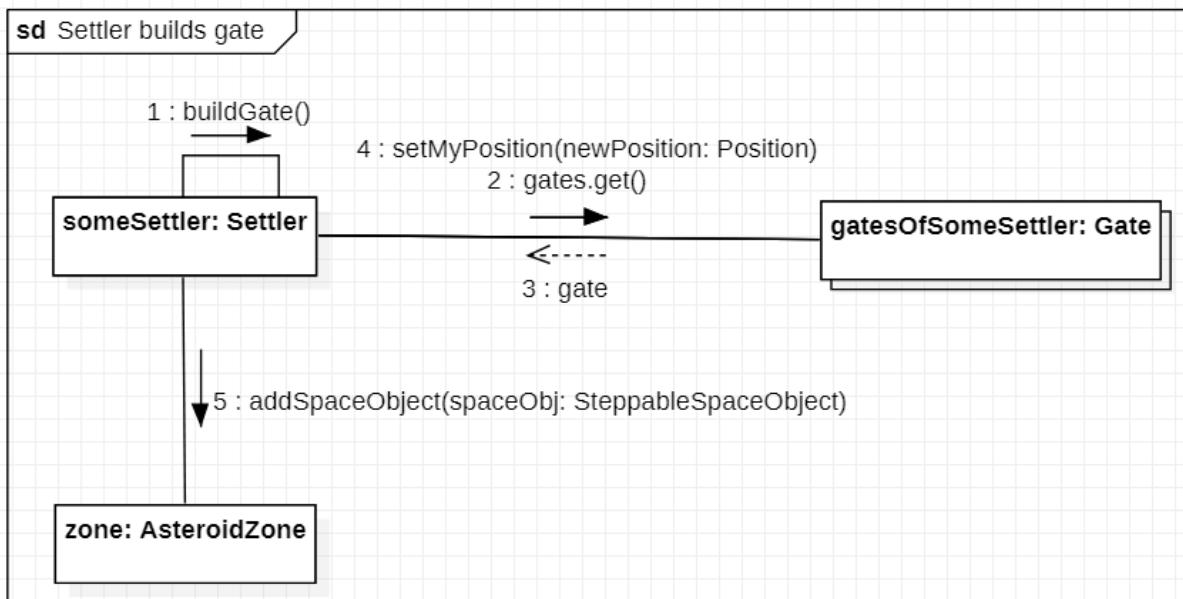
5.4.8 Creating asteroid belt



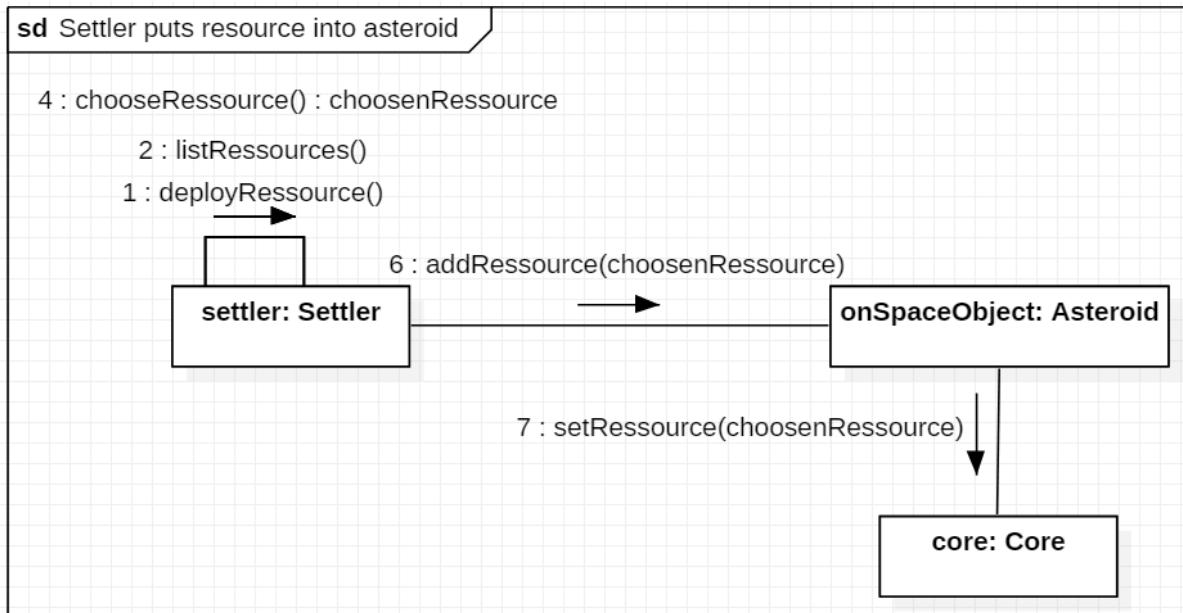
5.4.9 List entity's neighbours



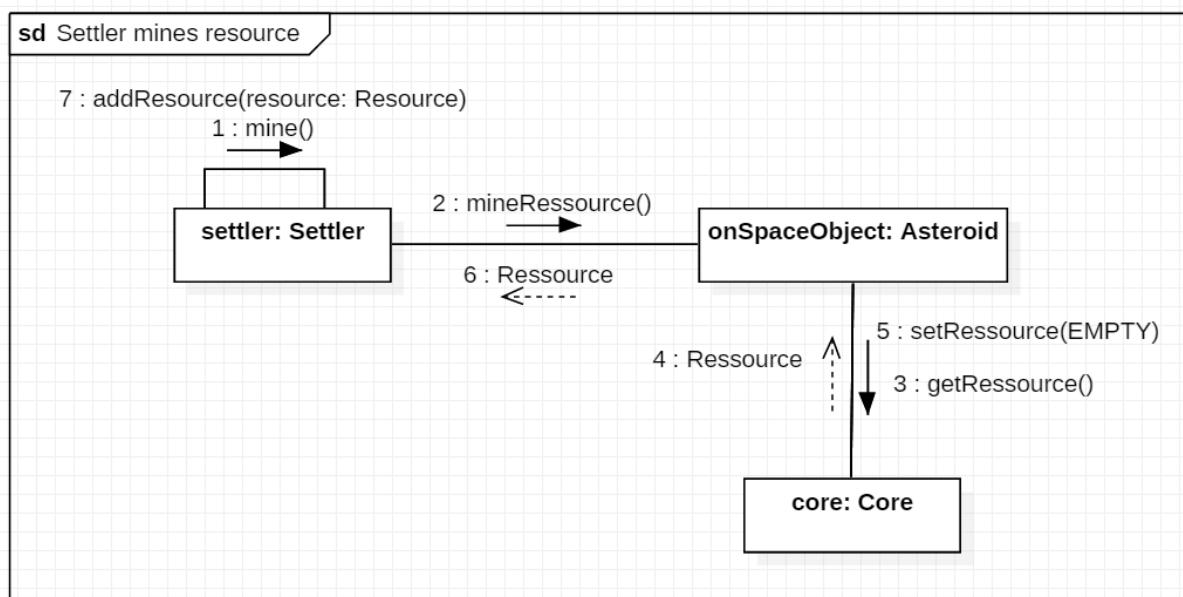
5.4.10 Settler places gate

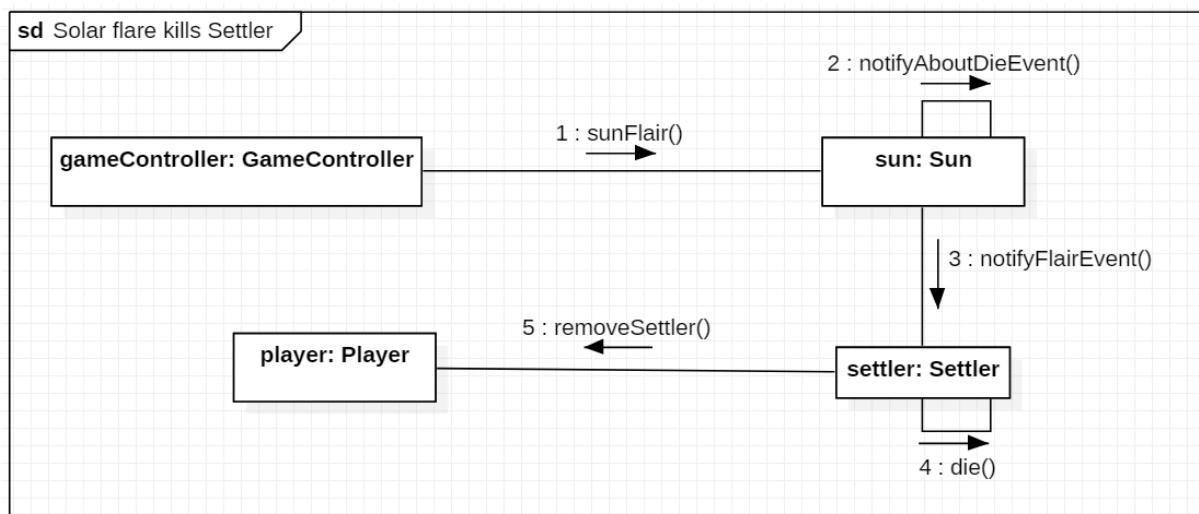


5.4.11 Settler puts resource into asteroid



5.4.12 Settler mines resource



5.4.13 Solar flare kills settler

5.5 Tagebuch

Anfang	Dauer	Teilnehmer	Beschreibung
2021.03.01	2 Stunde	Hrotkó	Nachdenken wie die Skeleton Oberfläche realisiert wird und danach die Planung wie die Tätigkeiten durchgeführt werden
2021.03.01	4 Stunde	Pongrácz	Anfang der Skeleton
2021.03.04	2 Stunde	Hrotkó	AsteroidZone Implementation und Dokumentation
2021.03.06	2 Stunde	Pongrácz	GameController Realisation
2021.03.06	1,5 Stunde	Hrotkó	Entity Klasse Implementation und Dokumentation
2021.03.07.	3 Stunde	Borbély Domokos Hrotkó Pongrácz	CodeTogether - Team Programmierung (zusammen)
2021.03.08	2 Stunde	Pongrácz	AIRObot implementation
2021.03.08	5 Stunde	Hrotkó	Settler Implementation und Dokumentation, Korrekturen in allen Funktionen bezüglich drill Ereignis
2021.03.08	1,5 Stunde	Borbély	Kommunikations-diagramme
2021.03.09	1,5 Stunde	Pongrácz	Kode testen, und Fehlerkorrekturen
2021.03.09	1 Stunde	alle	Besprechung über Vorlauf, Ergebnisse, Stand
2021.03.08.	2 Stunde	Domokos	Use-Case
2021.03.09	2 Stunde	Borbély	Kommunikations-diagramme

6. Skeleton Eingabe

6.1 Übersetzungs- und Ausführungsanleitung

6.1.1 Dateiliste

Dateinamen	Größe(Byte)	Ursprungsdatum	Inhalt
Player.java	2 459	2021.02.27. 16:01	Player class
GameController.java	15 468	2021.02.27. 16:01	GameController class
AsteroidZone.java	6 257	2021.02.27. 16:01	AsteroidZone class
AppController.java	4 272	2021.02.27. 16:01	AppController class
ConsoleUI.java	2 952	2021.02.27. 16:01	ConsoleUI
Asteroid.java	5 725	2021.02.27. 16:01	Asteroid class
Gate.java	1 655	2021.02.27. 16:01	Gate class
HomeAsteroid.java	1 169	2021.02.27. 16:01	HomeAsteroid class
Position.java	6 190	2021.02.27. 16:01	Position class
SteppableSpaceObject.java	3 841	2021.02.27. 16:01	SteppableSpaceObject class
Sun.java	3 833	2021.02.27. 16:01	Sun class
Core.java	1 782	2021.02.27. 16:01	Core class
Layer.java	1 145	2021.02.27. 16:01	Layer class
Coal.java	535	2021.02.27. 16:01	Coal class
Empty.java	583	2021.02.27. 16:01	Empty class
FrozenWater.java	570	2021.02.27. 16:01	FrozenWater class
Iron.java	534	2021.02.27. 16:01	Iron class
Resource.java	1 022	2021.02.27. 16:01	Resource class
Uran.java	578	2021.02.27. 16:01	Uran class
EventObservable.java	364	2021.02.27. 16:01	EventObservable interface
Observable.java	551	2021.02.27. 16:01	Observable interface
Observer.java	550	2021.02.27. 16:01	Observer interface
AIRobot.java	4 503	2021.02.27. 16:01	AIRobot class
Entity.java	5 231	2021.02.27. 16:01	Entity class
Settler.java	14 003	2021.02.27. 16:01	Settler class
RessourceStorage.java	3 323	2021.02.27. 16:01	RessourceStorage class
CallStackViewer.java	1 821	2021.02.27. 16:01	CallStackViewer class

6.1.2 Übersetzung

Mit Gradle

6.1.3 Programm lauf

Man muss im .../ideaProject/ Verzeichnis ein Kommandline öffnen, und die folgende Kommand ausgeben:

`./gradlew run`

Bevor diese Kommand muss man Gradle installieren: <https://gradle.org/install/>
Hoffentlich wird das Program laufen, falls Problemen auftreten, bitte unsere Team fragen und suchen.

