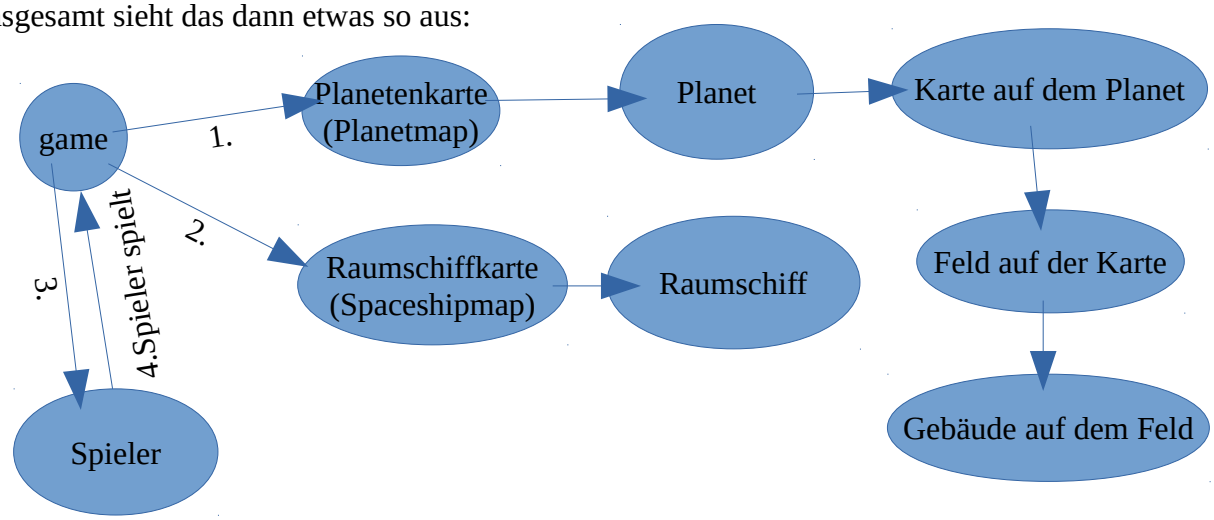
Anforderungsdefinition (Python)

Das Spiel ist hauptsächlich in 2 teile geteilt:
1. die Logik und 2.die Benutzeroberfläche

1. die Logik:

Die Hauptklasse der Logik ist game (das Spiel an sich), es enthält alles was für eine funktionierendes Spiel notwendig ist. Sie enthält die Karten (Spaceshipmap und Planetmap) und die Spieler (players).
Die Maps enthalten nur Planet und Spaceship Objekte und haben eine Methode um jeweils die Objekte hinzuzufügen. Die Runden werden auch von Game aus gesteuert, welches eine Methode get_current_player() enthält um den Spieler der an der Reihe ist abzufragen.
Es gibt eine gae.turn_start() welches bei Allen Spielern reihum die Methode turn_start() aufruft, bis diese false zurückgibt, was bedeutet, dass der jeweilige Spieler kein KI ist und seine Runde selbst beendet. Da den Spielern beim ihrer Erstellung das Spiel übergeben wird (*keine Kopie des Spiels, das kann man sich eher eine Referenz/Zeiger vorstellen, allerdings wäre technisch „geteiltes Objekt“ (shared object) korrekt*) könnnen sie num die methoden wie send_spaceship aufrufen, welche das senden von Raumschiffen ermöglicht.
Außerdem ruft game.turn_start() auch die Methoden der Karte (auch turn_start()) auf welche den Rundenbeginn dann weiter an ihren Inhalt geben, z.B. bis zu Spaceship.turn_start() wo das Raumschiff gespeichert hat wie es sich bewegt, bzw. Planet.turn_start() welches dann wieder von allen Gebäuden den Rundenbeginn Aufruft.
Insgesamt sieht das dann etwas so aus:



Um Gebäude zu Bauen oder Raumschiffe zu senden muss auch wieder eine Methode von game aufgerufen werden, welche dann folgendes tuen:
1.game.send_spaceship(from, to, num)
Diese Methode Kontrolliert zuerst ob genügend Material zum senden auf dem Planeten vorhanden ist, dazu ruft sie eine statische funktion can_send von der Raumschiffklasse auf. Danach ruft sie die add_spaceship Methode der Spaceshipmap auf welche das Raumschiff auf die Karte bringt.

