Planet Bound
Report
Final phase

Jedrzej Szor

June 11, 2020

Abstract

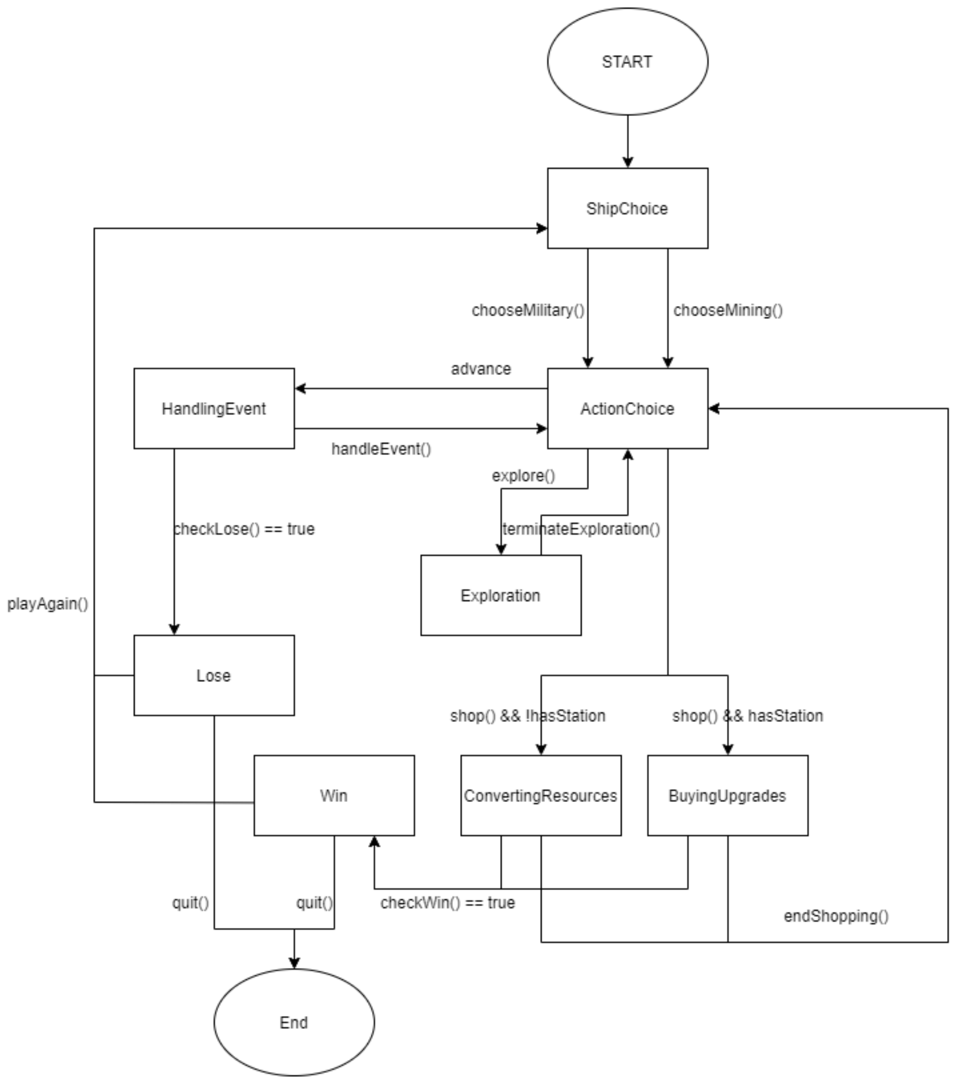
The report is a conclusion of work done to implement a board game Planet Bound. For the final phase checkpoint the polymorphic state machine, definition and implementation of the data structure, object factories, logger, serialization as well as a fully functional GUI have been implemented.

1 Brief description

Aside from the polymorphic state machine I have decided to implement the Singleton programming pattern in order to securely and conveniently access all major objects with minimal usage of static methods. My Singleton object controls the entire logic and contains instances of the current state, current planet, ship etc. Concerning objects like aliens, planets, events and resources, where there can be different types of them, I have implemented abstract classes, inheriting classes and factories with static methods. During implementation of GUI I have used javaFX and provided by this package primitive types' properties and data binding.