6.2 Auswertung

Mitgliedsname	Mitglieds Neptun	Prozentsatz der Arbeit
Borbély Ábel	DRP0DT	25%
Domonkos Henrietta	J0DCPT	25%
Hrotkó Renátó	OIT6HD	25%
Pongrácz Vince	MKMDJO	25%

6.3 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2021.03.12	1 Stunde	Borbély Domokos Hrotkó Pongrácz	Aufgabeaufteilung
2021.03.12	2 Stunde	Hrotkó	ResourceStorage Klasse wurde in dem Kode eingeführt und andere Bug fixes
2021.03.14	2 Stunde	Domokos	kleine setup für test
2021.03.15	2 Stunde	Domokos	kleine setup für test
2021.03.16.	1 Stunde	Domokos	Dokumentum schreiben
2021.03.16	3 Stunde	Pongrácz	CallStack Logger implementieren
2021.03.15	2 Stunde	Pongrácz	Kleinere Fehlerkorrekturen um Exception zu vermeiden, CallStack Logger
2021.03.15	1,5 Stunde	Hrotkó	Drill Problem gelöst und andere kleine Korrekturen
2021.03.13.	2 Stunde	Pongrácz	Spezielle Kollektion für Ressourcen und diese Kollektion einbetten
2021.03.16	1,5 Stunde	Hrotkó	DeployResource Problem gelöst und andere kleine Korrekturen
2021.03.15	3 Stunde	Borbély	Gate Klasse funktionsfähig machen. Kleine Änderungen am Settler.
2021.03.16	1 Stunde	Borbély Domokos Hrotkó Pongrácz	Besprechung über Presentation

6. Skeleton Eingabe

6.1 Übersetzungs- und Ausführungsanleitung

6.1.1 Dateiliste

Dateinamen	Größe(Byte)	Ursprungsdatum	Inhalt
Player.java	2 459	2021.02.27. 16:01	Player class
GameController.java	15 468	2021.02.27. 16:01	GameController class
AsteroidZone.java	6 257	2021.02.27. 16:01	AsteroidZone class
AppController.java	4 272	2021.02.27. 16:01	AppController class
ConsoleUI.java	2 952	2021.02.27. 16:01	ConsoleUI
Asteroid.java	5 725	2021.02.27. 16:01	Asteroid class
Gate.java	1 655	2021.02.27. 16:01	Gate class
HomeAsteroid.java	1 169	2021.02.27. 16:01	HomeAsteroid class
Position.java	6 190	2021.02.27. 16:01	Position class
SteppableSpaceObject.java	3 841	2021.02.27. 16:01	SteppableSpaceObject class
Sun.java	3 833	2021.02.27. 16:01	Sun class
Core.java	1 782	2021.02.27. 16:01	Core class
Layer.java	1 145	2021.02.27. 16:01	Layer class
Coal.java	535	2021.02.27. 16:01	Coal class
Empty.java	583	2021.02.27. 16:01	Empty class
FrozenWater.java	570	2021.02.27. 16:01	FrozenWater class
Iron.java	534	2021.02.27. 16:01	Iron class
Resource.java	1 022	2021.02.27. 16:01	Resource class
Uran.java	578	2021.02.27. 16:01	Uran class
EventObservable.java	364	2021.02.27. 16:01	EventObservable interface
Observable.java	551	2021.02.27. 16:01	Observable interface
Observer.java	550	2021.02.27. 16:01	Observer interface
AIRobot.java	4 503	2021.02.27. 16:01	AIRobot class
Entity.java	5 231	2021.02.27. 16:01	Entity class
Settler.java	14 003	2021.02.27. 16:01	Settler class
RessourceStorage.java	3 323	2021.02.27. 16:01	RessourceStorage class
CallStackViewer.java	1 821	2021.02.27. 16:01	CallStackViewer class

6.1.2 Übersetzung

Mit Gradle

6.1.3 Programmlauf

Man muss im .../ideaProject/ Verzeichnis ein Kommandline öffnen, und die folgende Kommand ausgeben:

`./gradlew run`

Bevor diese Kommand muss man Gradle installieren: <https://gradle.org/install/>
Hoffentlich wird das Program laufen, falls Problemen auftreten, bitte unsere Team fragen und suchen.

6.2 Auswertung

Mitgliedsname	Mitglieds Neptun	Prozentsatz der Arbeit
Borbély Ábel	DRP0DT	25%
Domonkos Henrietta	J0DCPT	25%
Hrotkó Renátó	OIT6HD	25%
Pongrácz Vince	MKMDJO	25%

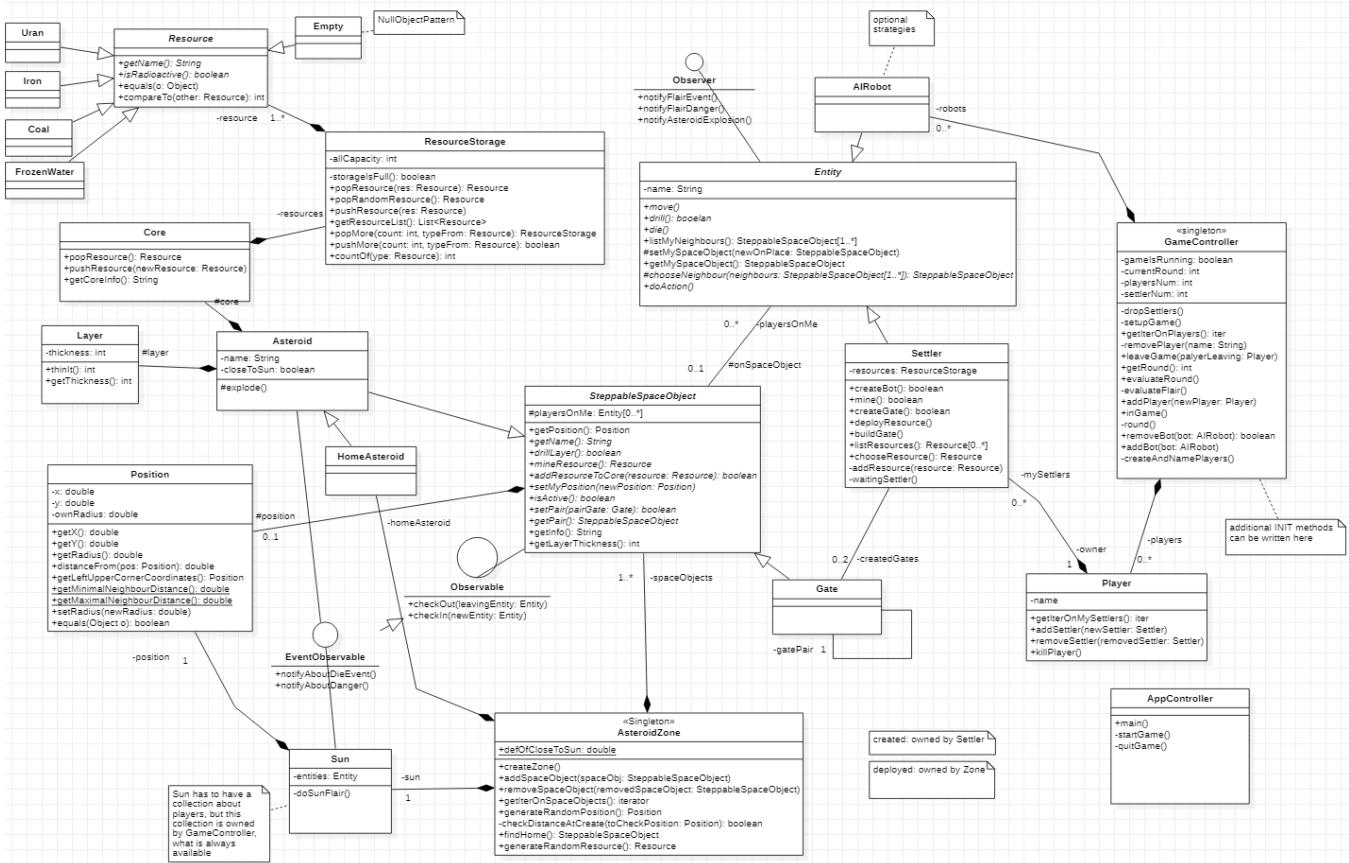
6.3 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2021.03.12	1 Stunde	Borbély Domokos Hrotkó Pongrácz	Aufgabeaufteilung
2021.03.14	2 Stunde	Domokos	kleine setup für test
2021.03.15	2 Stunde	Domokos	kleine setup für test
2021.03.16.	1 Stunde	Domokos	Dokumentum schreiben

7. Konzept von Prototyp

7.0 Wirkung der Änderung auf den Modell

7.0.1 Änderungen in Klassendiagramm



7.0.2 Neue oder veränderte Methoden

7.0.2.1 Ressource

Die Klasse ist schon abstrakt nicht ein interface.

Methoden:

- `compareTo(Resource o)`: Vergleicht 2 Resource ob sie die Gleiche sind oder die Name der Ressourcen in alphabetische Reihenfolge kleiner oder größer ist.
- `equals()`: Bestimmt ob 2 Ressourcen die Gleiche sind oder nicht

7.0.2.2 Core

Methoden:

- *popResource(): nimmt ein Ressource aus dem Stack der Ressourcen*
- *pushResource(resource res): nimmt ein Resource in den Stack rein*

7.0.2.3 ResourceStorage

Eine Hilfsklasse für die Speicherung von Ressourcen

Attribute:

- *allCapacity: int - Anzahl der gespeicherten Ressourcen*

Methoden:

- *storageIsFull(): Ergibt ob der Storage schon voll ist oder nicht*
- *popResource(): nimmt ein Ressource aus dem Storage der Ressourcen*
- *pushResource(resource res): nimmt ein Resource in dem Storage Rein*
- *popMore(int count): nimmt mehrere Ressource aus dem Storage der Ressourcen*
- *pushMore(int count, resource res): nimmt mehr Resource in dem Storage Rein*
- *countOfType(resource res) : ergibt die Anzahl der Rohstoffe der angegebenen Rohstoff*
- *getResourceList(): gibt zurück die Liste der Rohstoffe*

7.0.2.4 HomeAsteroid

Vererbt sich von Asteroid nur die übergeschriebene Funktionen sind anders

7.0.2.5 AsteroidZone

Methoden:

- *genereateRandomResource(): Es gibt zurück eine zufällig gewählte Ressource für die einfüllung der Kern beim Generierung der Asteroiden*
- *findHome(): Es gibt zurück das SteppableSpaceObject was den Basis Asteroid speichert*

7.0.2.6 SteppableSpaceObject

Methoden:

- *getLayerThickness(): Es gibt zurück ein int was die Manteldicke des Mantels in dem Asteroide speichert*

7.0.2.7 Entity

Methoden:

- *doAction(): In diesem Methode wird der Spieler auswählen welche Tätigkeit er ausführen wird z.B. Move, Drill usw.*

7.0.2.8 Settler

Attribute:

- *ResourceStorage: resources - Die Rohstoffe sind schon in einem Hilfsklasse ResourceStorage speichert nicht in einem Liste*

Methoden:

- *waitingSettler: Es macht nix, nur eine leere Runde um weitergehen zu können.*

7.0.2.9 GameController

Methoden:

- *createAndNamePlayers(): es herstellt die Spieler mit dem eingegebenen Namen und herstellt sie.*

7.0.2.10 Position

Methoden

- *equals(): Kontrolliert ob 2 Positionen die Gleiche sind oder nicht.*

7.0.3 Sequenz Diagramme

Es gibt keine nennenswerten Änderungen an den Sequenzen.

7.1 Interface Definition von Prototyp

7.1.1 Die allg. Beschreibung der Interface

Die Schnittstelle empfängt nur Befehle von der Standardeingabe und druckt alle Ausgaben an die Standardausgabe. Auf diese Weise kann es von einem Terminal aus verwendet und mithilfe des zu erstellenden Testdienstprogramms in Eingabedateien umgeleitet werden. Dies ermöglicht automatische Tests mit vorgefertigten Testfällen. Die Testfälle stammen aus der Reihe von Befehlen, die an den Prototyp gegeben werden sollen, sowie aus der korrekten Ausgabe, die für diese Reihe gegeben werden soll. Tests sind erfolgreich, wenn die tatsächliche und die beschriebene erwartete Ausgabe gleich sind.

7.1.2 Eingabesprache

jsonConfigTemplate:

Es ist eigentlich ein Testfall. Zeigt, welche Befehle ein Test mit welchen Parametern ausführen soll. Dementsprechend können Sie Testfälle manuell durchführen.

```
{
  "testname": "defaultTestName --- this test has default values, and don't do anything",
  "playerNumber": 1,
  "playerNames": ["testName"],
  "settlerNumber": 1,
  "asteroidNumber": 23,
  "range": 1000,
  "maxRound": 22,
  "sunFlairInEveryXRound": 10,
  "homeCapacity": 5,
  "settlerCapacity": 10,
  "defOfCloseToSun": 500,
  "maxNeighbourDistance": 500,
  "realCommandQueue": []
}
```

Die vom Prototyp akzeptierten Befehle sind:

start

Beschreibung: Starte das Spiel.

Optionen: -

move

Beschreibung: Schritt mit einem Siedler.

Optionen: Die Orte zu gehen.

drill

Beschreibung: Reduktion der Kortikalis des aktuellen Asteroiden.

Optionen: -

mine

Beschreibung: Rohstoffgewinnung aus einem Asteroiden.

Optionen: -

create gate

Beschreibung: Ein Teleport Tor machen.

Optionen: -

build gate

Beschreibung: Der Spieler legt das Teleporttor ab.

Optionen: -

create robot

Beschreibung: Ein AIRobot machen.