2.build_building(planet, building_class, etc.):
Sie Ruft von dem Planeten auf dem das Gebäude gebaut werden soll die build_building Funktion auf, welche dann letztendlich wieder durchgereicht wird (wie beim Rundenbegin in der Graphik oben) bis zu der building_class (in der Graphik „Gebäude auf dem Feld“).
Dieses enthält auch eine statische Methode, welche prüft ob das Gebäude gebaut werden darf, und wenn ja wird setzt build_building das jeweilige Gebäude als Inhalt (content) des Feldes und ruft dir Erstellungsfunktion (unter Python __init__) auf.

2. Die Benutzeroberfläche

Die Benutzeroberfläche besteht aus 2 Tabs einmal der Karte mit den Planeten und zum Senden der Raumschiffe. Bei diesem kann man unten in ein Edit-Feld die Anzahl der Raumschiffe die man senden will eintragen (Achtung die Eingabe ist wie beim Passwort nur mit Punkten, falls mehrere Spieler gemeinsam spielen). Den Start und Ziel Planeten muss man mit einem Einfachen Klick auswählen (den Start Planeten zuerst) und dann zum Bestätigen auf den „Send“-Button clicken. Um die Runde zu beenden klickt man dann im gleichen Tab auf „Next Turn“.
In dem Zweiten Tab sieht man Oben eine Karte des Aktuell ausgewählten Startplaneten, auf der man die Gebäude bauen kann, darunter befindet sich eine ComboBox zum Auswählen des Gebäudes das man bauen will, und darunter Steht auf einem Label, welche Rohstoffe gerade auf dem Planeten zu Verfügung stehen. Wenn der ausgewählte Planet nicht dem Spieler der gerade an der Reihe ist gehört, wird keine Karte angezeigt, außerdem steht auch nur der Name des Besitzers (owner) auf dem Label. Auf einem zweiten Label daneben steht der Besitzer des Zielplaneten. Beide Karten können mit der `+` und der `-` Taste rein/rausgezoomt werden und haben am Rand Scrollbars zum Scrollen. Im Menü steht nur ein Eintrag nämlich „new game“ um ein neues Spiel zu Beginnen.

Anforderungsdefinition (Lazarus)

Bei der Lazarus-Version habe ich keine starke Unterscheidung zwischen Logik und Benutzeroberfläche (GUI) erstellt, da es einfacher aufgebaut ist und es daher nicht benötigt wird. Die Oberfläche löst direkt methoden von Tgame aus, welche auch wieder teilweise auf das GUI zugreifen um z.B. das Edit-Feld auszulesen (siehe Problemanalyse).
Die Tgame klasse enthält wiederum alle Raumschiffe, Spieler und Planeten jeweils als dynamisches Array, die Karte wird durch ein 2 dimensionales dynamisches Array realisiert, von der jeder wert darin den Index des Planeten an dieser Position angibt, -1 bedeutet, dass dort kein Planet ist.
Die Methode next_player() funktioniert sehr ähnlich zu der in der Python Version, allerdings muss hier nicht auf KI-Spieler rücksicht genommen werden, weshalb diese Methode nur von dem „Next Player“ Button ausgelöst wird. Dabei wird einfach das Attribut von der Tgame instanz game um eins hochgezählt und wenn sie bei der gesamt-spieler zahl angekommen ist, wird sie zurückgesetzt. Die Methode click_on_screen() bewirkt, das game speichert auf welchen Planeten geclickt wurde und damit den Start und Endplanet für das Raumschiffsenden festlegt.
Dazu werden die Methoden set_target/set_startplanet ausgelöst. Die refresh_screen()-Methode greift auch direkt auf das GUI zu und diese Zeichnet auch das Spielfeld, sie wird nach jedem Rundenstart aufgerufen, um das verstecken der gegnerischen Raumschiffe zu gewährleisten und um die Planeten je nachdem wer an der Reihe ist zu markieren. Sie wird auch nach dem senden eines Raumschiffes aufgerufen. Ähnlich wie in der Python Implementierung wird auch hier beim Rundenbeginn für den Planeten die methode turn_start aufgerufen, welche bewirkt, dass Raumschiffe auf diesem produziert werden, und bei dem Raumschiff wird on_step aufgerufen, um dieses weiter fliegen zu lassen.