2 State machine

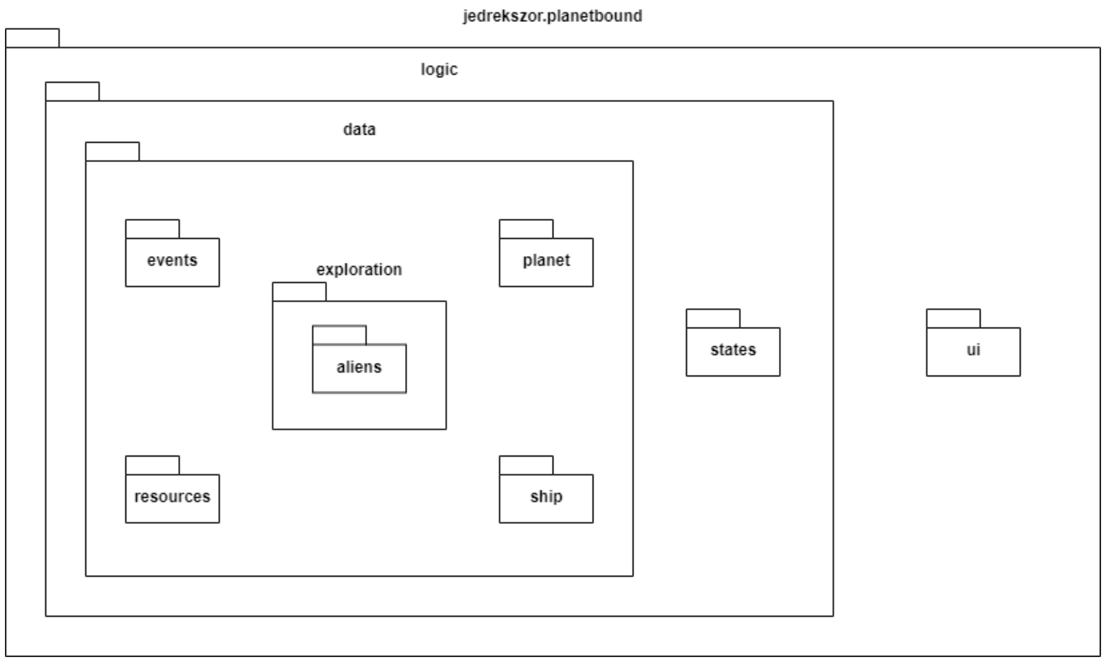
Figure 1: State machine diagram



3 Classes

My project is structured in the following way:

Figure 2: Package diagram

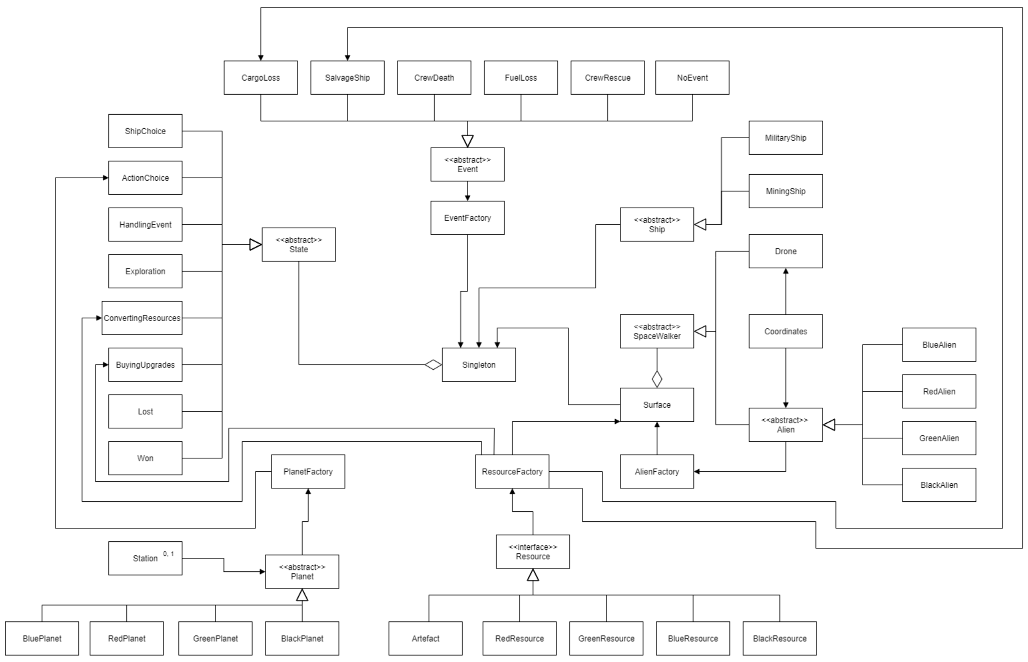


The "ui" package contains controller classes responsible for controlling the widgets appearing on the screen. "Logic" package is responsible for the entire logic of the application. It contains classes performing all the actions as well as classes working as data structures. In that package there is also class Singleton which works as the main controller of the game containing all of the major objects and being able to be accessed from anywhere. The "states" package contains the abstract class State and classes representing all of the states, which inherit from the State class. "Data" contains classes working mostly as data structures and representing ship, alien, etc. Package "events" contains abstract class Event, descendant classes representing every event and the EventFactory class with static method for returning random event. "Planet" has very similar structure but contains also a Station class which can be an attribute of Planet if it has a space station orbiting around it. The "resources" package contains interface Resource, classes representing different resources, artefact and the ResourceFactory. I used an interface here and abstract classes in previous cases because I did not need the Resource class itself to have any methods, as I did with Planet, Event and Alien. Package

”ship” contains an abstract class Ship and inheriting classes MiningShip and MilitaryShip. The ”exploration” package contains all of the logic necessary for managing the part of the game happening on the surface of the planet. It contains an abstract class SurfaceWalker which is a base for both Drone and Alien. It also contains class Surface, which is the main controller of the logic on the planet’s surface and class Coordinates, which represents the position of SurfaceWalker. Package ”aliens” is very similar to the ”planet”, ”events” and ”resources”. It contains an abstract class Alien, classes representing different Aliens and an AlienFactory.
In the second phase i have added a class Logger with static method log which appends the given string to the file. When a new game is started it is indicated with a timestamp. I have also implemented a StateAdapter class which handles all situations when the method cannot be called by a certain state.

4 Relationships between classes

Figure 3: Class relationship diagram without methods and attributes for better readability



5 Status of implementation

Feature	***Status of implementation***
Ship choice	Fully implemented
Proceeding to the next planet	Fully implemented
Drawing and resolving event	Fully implemented
Possibility to travel through a wormhole	Fully implemented
Possibility to land on the planet	Fully implemented
Movement on the surface of the planet	Fully implemented
Drawing and spawning aliens	Fully implemented
Aliens moving towards drone	Fully implemented
Fighting with aliens	Fully implemented
Buying resources	Fully implemented
Upgrading systems on station	Fully implemented
Win and lose conditions	Fully implemented
Possibility to play again	Fully implemented
GUI	Fully implemented
Log system	Fully implemented
Save and load game	Fully implemented