Optionen: -

list neighbours

Beschreibung: Liste benachbarter Asteroiden

Optionen: -

7.1.3 Ausgangssprache

Im Programm generierte Logos (auf Englisch).

7.2 Alle detaillierte Use-Case

Name der Use-Käse	<i>move</i>
Kurze Beschreibung	Schritt mit einem Siedler.
Aktoren	Tester
Drehbuch	Es bewegt sich zu einem der benachbarten Asteroiden.

Name der Use-Käse	<i>drill</i>
Kurze Beschreibung	Reduktion der Kortikalis des aktuellen Asteroiden.
Aktoren	Tester
Drehbuch	Es reduziert der Mantel des Asteroiden um eine Einheit.

Name der Use-Käse	<i>mine</i>
Kurze Beschreibung	Rohstoffgewinnung aus einem Asteroiden.
Aktoren	Tester
Drehbuch	Es können verschiedene Rohstoffe abgebaut werden. Wenn Sie sich auf einem radioaktiven Asteroiden in der Nähe befinden, explodiert der Siedler

Name der Use-Käse	<i>create gate</i>
Kurze Beschreibung	Ein Teleport Tor machen.
Aktoren	Tester
Drehbuch	Wenn es genug Rohmaterial gibt, um es herzustellen, kann es hergestellt werden, wenn nicht, dann nicht.

Name der Use-Käse	<i>build gate</i>
Kurze Beschreibung	Der Spieler legt das Teleporttor ab.
Aktoren	Tester
Drehbuch	Wenn es ein fertiges Teleport-Tor gibt, kann es abgelegt werden, wenn nicht, dann nein.

Name der Use-Käse	<i>create robot</i>
Kurze Beschreibung	Ein AIRobot machen.
Aktoren	Tester
Drehbuch	Wenn es genug Rohmaterial gibt, um es herzustellen, kann es hergestellt werden, wenn nicht, dann nicht.

Name der Use-Käse	<i>list neighbours</i>
Kurze Beschreibung	Liste der benachbarten Asteroiden
Aktoren	Tester
Drehbuch	Listet benachbarte Asteroiden auf

7.3 Testplan

Testen ist jetzt manuell, also man muss die Testkonfiguration angeben, und am Ende das Ergebnis manuell auswerten.

Um eine Testfall laufen lassen muss man im Testmodus eintreten, und dann die vordefinierte Testbeschreibungsdatei für Programm angeben. (z.B: testconfigs/createRobotTest.json)

Test Käse:	createRobotTest
Kurze Beschreibung:	Diese Testfall beweist, dass ein Settler kann ein Robot konstruieren.
Eingabedatei:	testconfigs/createRobotTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also ein Robot hergestellt wurde, aber die Zeitstempel sind anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	createBot
Testziel:	Eine Robot herstellen (Settler herstellt ein AIRobot)

```

round: 1 - 2021-03-23 17:12:43 [main] TRACE callStack
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
5
    createBot() called (Settler) - 2021-03-23 17:12:43 [main] TRACE callStack
        AIRobot constructor called - 2021-03-23 17:12:43 [main] TRACE callStack
        doAction() called (AIRobot) - 2021-03-23 17:12:43 [main] TRACE callStack
            move() called (Entity) - 2021-03-23 17:12:43 [main] TRACE callStack
                listMyNeighbours() called (Entity) - 2021-03-23 17:12:43 [main] TRACE callStack
name: temp0, layer: 4, core: Uran, isCloseToSun: true, position: x=279.0 y=391.0
        chooseNeighbour() called (AIRobot) - 2021-03-23 17:12:43 [main] TRACE callStack
        checkOut() called - 2021-03-23 17:12:43 [main] TRACE callStack
        checkIn() called - 2021-03-23 17:12:43 [main] TRACE callStack
        evaluateRound() called - 2021-03-23 17:12:43 [main] TRACE callStack
        evaluateFlair() called - 2021-03-23 17:12:43 [main] TRACE callStack
        listResources() called (Settler) - 2021-03-23 17:12:43 [main] TRACE callStack
Coal
Uran
FrozenWater
Iron
Iron

```

Test Käse:	createZoneTest
Kurze Beschreibung:	Diese Testfall konstruiert ein AsteroidZone, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll.
Eingabedatei:	testconfigs/createZoneTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also das AsteroidZone ist fertig, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	createZone
Testziel:	Eine AsteroidZone herstellen nach Konfiguration.

```

testName
Type the number of desired settlers for each player
-----
1
    createZone() called - 2021-03-23 17:15:11 [main] TRACE callStack
        Sun constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
        Asteroid constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
        HomeAsteroid constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
        Asteroid constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
        Asteroid constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
    dropSettlers() called - 2021-03-23 17:15:11 [main] TRACE callStack
        Settler constructor called - 2021-03-23 17:15:11 [main] TRACE callStack
        checkIn() called - 2021-03-23 17:15:11 [main] TRACE callStack
        checkIn() called (Sun) - 2021-03-23 17:15:11 [main] TRACE callStack

Setup ended
    inGame() - 2021-03-23 17:15:11 [main] TRACE callStack

```

Test Käse:	explodeTest
Kurze Beschreibung:	Testen einen Asteroidexplosion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Asteroid völlig bohren, und das Asteroid soll im Sonnennähe mit radioaktive Material im Kern sein.
Eingabedatei:	testconfigs/explodeTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Settler ist gestorben, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	explode
Testziel:	Eine AsteroidExplosion herstellen, und durchführen.

```

Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
1
    drill() called (Settler) - 2021-03-23 17:07:15 [main] TRACE callStack
        drillLayer() called (Asteroid) - 2021-03-23 17:07:15 [main] TRACE callStack
            thinIt() called (Layer) - 2021-03-23 17:07:15 [main] TRACE callStack
                The layer thickness is after the drilling: 0 - 2021-03-23 17:07:15 [main] TRACE callStack
            popResource() called (Core) - 2021-03-23 17:07:15 [main] TRACE callStack
            popRandomResource (ResourceStorage) called - 2021-03-23 17:07:15 [main] TRACE callStack
                The resource was: Uran - 2021-03-23 17:07:15 [main] TRACE callStack
            pushResource() called (Core) - 2021-03-23 17:07:15 [main] TRACE callStack
            notifyAboutDieEvent() called (Asteroid) - 2021-03-23 17:07:15 [main] TRACE callStack
                explode() called (Asteroid) - 2021-03-23 17:07:15 [main] TRACE callStack
                    notifyAsteroidExplosion() called (Settler) - 2021-03-23 17:07:15 [main] TRACE callStack
                    die() called (Settler) - 2021-03-23 17:07:15 [main] TRACE callStack
                        checkOut() called - 2021-03-23 17:07:15 [main] TRACE callStack
                        killPlayer() called - 2021-03-23 17:07:15 [main] TRACE callStack
                        checkOut() called (Sun) - 2021-03-23 17:07:15 [main] TRACE callStack
                        removeSpaceObject() called - 2021-03-23 17:07:15 [main] TRACE callStack
                    evaluateRound() called - 2021-03-23 17:07:15 [main] TRACE callStack
                    evaluateFlair() called - 2021-03-23 17:07:15 [main] TRACE callStack
Game lost

```

Test Käse:	listNeighboursTest
Kurze Beschreibung:	Testen das Auszahlfunktion, in dem wir alle benachbarte Asteroide durchschauen können. Falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll.
Eingabedatei:	testconfigs/listNeighboursTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also manche Nachbarn ausgelistet sind, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	listMyNeighbours
Testziel:	Nachbarnasteroiden auflisten.

```

inGame() - 2021-03-23 17:21:52 [main] TRACE callStack
    round: 1 - 2021-03-23 17:21:52 [main] TRACE callStack
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
7
        listMyNeighbours() called (Entity) - 2021-03-23 17:21:52 [main] TRACE callStack
name: temp0, layer: 4, core: Uran, isCloseToSun: true, position: x=279.0 y=391.0
name: templ, layer: 3, core: Iron, isCloseToSun: true, position: x=512.0 y=485.0
name: temp5, layer: 4, core: Empty, isCloseToSun: true, position: x=172.0 y=80.0
name: temp3, layer: 5, core: Coal, isCloseToSun: true, position: x=183.0 y=109.0
name: temp6, layer: 4, core: FrozenWater, isCloseToSun: false, position: x=491.0 y=710.0
        evaluateRound() called - 2021-03-23 17:21:52 [main] TRACE callStack
        evaluateFlair() called - 2021-03-23 17:21:52 [main] TRACE callStack
        listResources() called (Settler) - 2021-03-23 17:21:52 [main] TRACE callStack
Coal
Coal
Uran
Uran
FrozenWater
Iron
Iron
Iron

```

Test Käse:	sunFlairTest
Kurze Beschreibung:	Testen einen SunFlair Ereignis, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Im Test wird das Settler auf einem benachbarten Asteroide bewegen, dort auf dem SunFlair warten, und endlich gestorben.
Eingabedatei:	testconfigs/sunFlairTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Settler ist gestorben auf einem anderen Asteroid, das sie/er nicht geschützt hat. Die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	SunFlair
Testziel:	Auf eine Solarflair warten, und wegen der sterben.

```

temp18
temp3
temp8
temp14
temp21
-----
temp1
    checkOut() called - 2021-03-23 17:26:36 [main] TRACE callStack
    checkIn() called - 2021-03-23 17:26:36 [main] TRACE callStack
    evaluateRound() called - 2021-03-23 17:26:36 [main] TRACE callStack
    evaluateFlair() called - 2021-03-23 17:26:36 [main] TRACE callStack
    round: 2 - 2021-03-23 17:26:36 [main] TRACE callStack
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
8
    waitingSettler() called (Settler) - 2021-03-23 17:26:36 [main] TRACE callStack
    evaluateRound() called - 2021-03-23 17:26:36 [main] TRACE callStack
    evaluateFlair() called - 2021-03-23 17:26:36 [main] TRACE callStack
    notifyAboutDieEvent() called (Sun) - 2021-03-23 17:26:36 [main] TRACE callStack
    doSunFlair() called (Sun) - 2021-03-23 17:26:36 [main] TRACE callStack
    notifyFlairEvent() called (Settler) - 2021-03-23 17:26:36 [main] TRACE callStack
    die() called (Settler) - 2021-03-23 17:26:36 [main] TRACE callStack
    checkOut() called - 2021-03-23 17:26:36 [main] TRACE callStack
    killPlayer() called - 2021-03-23 17:26:36 [main] TRACE callStack
    checkOut() called (Sun) - 2021-03-23 17:26:36 [main] TRACE callStack
    killPlayer() called - 2021-03-23 17:26:36 [main] TRACE callStack
Game lost

```

Test Käse:	sunFlairTest2
Kurze Beschreibung:	Testen einen SunFlair, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler nichts machen, weil die auf dem Home automatisch geschützt ist, also er/sie muss nur warten.
Eingabedatei:	testconfigs/sunFlairTest2.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Settler noch lebend, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	SunFlair
Testziel:	Auf eine Solarflair warten, und das Flairereignis überleben.

```

Setup ended
    inGame() - 2021-03-23 17:28:08 [main] TRACE callStack
        round: 1 - 2021-03-23 17:28:08 [main] TRACE callStack
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
8
    waitingSettler() called (Settler) - 2021-03-23 17:28:08 [main] TRACE callStack
    evaluateRound() called - 2021-03-23 17:28:08 [main] TRACE callStack
        evaluateFlair() called - 2021-03-23 17:28:08 [main] TRACE callStack
            notifyAboutDieEvent() called (Sun) - 2021-03-23 17:28:08 [main] TRACE callStack
                doSunFlair() called (Sun) - 2021-03-23 17:28:08 [main] TRACE callStack
                    notifyFlairEvent() called (Settler) - 2021-03-23 17:28:08 [main] TRACE callStack
                        mineResource() called (HomeAsteroid) - 2021-03-23 17:28:08 [main] TRACE callStack
                            popResource() called (Core) - 2021-03-23 17:28:08 [main] TRACE callStack
                                popRandomResource (ResourceStorage) called - 2021-03-23 17:28:08 [main] TRACE callStack
                                    The resource was: Empty - 2021-03-23 17:28:08 [main] TRACE callStack
                            listResources() called (Settler) - 2021-03-23 17:28:08 [main] TRACE callStack
Coal
Coal
Uran
Uran
FrozenWater
Iron
Iron
Iron

```

Test Käse:	drillTest
Kurze Beschreibung:	Testen das Drill Funktion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler bohren(drill).
Eingabedatei:	testconfigs/drillTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also das Asteroid ist gebohrt, ihre Mantelgröße ist kleiner (falls noch nicht durchgebohrt), aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	drill()
Testziel:	Eine Asteroid bohren.

```
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
        evaluateRound() called
        evaluateFlair() called
            notifyAboutDanger() called (Sun)
            notifyFlairDanger() called (Settler)
    round: 7
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
```

Test Käse:	mineTest
Kurze Beschreibung:	Testen das MineFunktion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler fördern(mine).
Eingabedatei:	testconfigs/mineTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also das Asteroid ist gebohrt und fördert, ihre Kern ist leer, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	mine()
Testziel:	Eine Asteroid fördern.

```

Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
2
    mine() called (Settler)
        mineResource() called (Asteroid)
        popResource() called (Core)
        popRandomResource (ResourceStorage) called
            The resource was: Coal
            addResource called
evaluateRound() called
    evaluateFlair() called
round: 8

```

```
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
        evaluateRound() called
        evaluateFlair() called
            notifyAboutDanger() called (Sun)
            notifyFlairDanger() called (Settler)
    round: 7
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
```

Test Käse:	deployResourceTest
Kurze Beschreibung:	Testen das deployResource, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(deployResource).
Eingabedatei:	testconfigs/deployResource.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also das Asteroid ist gebohrt und fördert, ihre

	Kern ist leer, und dann ist die geförderte Rohstoff zurückgesteckt, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	deployResource()
Testziel:	In eine leere Asteroide ein Rohstoff zurückstecken

```

Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
6
    deployResource() called (Settler)
        listResources() called (Settler)
Coal
Coal
Uran
Uran
FrozenWater
Iron
Iron
Iron
Coal
    chooseResource() called (Settler)
-----
1
    addResourceToCore() called (Asteroid)
        popResource() called (Core)
        popRandomResource (ResourceStorage) called
            The resource was: Empty
        pushResource() called (Core)

```

```
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
        evaluateRound() called
        evaluateFlair() called
            notifyAboutDanger() called (Sun)
            notifyFlairDanger() called (Settler)
    round: 7
Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
1
    drill() called (Settler)
        drillLayer() called (Asteroid)
            thinIt() called (Layer)
                The layer thickness is after the drilling: 0
            popResource() called (Core)
            popRandomResource (ResourceStorage) called
                The resource was: Coal
            pushResource() called (Core)
```

Test Käse:	mineTest
Kurze Beschreibung:	Testen das deployResource, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(deployResource).
Eingabedatei:	testconfigs/deployResource.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also das Asteroid ist gebohrt und fördert, ihre

	Kern ist leer, und dann ist die geförderte Rohstoff zurückgesteckt, aber die Zeitstempel sind natürlich anders. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	deployResource()
Testziel:	In eine leere Asteroide ein Rohstoff zurückstecken

```
-----
0
    move() called (Entity)
        listMyNeighbours() called (Entity)
name: temp20, layer: 5, core: Iron, isCloseToSun: true, position: x=1.0 y=355.0
name: temp16, layer: 4, core: Iron, isCloseToSun: false, position: x=752.0 y=668.0
name: temp17, layer: 5, core: FrozenWater, isCloseToSun: false, position: x=601.0 y=624.0
name: temp18, layer: 3, core: Uran, isCloseToSun: true, position: x=373.0 y=363.0
name: temp10, layer: 4, core: Uran, isCloseToSun: false, position: x=797.0 y=208.0
name: temp5, layer: 4, core: Empty, isCloseToSun: true, position: x=172.0 y=80.0
name: temp11, layer: 5, core: Iron, isCloseToSun: false, position: x=781.0 y=476.0
name: temp7, layer: 5, core: Iron, isCloseToSun: false, position: x=11.0 y=695.0
name: temp19, layer: 5, core: Coal, isCloseToSun: true, position: x=97.0 y=607.0
name: temp9, layer: 5, core: Empty, isCloseToSun: true, position: x=390.0 y=263.0
name: temp12, layer: 5, core: Uran, isCloseToSun: true, position: x=114.0 y=377.0
name: temp0, layer: 4, core: Uran, isCloseToSun: true, position: x=279.0 y=391.0
name: temp8, layer: 4, core: Iron, isCloseToSun: false, position: x=155.0 y=677.0
name: temp6, layer: 4, core: FrozenWater, isCloseToSun: false, position: x=491.0 y=710.0
name: temp22, layer: 4, core: FrozenWater, isCloseToSun: true, position: x=367.0 y=585.0
name: temp15, layer: 5, core: FrozenWater, isCloseToSun: true, position: x=630.0 y=201.0
name: temp3, layer: 5, core: Coal, isCloseToSun: true, position: x=183.0 y=109.0
name: temp14, layer: 5, core: Uran, isCloseToSun: false, position: x=645.0 y=884.0
name: temp1, layer: 3, core: Iron, isCloseToSun: true, position: x=512.0 y=485.0
name: temp21, layer: 5, core: Coal, isCloseToSun: true, position: x=271.0 y=349.0
chooseNeighbour() called (Settler)

Choose destination:
temp20
temp16
temp17
temp18
temp10
temp5
temp11
temp7
temp19
temp9
temp12
temp0
temp8
temp6
temp22
temp15
temp3
temp14
temp1
temp21
-----
temp11
    checkOut() called
    checkIn() called
    evaluateRound() called
        evaluateFlair() called
    round: 2
```

Test Käse:	moveTest
Kurze Beschreibung:	Testen das move, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler , um zu einem anderen Asteroiden zu ziehen
Eingabedatei:	testconfigs/moveTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Siedler befindet sich auf dem entsprechenden Asteroiden. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	move()
Testziel:	Ein Siedler, um zu einem anderen Asteroiden zu bewegen.

Test Käse:	createGateTest
Kurze Beschreibung:	Testen das createGate, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Teleportpaar aus seinen Ressourcen machen.
Eingabedatei:	testconfigs/createGateTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Siedler konnte die Teleporter erstellen. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	createGate()
Testziel:	Erstellen eines Teleporterpaars aus den erforderlichen Ressourcen.

```

Choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
3
    createGate() called (Settler)
        Gate constructor called
        Gate constructor called
        The 2 gates were added to the player
        setPair() called (Gate)
        setPair() called (Gate)
    evaluateRound() called
        evaluateFlair() called
round: 3

```

Test Käse:	buildGateTest
Kurze Beschreibung:	Testen das buildGate, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Teleportpaar in dem Asteroidenfeld platzieren.
Eingabedatei:	testconfigs/buildGateTest.json
Erwartete Ausgabe:	siehe unten
Testauswertung:	Test ist erfolgreich, falls die erwartete Ausgabe erscheint, also der Siedler konnte die Teleporter platzieren. Alle andere Fall ist unerfolgreich.
Dazugehörige Sequence:	buildGate()
Testziel:	Platzieren eines Teleporterpaars in dem Spielfeld.

```

choose an option:
0 : move
1 : drill
2 : mine
3 : create gate
4 : build gate
5 : create robot
6 : deploy resource
7 : list neighbours
8 : wait
-----
4
    buildGate() called (Settler)
    addSpaceObject() called

```

7.4 Hilfsprogramme beim Testen, und ihre Spezifikation

Zurzeit gibt's keine spezielle Programme für Testen ausserhalb von **java JVM** und **gradle** oder **IntelliJ IDEA**.

Aber wir haben von uns definierte Testkonfigurationsdateien, die die Testszenarien beschreiben. Diese Konfigurationsdateien sind im ideaProject/testconfigs/ Verzeichnis, und das Programm kann diesen einlesen, und laufen lassen. Leider die Auswertung ist noch nicht automatisch.

Template dieses Testdateien: (Bsp: jsonConfigTemplate_DO_NOT_OVERWRITE.json)

```
{
  "testname": "defaultTestName --- this test has default values, and don't do anything",
  "playerNumber": 1,
  "playerNames": ["testName"],
  "settlerNumber": 1,
  "asteroidNumber": 23,
  "range": 1000,
  "maxRound": 22,
  "sunFlairInEveryXRound": 10,
  "homeCapacity": 5,
  "settlerCapacity": 10,
  "defOfCloseToSun": 500,
  "maxNeighbourDistance": 500,
  "realCommandQueue": []
}
```

7.5 Tagebuch

Anfang	Dauer	Teilnehmer	Beschreibung
--------	-------	------------	--------------

2021.03.17.	1 Stunde	alle	Allg Gespräch über die Aufgaben
2021.03.17.	2 Stunde	Pongrácz	Fehlerbehebung (Concurrent Modification) und kleinere Korrektur im move().
2021.03.17	1 Stunde	Hrotkó	Seed einstellen für das Spiel und Modifikationen
2021.03.18	3-4 Stunde	Hrotkó	Asteroide generierung optimalisieren mit kleineren radius, und slipping realisieren und bug fixes
2021.03.21	5 Stunde	Pongrácz	Testumgebung herstellen, testconfig template herstellen, manche Testreporte schreiben
2021.03.23	2 Stunde	Alle	Tests schreiben, Dokument ergänzen und Gespräch über die Aufgaben
2021.03.22.			

- **Detaillierte Pläne**

8.1. Entwürfe von Klassen und Methoden

8.1.1 AIRobot

- **Verantwortung**

Der AIRobot bohrt, bewegt sich und hilft den Spielern, ihr Ziel zu erreichen.

- **Basisklasse**

Entity → AIRobot

8.1.2 AppController

- **Verantwortung**

Diese Abteilung ist für die Ausführung des Spiels verantwortlich. Startet, stoppt oder pausiert das Spiel.

- **Methoden**

- **+main()**: Ein Programm muss eine globale Funktion namens main enthalten, die den festgelegten Start des Programms darstellt.
- **-startGame()**: Wird aufgerufen, wenn das Spiel beginnt.
- **-quitGame()**: Wird aufgerufen, wenn das Spiel beendet ist.

8.1.3 Asteroid

- **Verantwortung**

Der Asteroid speichert Rohmaterial in seinem Kern. Er kann Astronauten schützen, wenn sie sich in seinem Kern verstecken. Das Asteroid kann explodieren. Das Rohmaterial des Kerns kann abgebaut werden..

- **Ősosztályok**

SteppableSpaceObject → Asteroid

- **Schnittstellen / Interface**

- EventObservable
- Observable

- **Attribute**

- **String name:** Der Name des Asteroiden.
- **boolean closeToSun:** Ist der Asteroid in der Nähe des Sonne.

- **Methoden**

- noch überschreibende Methoden von der abstrakten Basisklasse (SteppableSpaceObject)
- **explode():** innere Method um Explosion zu behandeln

8.1.4 HomeAsteroid

- **Verantwortung**

Diese Asteroid speichert gleichzeitig mehrere Rohmaterial in seinem Kern. Er kann Astronauten schützen, wenn sie sich in seinem Kern verstecken. Dieser Asteroid kann nicht explodieren, er hat andere Verhalten aus diese Sichtpunkt.

- **Basisklassen**

SteppableSpaceObject, Asteroid → HomeAsteroid

- **Schnittstellen / Interface**

- EventObservable
- Observable

- **Attribute**

- **String name:** Der Name des Asteroiden. (“Home”)
- **boolean closeToSun:** Ist der Asteroid in der Nähe des Sonne.

- **Methoden**

- noch überschreibende Methoden von der abstrakten Basisklasse (SteppableSpaceObject)
- **explode():** innere Method um Explosion zu behandeln.

8.1.5 AsteroidZone

- **Verantwortung**

Verwaltet das Spielfeld.

- **Attribute**

- **double defOfCloseToSun:** Wenn Sie diese Zahl mit einem SteppableSpaceObject vergleichen, können Sie feststellen, ob es sich in der Nähe der Sonne befindet

- **Methoden**

- **+createZone():** Es schafft das Spielfeld vor dem Spielstart.
- **+addSpaceObject(spaceObj: SteppableSpaceObject):** Fügt der Asteroidenzone einen SteppableSpaceObject hinzu.
- **+removeSpaceObject(onPosition: Position):** Entfernt ein Objekt in einer Asteroidenzone von einer bestimmten Position.
- **+iterator getIterOnSpaceObjects():** Wenn ich die Nachbarn auflisten möchte, gibt diese Funktion eine Liste aller SpaceObjects und ich kann basierend darauf weiter filtern.
- **+Position generateRandomPosition():** Erstellt Positionen durch Generieren von Zufallszahlen. Erforderlich, um die Strecke zu bauen.
- **-boolean checkDistanceAtCreate(toCheckPosition: Position):** Hier wird geprüft, ob um eine bestimmte Position herum genügend Platz vorhanden ist, um ein anderes SpaceObject abzulegen.
- **+SteppableSpaceObject findHome():** Das Haus kehrt mit einem Asteroiden zurück.
- **+Ressource generateRandomRessource():** Er kehrt mit einem zufälligen Rohstoff zurück.

8.1.6 Core

- **Verantwortung**

Es ist verantwortlich für die Speicherung der Rohstoffe innerhalb des Asteroiden

- **Attribute**

- **int capacity:** Gibt an, wie viele Rohstoffe sich im Kern befinden können.

- **Methoden**

- **Resource[1...*] popResource:** Es kehrt mit dem Rohmaterial des Kerns zurück.
- **pushResource(newResource: Resource):** Setzt den Kernrohstoff auf eine bestimmte Ressource
- **+getCoreInfo():** Gibt den Namen des Rohmaterials zurück.

8.1.7 Entity

- **Verantwortung**

Abstrakte Basisklasse für Roboter und Siedler.

- **Schnittstellen / Interface**

- Observer → Entity

- **Attribute**

- **-String name:** Name des Entity.

- **Methoden**

- **+move():** Durch Aufrufen dieser Funktion werden die Entitäten verschoben.
- **+drill():** Durch Aufrufen dieser Funktion werden die Entitäten bohren.
- **+die():** Durch Aufrufen dieser Funktion werden Entitäten beendet.
- **+SteppableSpaceObject[] listMyNeighbours():** Gibt die Nachbarn des Asteroiden zurück, auf dem sich die Entität befindet.
- **+SteppableSpaceObject getMySpaceObject():**
- **+doAction():**
- **#setMySpaceObject(newOnPlace: SteppableSpaceObject):**
- **#SteppableSpaceObject chooseNeighbour(neighbours: SteppableSpaceObject[]):**

8.1.8 EventObservable

- **Verantwortung**

Wenn ein Asteroid explodiert oder eine Sonneneruption gibt, benachrichtigt SteppableSpaceObject.

- **Methoden**

- **notifyAboutDanger():** Informiert alle aufgeschriebene Entität über die Gefahr von Solarflair.
- **notifyAboutDieEvent():** Informiert alle aufgeschriebene Entität über den Tod, und das Grund darauf.

8.1.9 GameController

- **Verantwortung**

Verwaltet den Verlauf des Spiels. Vom Start zum Ende.

- **Attribute**

- **-boolean gameIsRunning:** Sagt dir, ob das Spiel läuft.
- **-int currentRound:** Die Nummer der aktuellen Runde des Spiels.
- **+int playersNum:** Anzahl der Spieler. Etwas, das während des Setups konfiguriert werden kann.
- **-int settlerNum:** Anzahl der Siedler. Etwas, das während des Setups konfiguriert werden kann.

- **Methoden**

- **-removePlayer(name: String):** Spieler aus dem Spiel entfernen.
- **leaveGame(playerLeaving: Player):** Wenn ein Spieler aufhören möchte, gibt er auf, tötet im Wesentlichen alle seine Siedler und löscht sie auch aus den Spielern.
- **+int getRound():** Gibt das Anzahl der aktuelle Runde im Spiel zurück.
- **+evaluateRound():** Wertet den gegebenen Rund aus. Überprüfen Sie, ob alle Siedler des Spielers gestorben sind, und ob die Spieler gewonnen oder verloren haben.
- **+addPlayer(newPlayer: Player):** Fügt dem Spiel einen neuen Spieler hinzu.
- **+inGame():** Eine fast unendliche Schleife, in dieser Funktion läuft das Spiel und das Spiel kreist als Endlosschleife. Wir rufen diese Funktion auf, indem wir das Spiel starten und am Ende des Spiels beenden.
- **+boolean removeBot(bot: AIRobot):** Entfernt den Roboter. Der Boolesche Wert gibt an, ob Sie ihn entfernt haben.
- **+addBot(bot: AIRobot):** Fügt dem Spiel einen neuen AIRoboter hinzu.
- **-dropSettlers():** Macht einen Siedler für den Spieler.
- **-setUpGame():** Rufen Sie die Funktionen auf, mit denen das Spiel eingestellt wird.
- **+iter getterOnPlayer():** Kehrt mit den Spielern zurück, die den ganzen Weg iteriert wurden.
- **-removePlayer(name: String):** Entfernt einen bestimmten Spieler von den Spielern.
- **-evaluateFlair():** Bewertet, dass es diesen Sonnenwind im gegebenen Kreis gibt.
- **-round():** Diese Funktion geht abwechselnd durch die Spieler und Siedler, wenn sie etwas tun.
- **-createAndNamePlayers():** Legt den Namen des Spielers fest.

4.3.9 Gate

- **Verantwortung**

Kann von Siedlern erstellt und gespeichert werden. Ermöglicht das Reisen zwischen einem Paar von ihnen.

- **Basisklasse**

SteppableSpaceObject → Gate

- **Attribute**

- **Gate gatePair:** Der Paar eines Portals.

- **Methoden**

nur überschreibende Methoden von der abstrakten Basisklasse.

8.1.10 Layer

- **Verantwortung**

Es stellt die Kortikalis/Kruste/Layer des Asteroiden dar, weiß, wie dick er ist und kann ihn reduzieren.

- **Attribute**

- **int thickness:** Die Dicke der Kruste des Asteroiden.

- **Methoden**

- **int thinIt():** Es verdünnt sich. Der DrillLayer ruft Sie an.
- **int getThickness():** Der Asteroid kehrt mit der Dicke seiner Kortikalis zurück.

8.1.11 Player

- **Verantwortung**

Hilft bei der Verwaltung der Entitäten eines Spielers, repräsentiert ein Spieler. (nicht ein Astronaut/Settler)

- **Attribute**

- **-String name:** Der Name des Spielers.
- **-Settler[1..n] mySettlers:** Die Kollektion der Siedler, die zu einem Spieler gehören.

- **Methoden**

- **+Iterator getIterOnMyEntities():** Erzeugt einen Iterator für die Kollektion der Siedler eines Spielers
- **+addEntity(newEntity: Entity):** Weist einem Spieler eine neu Entität zu.
- **+removeEntity(removedEntity: Entity):** Hebt die Zuordnung einer Entität zu einem Spieler auf
- **+setName(name: String):** Setter für Name.
- **+String getName():** Getter für Name.
- **+killPlayer():** Tötet/eliminiert ein Spieler mit ihre Settlern aus dem Spiel.

4.3.12 Position

- **Verantwortung**

Ordnet Objekten im Feld eine bestimmte Position zu.

- **Schnittstellen / Interface**

Comparable

- **Attribute**

- **double x:** Koordinate auf der x-Achse.
- **double y:** Koordinate auf der y-Achse
- **double ownRadius:** Hilft überlappende Objekte zu vermeiden.

- **Methoden**

- **+double getX():** Gibt die x-Koordinate zurück.
- **+double getY():** Gibt die y-Koordinate zurück.
- **+double getRadius():** Gibt den Radius des zugehörigen Objekts zurück.
- **+double distanceFrom(pos: Position):** Kalkuliert die Distanz zweier Positionen.
- **+double[2] getLeftUpperCornerCoordinate():** Hilft mit dem korrekten Platzieren der Objekte.
- **+double getMinimalNeighbourDistance():** Untere Schranke für die Zufallsdistanz, in der die Objekte benachbart sind.
- **+double getMaximalNeighbourDistance():** Obere Schranke...
- **+setRadius(newRadius: double):** Legt den Radius der Position fest.
- **+boolean equals(Object o):** Sagt, dass die Positionen der beiden Objekte gleich sind.

8.1.13 Settler

- **Verantwortung**

Siedler sind die Entitäten, die die Spieler bewegen und über sie mit dem Spielfeld interagieren können.

- **Basisklasse**

Entity → Settler

- **Attribute**

- **Resources[0...10] resources:** Speichert die abgebauten Ressourcen eines Siedlers.
- **Player owner:** Der Spieler, dem die Siedler gehören.
- **Gate[0...2] createdGates:** Die Kollektion der erstellten Portale.

- **Methoden**

- **+createBot():** Erschafft einen AI Roboter.
- **+mine():** Entfernt die Ressource aus dem Asteroidenkern, und fügt es seiner eigenen Kollektion von Ressourcen zu.
- **+createGate():** Erstellt ein Paar Portale, und fügt sie seiner eigenen Kollektion von Portalen zu.
- **+deployResource():** Setzt eine Ressource wieder in einen Asteroiden ein.
- **+buildGate():** Platziert eines der getragenen Portale.
- **+listResources():** Listet die Ressourcen auf, die der Siedler besitzt.
- **+Resource chooseResource():** Der Siedler wählt eine Ressource aus seinem Inventar aus.
- **-addResource(resource: Resource):** Fügt Ressource zu der Kollektion .
- **-waitingSettler():** Der Siedler tut nichts und wartet auf die nächste Runde.

8.1.14 SteppableSpaceObject

- **Verantwortung**

Diese sind Objekte, auf die die Spieler mit ihren Siedlern treten können.

- **Schnittstellen / Interface**

Observable

- **Attribute**

- **Entity[0...*] playersOnMe:** Speichert die Entitäten, die auf einem solchen Objekt stehen.

- **Methoden**

- **Position getPosition():** Gibt die Position des Objektes zurück.
- **String getName():** Gibt den Namen des SteppableSpaceObjectes zurück, falls es existiert.
- **boolean drillLayer():** Falls es eine Asteroid ist, reduziert die Größe der Kortikalis des Asteroiden um eins. Übergibt die Bohraufgabe an das Layer.
- **Ressource mineResource():** Falls es eine Asteroid ist, baut es das Innere des Asteroiden ab und setzt die Art des Rohmaterials auf leer. Leitet die Mining-Aufgabe an das Core weiter.
- **boolean addResourceToCore (resource: Resource):** Übergibt die Aufgabe der Rohstoffzugabe an das Core, falls es eine solche Objekt ist, welches ein Core (Kern) hat.
- **setMyPosition(newPosition: Position):** Setzt die Position eines Portals, falls solche Operation interpretierbar ist.
- **boolean isActive():** Gibt an, ob ein Portal, oder Objekt aktiv ist, also ob es wirklich funktioniert.
- **boolean setPair(pairGate: Gate):** Bindet das aktuelle Objekt mit einem Anderen zu.
- **SteppableSpaceObject getPair():** Gibt das Paar eines Portals/SteppableSpaceObject zurück.
- **String getInfo():** manche Information über das Objekt. Später wird es nützlich, beim GUI.
- **int getLayerThickness():** Gibt die Manteldicke einer SpaceObject zurück.

8.1.15 Sun

- **Verantwortung**

Startet Sunflairs, was gefährlich für Siedler und Roboter sind.

- **Schnittstellen / Interface**

EventObservable

- **Attribute**

- **Position position:** Position der Sonne auf dem Spielfeld.

- **Methoden**

- **doSunFlair():** Jede Entität auf dem Feld wird überprüft, ob sie in Gefahr ist, und wenn ja, stirbt sie.

8.1.16 Observable

- **Verantwortung**

Diese Schnittstelle hat Operationen, dadurch Entitäten (Siedler und Roboter) zur Kollektion von SteppableSpaceObjects hinzugefügt werden können.

- **Methoden**

- **checkOut(leavingEntity: Entity):** Das SteppableSpaceObject entfernt die Entität von sich.
- **checkIn(newEntity: Entity):** Das SteppableSpaceObject registriert eine ankommende Entität an sich.

8.1.17 Observer

- **Verantwortung**

Diese Schnittstelle hat das Aufgabe, dass die Entitäten auf SolarFlair, oder Asteroidexplosion reagieren. Diese Methoden der Interface hat die Kenntnisse, wie das aktuelle Entität das Ereignisse behandelt.

- **Methoden**

- **notifyFlairEvent()**: Benachrichtigt die Entität (und Spieler auch) über einem FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt.
- **notifyFlairDanger()**: Benachrichtigt die Entität (und Spieler auch) über einem bevorstehende FlairEreignis, und reagiert darauf mit Methodenaufrufe der konkrete Objekt. Oft nur ein Message auf dem Bildschirm, über die zukünftige Solarflair.
- **notifyAsteroidExplosion()**: Benachrichtigt die Entität (und Spieler auch), dass ihre Asteroid explodiert hat, und soll sie (die Entität) dieses Ereignis irgendwie abhängig von ihrem Typ behandeln.

8.1.18 Resource

- **Verantwortung**

Diese abstrakte Klasse wird von den anderen konkrete Ressourcen implementiert, die sich im Kern der Asteroiden (im ResourceStorage) befinden. Sie hat Operationen, dadurch die Name der Ressourcen abgefragt werden kann. Das kann man auch bestimmen, ob es radioaktiv ist.

- **Methoden**

- **String getName()**: Gibt den Namen (Typ) der Ressource zurück.
- **boolean isRadioactive()**: Sagt, ob die Ressource radioaktiv ist.
- **boolean equals(o: Object)**: Vergleicht zwei Ressourcen. Die sind gleich, wenn ihre Namen sind gleich.
- **int compareTo()**: Vergleicht zwei Ressourcen

8.1.19 ResourceStorage

- **Verantwortung**

Eigene Klasse für einfache Ressourcespeicherung mit einstellbare Kapazität.

Oftens wird als Container referenziert in der nachkommenden Beschreibung.

- **Methoden**

- **boolean storageIsFull():** Gibt das Containerzustand zurück, ob es voll ist. (ob man weitere Ressourcen einpacken kann)
- **Resource popResource(res:Resource):** Probiert ein bestimmte Ressource aus dem Container ausheben.
- **Resource popRandomResource(res:Resource):** Probiert ein zufälliges Ressource aus dem Container ausheben. Falls nur eine Ressource im Container ist, gibt diese einzige zurück.
- **boolean pushResource(res: Resource):** Probiert ein bestimmte Ressource im Container speichern, falls es noch nicht voll ist.
- **List<Resource> getResourceList():** Gibt das aktuelle Inhalt der Container zurück.
- **ResourceStorage popMore(count: int, res: Resource):** Gibt ein ResourceStorage mit gesuchte Menge der Resourcen aus dem originelle Container, falls es möglich ist.
- **boolean pushMore(res: Resource):** Probiert die bestimmte Menge der Resourcen im Container speichern, falls es noch möglich ist.
- **int countOf(type: Resource):** Gibt das Ressourcемenge der angefragte Ressource aus dem Container.
- **int getAllCapacity():** Gibt das Containerkapazität zurück.
- **setAllCapacity(allCapacity: int):** Stellt das Kapazitätsgrenze des Containers ein.

- **Attributen**

-resource: List<Resource>: Um Ressourcen zu speichern.

-allCapacity: int: Obere Schrank für Listenelemente.

8.1.20 Coal

- **Verantwortung**

Ressource vom Typ Coal.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Coal" zurück.
- **boolean isRadioactive():** Gibt false zurück.

8.1.21 Empty

- **Verantwortung**

Ressource vom Typ Empty.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Empty" zurück
- **boolean isRadioactive():** Gibt false zurück.

8.1.22 FrozenWater

- **Verantwortung**

Ressource vom Typ FrozenWater.

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "FrozenWater" zurück
- **boolean isRadioactive():** Gibt false zurück.

8.1.23 Iron

- **Verantwortung**

Ressource vom Typ Iron

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Iron" zurück
- **boolean isRadioactive():** Gibt false zurück.

8.1.24 Uran

- **Verantwortung**

Ressource vom Typ Uran

- **Schnittstellen / Interface**

Resource

- **Methoden**

- **String getName():** Gibt "Uran" zurück
- **boolean isRadioactive():** Gibt true zurück.

8.2 Die Pläne der Tests und ihre Beschreibung mit der Testsprache

8.2.1 CreateGateTest

- **Beschreibung:**

Testen das createGate, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Teleportpaar aus seinen Ressourcen machen.

- **Inspekierte Funktionalität: CreateGate**

- **Eingabe:** createGateTest.json

- **Erwartete Ausgabe:**

```
EVAL TEST : createGateTest
-----
GameController - setup ended : pattern found
Gate - New gate created : pattern found
Gate - New gate created : pattern found
-----
createGateTest : PASSED
-----
```

8.2.2 BuildGateTest

- **Beschreibung:**

Testen das buildGate, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Teleportpaar in dem Asteroidenfeld platzieren.

- **Inspekierte Funktionalität: BuildGate**

- **Eingabe:** buildGateTest.json

- **Erwartete Ausgabe:**

```
EVAL TEST : buildGateTest
-----
GameController - setup ended : pattern found
Gate - New gate created : pattern found
Gate - New gate created : pattern found
Settler - BuildGate called : pattern found
Settler - Gate placed at : pattern found
-----
buildGateTest : PASSED
-----
```

8.2.3 SunFlairTest

- **Beschreibung:**
Testen einen SunFlair Ereignis, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Im Test wird das Settler auf einem benachbarten Asteroide bewegen, dort auf dem SunFlair warten, und endlich gestorben.
- **Inspekierte Funktionalität: SunFlair**
- **Eingabe:** sunFlairTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : sunFlairTest
-----
GameController - setup ended : pattern found
Entity - move called : pattern found
ConsoleUI - Used automatic command: templ : pattern found
GameController - evaluateFlair called : pattern found
GameController - evaluateFlair called : pattern found
Settler - Die method of player testName : pattern found
-----
sunFlairTest : PASSED
-----
```

8.2.4 ListNeighbourTest

- **Beschreibung:**
Testen das Auszahlfunktion, in dem wir alle benachbarte Asteroide durchschauen können. Falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll.
- **Inspekierte Funktionalität: ListNeighbour**
- **Eingabe:** listNeighboursTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : listNeighboursTest
-----
GameController - setup ended : pattern found
Entity - listMyNeighbours called : pattern found
Entity - Possible neighbour: : pattern found
-----
listNeighboursTest : PASSED
-----
```

8.2.5 ExplodeTest

- **Beschreibung:**
Testen einen Asteroidexplosion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Asteroid völlig bohren, und das Asteroid soll im Sonnennahe mit radioaktive Material im Kern sein.
- **Inspekierte Funktionalität: Explode**
- **Eingabe:** explodeTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : explodeTest
-----
GameController - setup ended : pattern found
Entity - move called : pattern found
ConsoleUI - Used automatic command: temp0 : pattern found
Settler - Settler tried to drill an object : pattern found
Asteroid - drillLayer called, before drill was the layer: 1 : pattern found
Asteroid - Asteroid has radioactive core _and_ it's close to Sun --> EXPLODE : pattern found
Asteroid - explode called : pattern found
Settler - Die method of player : pattern found
-----
explodeTest : PASSED
-----
```

8.2.6 DrillTest

- **Beschreibung**
Testen das Drill Funktion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler bohren(drill).
- **Inspekierte Funktionalität: drill**
- **Eingabe:** drillTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : drillTest
-----
GameController - setup ended : pattern found
Settler - Drill called : pattern found
Settler - Settler tried to drill an object : pattern found
Asteroid - drillLayer called, before drill was the layer: : pattern found
Settler - Drill was successful : pattern found
-----
drillTest : PASSED
-----
```

8.2.7 MineTest

- **Beschreibung**

Testen das MineFunktion, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler fördern(mine).

- **Inspekierte Funktionalität: mine**

- **Eingabe:** mineTest.json

- **Erwartete Ausgabe:**

```
EVAL TEST : mineTest
-----
GameController - setup ended : pattern found
Settler - Mine called : pattern found
Asteroid - mineResource called : pattern found
Settler - addResource called : pattern found
Settler - Coal added to settler successfully : pattern found
-----
mineTest : PASSED
-----
```

8.2.8 DeployResource Test

- **Beschreibung**

Testen das deployResource, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(deployResource).

- **Inspekierte Funktionalität: deployResource**

- **Eingabe:** deployResource.json

- **Erwartete Ausgabe:**

```
EVAL TEST : mineTest
-----
Settler - DeployResource called : pattern found
Settler - ListResources called : pattern found
Settler - chooseResource called : pattern found
Settler - Write the number of the resource you would like to choose : 1 - Coal, 2 - FrozenWater, 3 - Iron, 4 - Uran : pattern found
Settler - The selected resource can be chosen : pattern found
Asteroid - addResourceToCore called : pattern found
-----
mineTest : PASSED
-----
```

8.2.9 Move Test

- **Beschreibung**

Testen das move, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(move).

- **Inspektierte Funktionalität:** moveTest
- **Eingabe:** moveTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : move
-----
SteppableSpaceObject - checkIn called : pattern found
Settler - doAction called : pattern found
Entity - move called : pattern found
Entity - listMyNeighbours called : pattern found
Entity - Possible neighbour: exampleInfo : pattern found
Settler - ChooseNeighbour called : pattern found
SteppableSpaceObject - checkOut called : pattern found
SteppableSpaceObject - checkIn called : pattern found
-----
move : PASSED
-----
```

8.2.10 Create Zone Test

- **Beschreibung**

Testen das createZone, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(createZone).

- **Inspektierte Funktionalität:** createZone
- **Eingabe:** createZoneTest.json
- **Erwartete Ausgabe:**

```
EVAL TEST : createZoneTest
-----
GameController - createAndNamePlayers called : pattern found
Sun - Sun constructor called : pattern found
SteppableSpaceObject - SteppableSpaceObject constructor called : pattern found
SteppableSpaceObject - SteppableSpaceObject constructor called : pattern found
Asteroid - Asteroid constructor called : pattern found
Asteroid - Asteroid constructor called : pattern found
Sun - getPosition called : pattern found
Sun - getPosition called : pattern found
Asteroid - Asteroid constructor called : pattern found
Sun - getPosition called : pattern found
GameController - dropSettlers called : pattern found
SteppableSpaceObject - checkIn called : pattern found
Sun - checkIn called : pattern found
-----
createZoneTest : PASSED
-----
```

8.2.11 Create Robot Test

- **Beschreibung**

Testen das createRobot, und falls alles stimmt, keine Fehlermeldung während der Ablauf ausschreiben werden soll. Dazu muss ein Settler ein Rohstoff zurückstecken(createRobot).

- **Inspektierte Funktionalität:** createRobot

- **Eingabe:** createRobotTest.json

- **Erwartete Ausgabe:**

```
EVAL TEST : createRobotTest
-----
GameController - getInstance called : pattern found
GameController - getInstance called : pattern found
SteppableSpaceObject - checkIn called : pattern found
Settler - Bot created at Home asteroid : pattern found
GameController - Iterate on robots, they do things: : pattern found
AIRobot - doAction called : pattern found
AIRobot - strat_1 called: AIRobot on strat1 : pattern found
Entity - move called : pattern found
Entity - listMyNeighbours called : pattern found
Entity - Possible neighbour: exampleInfo : pattern found
AIRobot - chooseNeighbour called : pattern found
SteppableSpaceObject - checkOut called : pattern found
SteppableSpaceObject - checkIn called : pattern found
-----
createRobotTest : PASSED
-----
```

8.3 Napló

Anfang	Zeitdauer	Teilnehmern	Beschreibung
2021.03.29.	2,5 óra	alle	Meeting: Aufgaben teilen, Problemen besprechen
2021.03.29.	4 óra	Pongrácz	Testvalidation implementieren
2021.03.29	1-2 Stunde	Hrotkó	Modifikationen implementieren
2021.03.30	3 Stunde	alle	Tests schreiben, Dokumentation schreiben
2021.03.29.	4 Stunde	Domokos	Ufo class, Dokumentation (Entwürfe von Klassen und Methoden)

10. Prototypeingabe

10.1 Übersetzungs- und Lauf Syllabus

10.1.1 Dateiliste

Dateiname	Grösse	Herstellungsdatum	Inhalt
AppController.java	6.9KB	2021.02.27.	AppController class
AsteroidZone.java	6.3KB	2021.02.27.	AsteroidZone class
GameController.java	16KB	2021.02.27.	GameController class
Player.java	2.6KB	2021.02.27.	Player class
AIRobot.java	5.7KB	2021.02.27.	AIRobot class
Entity.java	5KB	2021.02.27.	Entity class
Settler.java	18KB	2021.02.27.	Settler class
Ufo.java	6KB	2021.04.12.	Ufo class
EventObservable.java	350B	2021.02.27.	EventObservable interface
Observable.java	551B	2021.02.27.	Observable interface
Observer.java	529B	2021.02.27.	Observer interface
Coal.java	579B	2021.02.27.	Coal class
Empty.java	627B	2021.02.27.	Empty class
FrozenWater.java	614B	2021.02.27.	FrozenWater class
Iron.java	578B	2021.02.27.	Iron class
Resource.java	927B	2021.02.27.	Resource class
ResourceStorage.java	4.9KB	2021.03.16.	ResourceStorage class
Uran.java	1.6KB	2021.02.27.	Uran class
Core.java	2.2KB	2021.02.27.	Core class
Layer.java	1.4KB	2021.02.27.	Layer class
Asteroid.java	6.8KB	2021.02.27.	Asteroid class
Gate.java	3.6KB	2021.02.27.	Gate class
HomeAsteroid.java	1.7KB	2021.02.27.	HomeAsteroid class
Position.java	6KB	2021.02.27.	Position class
SteppableSpaceObject.java	4.1KB	2021.02.27.	SteppableSpaceObject class
Sun.java	4.5KB	2021.02.27.	Sun class
CallStackViewer.java	2.5KB	2021.03.08.	CallStackViewer class
ConsoleUI.java	3.4KB	2021.02.27.	ConsoleUI class
TestConfig.java	6.7KB	2021.03.21.	TestConfig class

10.1.2 Übersetzung

Mit gradle 6.8.3, man muss aus dem ideaProject Bibliothek in einem Terminal das folgende Kommando ausgeben:

```
./gradlew run
```

Gradle wird das Buildprozess automatisch durchführen.

Wenn es nicht funktioniert, vielleicht Gradle ist noch nicht auf dem PC heruntergeladen worden.

10.1.3 Programmstart, Programmablauf

Wie im vorigen Abschnitt beschrieben ist, mit gradle.

10.2 Testbeschreibungen, Testszenarien

10.2.1 AssertTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.2 BuildGateTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.3 CreateGateTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.4 CreateRobotTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d

Ergebnis	PASSED
-----------------	--------

10.2.5 CreateZoneTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.6 DeployResourceTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.7 DrillTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.8 ExplodeTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.9 ListNeighboursTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.10 MineTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d

Ergebnis	PASSED
-----------------	--------

10.2.11 MoveTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	7821f6d
Ergebnis	PASSED

10.2.12 SunFlairTest

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Testergebnis	FAILED
Mögliche Fehlergründe	Schlechte Testeingabe
Änderungen	Eingabe modifiziert, anstatt temp1 temp14 ist das neue Ziel für eine Bewegung.

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	850d56f
Ergebnis	PASSED

10.2.13 SunFlairTest2

Tester	Vince Pongracz
Testausführungsdatum	2021.04.13
Commit Hash:	850d56f
Ergebnis	PASSED

10.3 Bewertung

Mitglieder in	NEPTUN-Kode	Arbeitsverteilung in Prozent
Ábel Borbély	DRP0DT	25
Henrietta Domokos	J0DCPT	25
Renátó Hrotkó	OIT6HD	25
Vince Pongrácz	MKMDJO	25

10.4 Tagebuch (Journal Log)

Anfang	Zeitaufwand	Teilnehmer(n)	Beschreibung
2021.04.06.	3 Stunde	Pongrácz	Implementieren andere Verhalten von Uran, und kleinere Fehlerbehebungen.
2020.04.07.	1 Stunde	Alle	Besprechen weitere Aufgaben, Aufgabeausteilung. Besprechung über Asteroidenplatzierung, und Implementationstatus der neue Änderungen
2020.04.07	2-3 Stunde	Hrotkó	Zufällige generierung von asteroiden gelöst
2021.04.11.	4,5 Stunde	Borbély	Teleportgate ergänzen, neues Verhalten
2021.04.13	1 Stunde	Pongrácz	Testen durchführen, Dokument ausfüllen
2021.04.13	1 Stunde	Alle	Besprechung, grössere Refactor ist nötig beim Gate.

11. Spezifikation der graphischen Oberfläche

11.1 Die graphische interface

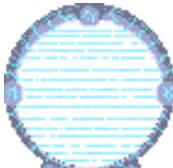
Icons:



Die Ufos die die Rohstoffe fördern können



Die Asteroiden aus denen die Rohstoffen gefördert werden können



Die Teleport Portale durch denen die Spieler teleportieren können



Die Roboter die den Siedlern helfen



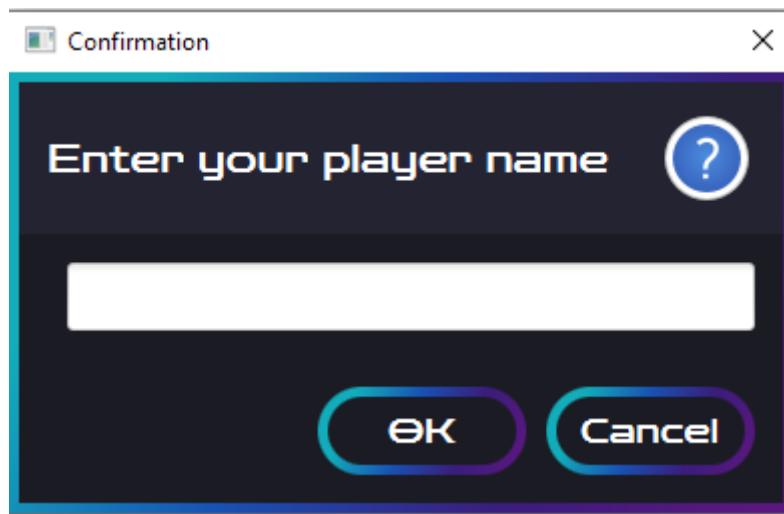
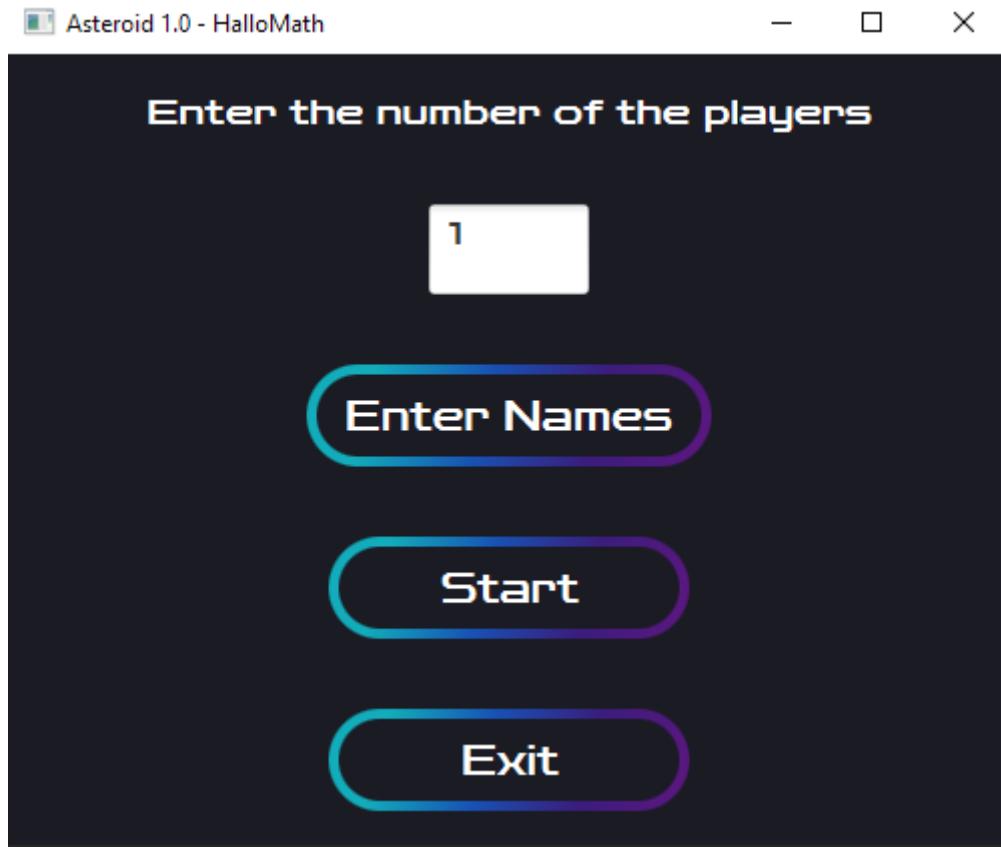
Die Raumschiffe der Siedler mit denen die fahren

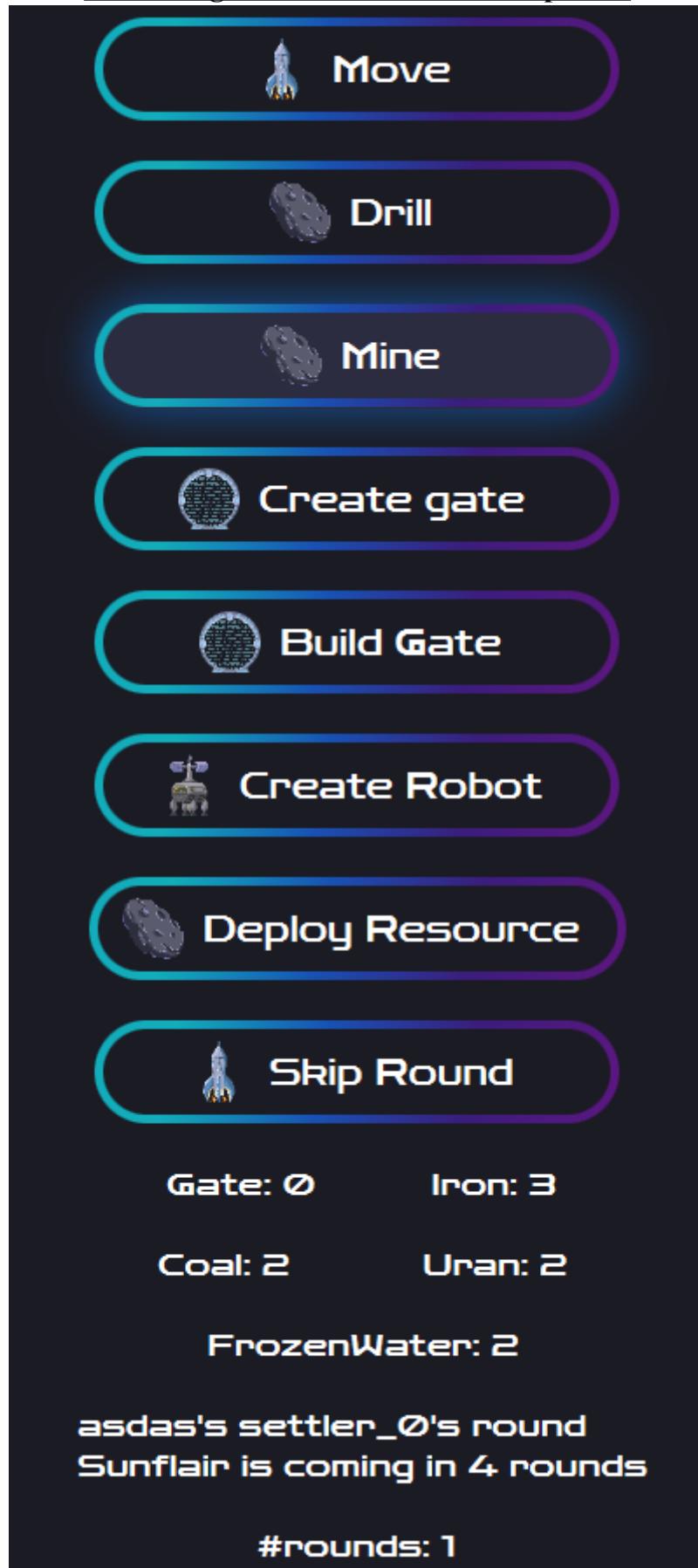


Die Sonne die ein Sonnenflair verursachen kann

Spielfeld:



Menüsystem am Anfang:

Rechts liegende Menü Panel beim Spielen:

11.2 Die Architektur der graphischen System

11.2.1 Funktionsweise der Oberfläche

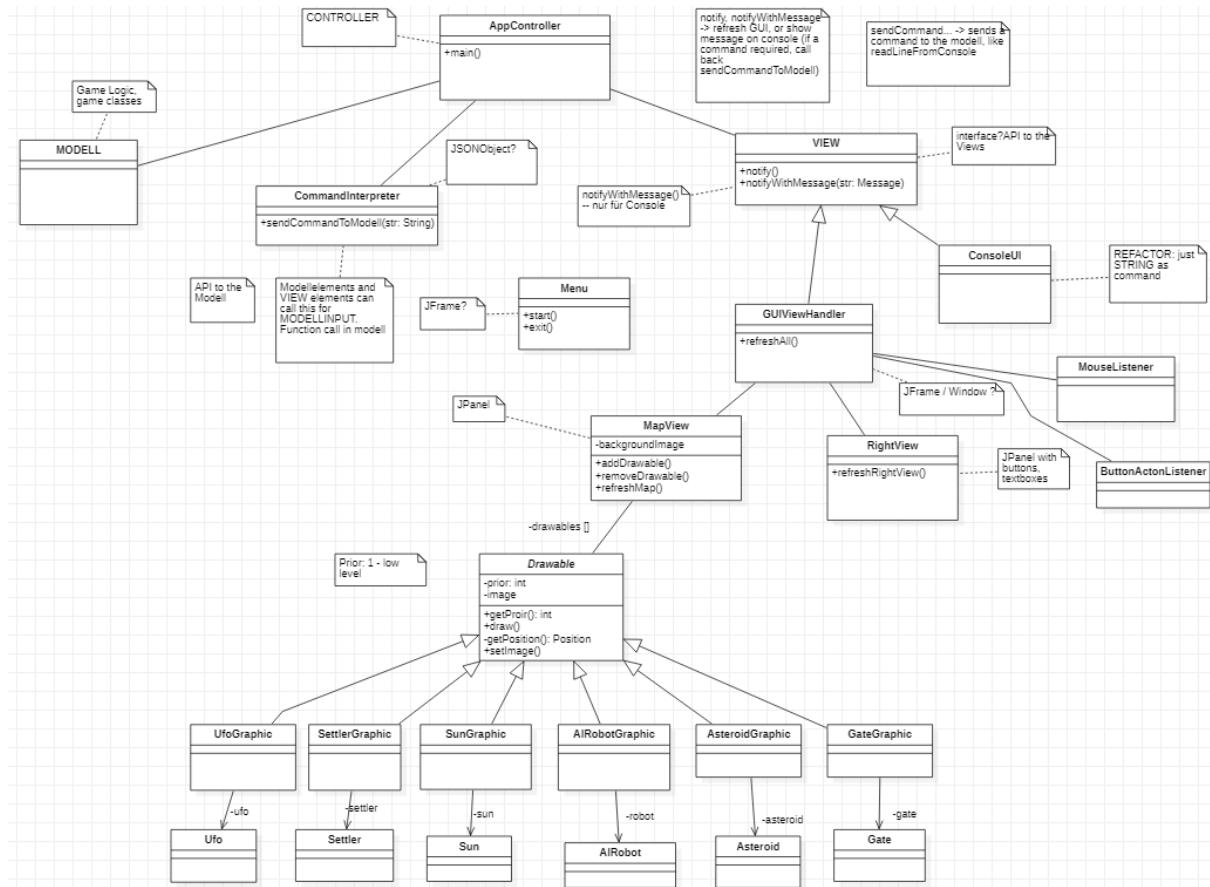
11.2.2 Die Klassenstruktur der Oberfläche

Das Architektur basiert auf dem MVC Pattern (Modell-View-Controller), wo das Spiellogik(Modell) durch das Controller erreichbar, und das Modell benachrichtigt das View über ihre Änderungen.

Durchschau:

- Modell: Das Spiel Logik, das wurde auf den vorigen, alten Klassendiagramme entwickelt. Es ist verantwortlich für die Datenbehandlung und Datenspeicherung, und das ganze Spiellogik.
- View: Verantwortlich für IO der Programm, behandelt das In und Output, informiert die Spieler über das Spielzustand.
- Controller: API zur Modell, dadurch werden die neuen Befehle, Aufrufe von dem View zur Modell geschickt, und auf dem Modell interpretierbare Sprache übersetzt.

Unsere Modell folgt das gemischte (pull and push) Prinzip, weil durch View wird kontrollerfunktionalität aufgerufen (es führt weiter zum Modell), und das Modell benachrichtigt auch das View, das etwas erfrischen soll.



11.3 Auflistung der graphischen Objekte

11.3.1 View

- **Verantwortung**

Abstrakte Basisklasse, das Konsole und das Graphische Anzeige bringt zusammen, und aktualisiert.

- **Methoden**

- **+void notify():** Benachrichtigt über dem Änderung.
- **+void notifyWithMessage(string Message):** Benachrichtigt über dem Änderung mit Text.

11.3.2 GUIViewHandler

- **Verantwortung**

Eine Klasse, die für die Anzeige der grafischen Elemente verantwortlich ist.

- **Basisklassen**

View → GUIViewHandler

- **Methoden**

- **+void refreshAll()**: Aktualisiert jeden Teil der GUI, so dass der korrekte Status des Spiels angezeigt wird.

11.3.3 ConsoleUI

- **Verantwortung**

Es ist für das Erscheinungsbild der Konsole verantwortlich und spielt beim Testen eine wichtige Rolle.

- **Basisklassen**

View → ConsoleUI

- **Methoden**

11.3.4 RightView

- **Verantwortung**

Diese Klasse ist für die grafischen Komponenten auf der rechten Seite des Fensters verantwortlich. Es ist ein Panel, das Schaltflächen für Befehle enthält und den Status des Spiels ausschreibt.

- **Methoden**

+void refreshRightView(): Aktualisiert die rechten Seite der grafischen Oberfläche.

11.3.5MapView

- **Verantwortung**

Diese Klasse ist für die grafischen Komponenten auf der linken Seite des Fensters verantwortlich. Es ist ein Panel, auf dem das Spielfeld ist.

- **Attribute**

-backgroundImage: Die Hintergrund mit anderen Bildern (UFOs, Asteroiden ..).

- **Methoden**

+void refreshMap(): Aktualisiert den Inhalt des Panels.

+void removeDrawable(): Fügt dem Pannel ein Element hinzu, das angezeigt werden soll.

+void addDrawable(): Entfernt ein Element, das für das Panel angezeigt werden soll

11.3.6 Drawable

- **Verantwortung**

Dies ist eine abstrakte Basisklasse zum Zeichnen jedes Spielobjekts, das Sie auf dem Spielfeld sehen.

- **Attribute**

-image: Enthält das dem Spielobjekt entsprechende Bild. So wird das Objekt auf dem Feld angezeigt.

- int prior: Diese Zahl stellt die Hierarchie der grafischen Objekte dar und zeigt an, ob ein Objekt ein anderes Objekt abdeckt oder von einem anderen abgedeckt wird.

- **Methoden**

+int getPrior(): Gibt prior zurück.

+void draw(): Zeichnet das Objekt, das die Klasse darstellt.

+Position getPosition(): Gibt die Position des Objektes zurück.

+void setImage(): Setzt das entsprechende Bild auf das Objekt.

11.3.7 UfoGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen die UFOs.

- **Basisklassen**

Drawable → UfoGraphic

- **Attribute**

- **Methoden**

11.3.8 SettlerGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen der Settler.

- **Basisklassen**

Drawable → SettlerGraphic

- **Attribute**

- **Methoden**

11.3.9 SunGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen die Sonne.

- **Basisklassen**

Drawable → SunGraphic

- **Attribute**

- **Methoden**

11.3.10 AIRobotGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen die AIRoboten.

- **Basisklassen**

Drawable → AIRobotGraphic

- **Attribute**

- **Methoden**

11.3.11 AsteroidGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen die Asteroiden.

- **Basisklassen**

Drawable → AsteroidGraphic

- **Attribute**

- **Methoden**

11.3.12 GateGraphic

- **Verantwortung**

Diese Klasse ist verantwortlich für das Erscheinen der Teleporter.

- **Basisklassen**

Drawable → GateGraphic

- **Attribute**

- **Methoden**

11.3.13 Menu

- **Verantwortung**

Es ist eine JFrame und ist für die Anzeige der Homepage verantwortlich.

- **Basisklassen**

- **Attribute**

- **Methoden**

- + **void start()**: Das Spiel beginnt mit dem Aufrufen dieser Funktion.
- + **void exit()**: Das Spiel beendet mit dem Aufrufen dieser Funktion.

11.3.14 CommandInterpreter

- **Verantwortung**

- **Basisklassen**

Interfészek

- **Attribute**

- **Methoden**

+sendCommandToModel(str: String):

Tagebuch

Anfang	Dauer	Teilnehmer	Beschreibung
2021.04.17	2 Stunde	Alle	Beschprechen der Realisationsmöglichkeiten der MVC Modell
2021.04.19	1 Stunde	Hrotkó	Bilder/Ikons modifizieren und ein Preview herstellen
2021.04.19.	1,5 Stunde	Pongrácz	MVC Planung, Diagramm herstellen
2021.04.20	2 Stunde	Alle	Dokumentation schreiben und Beschprechung
2021.04.20	45 Minuten	Domokos	Dokumentum Skeleton schreiben

13. Angabe der grafische Version

13.1 Übersetzung und Ablaufanweisung

13.1.1 Dateiliste:

```
[2.7K May 9 22:49] AppController.java*
[6.0K May 10 15:05] AsteroidZone.java*
[4.3K May 9 22:49] CommandInterpreter.java*
[ 16K May 10 15:22] GameController.java*
[3.1K May 10 15:05] Player.java*
[ 199 May 9 22:49] EventType.java*
[4.3K May 3 16:49] AIRobot.java*
[5.9K May 10 15:05] Entity.java*
[ 13K May 10 15:05] Settler.java*
[4.2K May 3 16:49] Ufo.java*
[ 250 May 10 15:05] AutoEntity.java*
[ 266 May 10 15:05] Drill.java*
[ 95 May 10 15:05] Mine.java*
[ 200 May 10 15:05] Moveable.java*
[ 113 Apr 29 18:11] MoveableObserver.java*
[ 598 May 3 16:49] Observable.java*
[ 201 May 10 15:05] Observer.java*
[ 593 Apr 29 18:11] Coal.java*
[ 641 Apr 29 18:11] Empty.java*
[ 628 Apr 29 18:11] FrozenWater.java*
[ 592 Apr 29 18:11] Iron.java*
[ 926 May 9 22:49] Resource.java*
[4.8K May 10 15:05] ResourceStorage.java*
[1.6K Apr 29 18:11] Uran.java*
[7.4K May 10 14:13] Asteroid.java*
[5.4K May 8 19:08] Gate.java*
[2.6K May 8 19:08] HomeAsteroid.java*
[6.0K Apr 29 18:11] Position.java*
[3.5K May 9 22:49] SteppableSpaceObject.java*
[4.8K May 5 9:03] Sun.java*
[2.8K May 8 19:08] Core.java*
[1.4K May 2 18:05] Layer.java*
[4.9K May 10 15:05] MapView.java*
[5.4K May 10 18:39] Menu.java*
[ 13K May 10 18:15] RightView.java*
[1.4K May 8 19:08] AIRobotGraphic.java*
[1.1K May 8 19:08] AsteroidGraphic.java*
[3.3K May 10 15:05] Drawable.java*
[1.4K May 8 19:08] GateGraphic.java*
[2.1K May 9 22:49] SettlerGraphic.java*
[1018 May 6 9:27] SunGraphic.java*
[1.4K May 8 19:08] UfoGraphic.java*
[1.1K May 8 19:08] ActionResponse.java*
```

```
[2.4K May 9 22:49] CallStackViewer.java*
[3.2K May 9 22:49] ConsoleUI.java*
[ 95 May 9 22:49] GameState.java*
[3.1K May 10 15:05] InitMessage.java*
[6.7K May 10 15:05] TestConfig.java*
[2.7K Apr 29 18:11] Alien.png*
[1.7K Apr 29 18:11] Asteroid_01.png*
[2.5K Apr 29 18:11] Portal.gif*
[3.2K Apr 29 18:11] Robot.png*
[2.1K Apr 29 18:11] Spaceship.gif*
[ 31K Apr 29 18:11] Sun.gif*
[7.2M Apr 29 18:11] asteroid_game.gif*
[1.8K Apr 16 17:11] log4j2.xml*
[2.0K May 10 15:05] style.css*
```

15 directories, 57 files

Testdeskriptors:

```
[ 493 Apr 29 18:11] assertTest.json*
[ 574 Apr 29 18:11] buildGateTest.json*
[ 482 Apr 29 18:11] createGateTest.json*
[ 966 Apr 29 18:11] createRobotTest.json*
[ 976 Apr 29 18:11] createZoneTest.json*
[ 753 Apr 16 17:11] deployResourceTest.json*
[ 614 Apr 29 18:11] drillTest.json*
[ 843 Apr 29 18:11] explodeTest.json*
[ 780 Apr 16 17:11] idxOutOfTest.json*
[ 440 Apr 29 18:11] jsonConfigTemplate_DO_NOT_OVERWRITE.json*
[ 520 Apr 29 18:11] listNeighboursTest.json*
[ 556 Apr 29 18:11] mineTest.json*
[ 740 Apr 29 18:11] moveTest.json*
[ 687 Apr 29 18:11] sunFlairTest.json*
[ 699 Apr 29 18:11] sunFlairTest2.json*
```

0 directories, 15 files

13.1.2 Übersetzung und Installationsanweisung

Mit gradle 6.8.3, man muss aus dem ideaProject Bibliothek in einem Terminal das folgende Kommando ausgeben:

`./gradlew run`

Gradle wird das Buildprozess automatisch durchführen.

Wenn es nicht funktioniert, vielleicht Gradle ist noch nicht auf dem PC heruntergeladen worden.

13.1.3 Programmablauf, Programmstart

Wie im vorigen Abschnitt beschrieben ist, mit gradle.

13.2 Bewertung

Name	Neptun	Arbeit in Prozent
Borbély Ábel	DRP0DT	20%
Domokos Henrietta	J0DCPT	23%
Hrotkó Renátó	OIT6HD	27%
Pongrácz Vince	MKMDJO	30%

13.3 Tagebuch

Anfang	Zeitdauer	Teilnehmer	Beschreibung
04.27	2 Stunde	Alle	Meeting
04.28	1 Stunde	Alle	Meeting
05.02	3 Stunde	Alle	Meeting
05.05	2 Stunde	Alle	Meeting
05.07	1 Stunde	Alle	Meeting
05.11	2 Stunde	Alle	Meeting

14. Zusammenfassung

14.1 Mit dem Projekt verbrachte Zeitdauer

Name	Zeitdauer
Borbély Ábel	70 Stunde
Domokos Henrietta	81 Stunde
Hrotkó Renátó	114 Stunde
Pongrácz Vince	131 Stunde
-	
Insgesamt:	396 Stunde

Anzahl der hochgeladene Programmzeilen

Phase	Anzahl von Zeilen
Skeleton	710
Prototyp	3010
GUI	1130
Insgesamt:	4850

14.2 Zusammenfassung der Projekt

14.2.1 Was haben Sie aus dem Projekt konkret und im Allg. gelernt?

Alle: In einem Team zu arbeiten ist schwer, aber gleichzeitig zu arbeiten macht die Sache einfacher. Wenn jemandem keine Aufgaben zugewiesen sind, wird diese Aufgabe nicht ausgeführt. Nach viel Planung und Dokumentation ist es einfach zu programmieren. Wir kannten die Methode, aber es half uns sehr, sie in der Gehirnpraxis zu sehen, um wirklich einen Sinn daraus zu machen. Es war auch nützlich, verschiedene Programme und Sprachen (Github, Jira, ...) kennenzulernen, die in Zukunft nützlich sein könnten.

Vince: Es ist sehr schwierig die Aufgaben aus und verteilen. Ich habe sehr viel über Git, gradle und architekturelle Problemen, Planung gelernt, weil ich habe die Mehrheit die Planungsaufgaben. Deswegen habe ich die meiste Dinge über Details gewusst, also alle Frage kommt zu mir :D

14.2.2 Was war das Leichteste und das Schwerste?

Alle: Das Entwerfen des Frameworks war vielleicht das schwierigste, da es sehr schwierig ist, eine Architektur zu entwerfen, die alle Änderungen und alle hinzugefügten Dinge überlebt und ohne größere Änderungen funktioniert. Ohne unser exzellentes Team wäre es nicht einfach gewesen, in einem Team zu arbeiten.

Vince: Ich habe keine Ahnung, welche Aufgabe war die leichteste, die Schwerste waren viel :D Das Schwerste war, wenn wir die ganze Spiellogik umstrukturieren müssen haben, weil das vorige Architektur nicht eine ereignisbasierte Lösung war, aber für GUI sollten wir solch eine Logik finden. Eine andere Schwierigkeit war, wenn wir 3-4 Tage für eine GUI Problem gekämpft haben.

14.2.3 War die Zeitaufwand und die Punktzahl mit dem erledigenden Aufgaben?

Vince: Ich weiss nicht, wie viele Punkte wir bekommen haben. Für mich war die Mehrheit des Projektes ein gutes Erlebnis (außerdem zu viele sinnlose Dokumentation), also es macht nichts für mich, wenn mehrere Zeit auf diese Projekt angewendet habe.

14.2.4 Falls nicht, dann wo hat es Schwierigkeiten verursacht?

14.2.5 Was für Änderungen würden Sie empfehlen?

Alle: Bei der Planung gibt es ein oder zwei Aufgaben, die zuvor mit sehr kleinen Änderungen aufgenommen wurden und daher keine sinnvolle Arbeit leisten, sondern nur die Zeit vergeht.

Vince: Wenige Dokumentation, und mehrere Technik und Kenntnisse davon, wie kann man ein Team gut leiten, wie kann man die Aufgaben verteilen, wie kann man die Prozeduren besser und alles zusammen als Team machen. Beim Technik denke ich hier darauf, dass wir über Versionskontroll, Testtechniken, build-Tools, Organisationstechniken und Praktiken mehr in vorigen gelernt haben können. So viele Dinge, die im realen Leben und Arbeit sehr nützliche Skills sind.

14.2.6 Was für Aufgabe würden Sie für einem Projekt empfehlen?

Alle: Ein solches ist perfekt für diesen Zweck geeignet, da am Ende nicht nur ein einmaliges Programm erstellt wird, sondern auch ein funktionierendes Spiel.

Vince: Ein Informationssystem(wie Neptun oder andere Datenbasierte Systemen) planen und verwirklichen mit Datenbanken, mehrerer Kommunikation (durch Netz oder sowas) und andere spannenden Techniken.

14.2.7 Andere Kritiken und Empfehlungen:

Vince: Zu viele sinnlose Dokumentation. Wir konnten die Kode nicht hundertprozentig dokumentieren, weil die andere Dokumentationen geschrieben werden sollen.