Galaxy Game Documentation

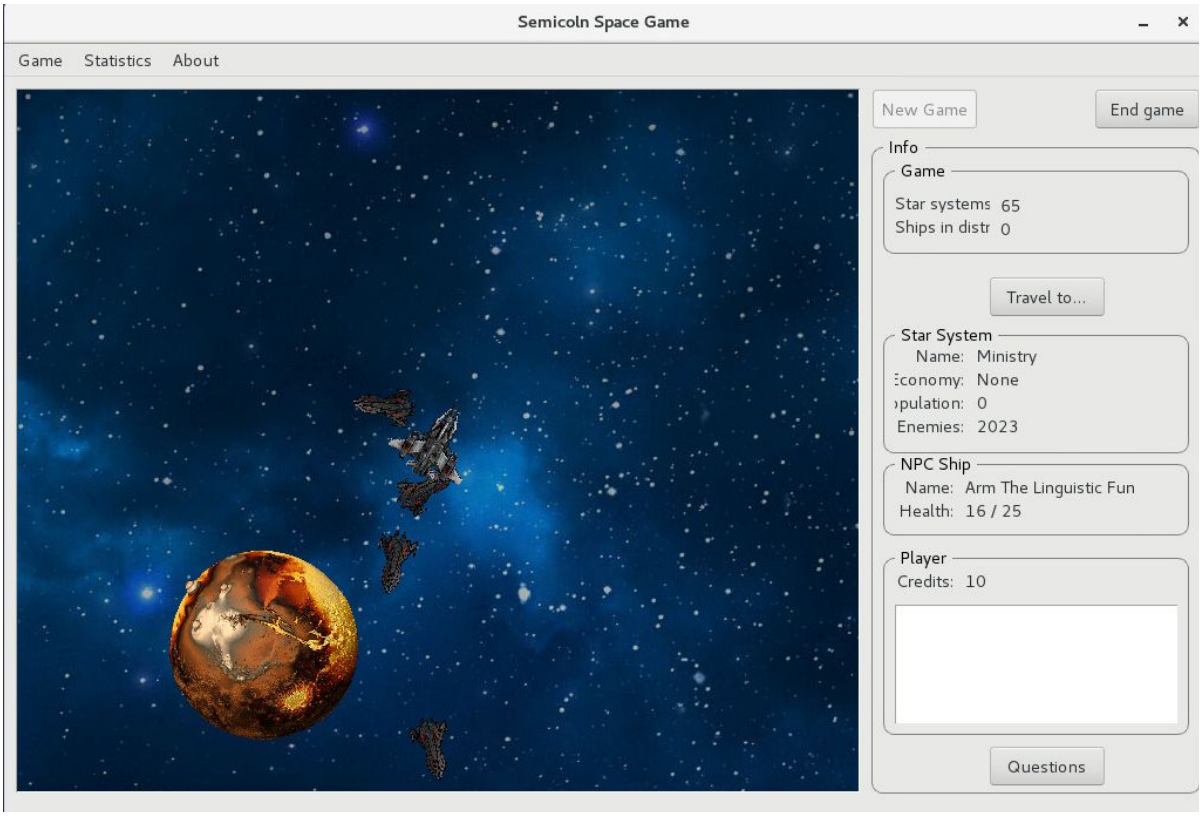
Team Semicolon
Andrej Vlasatý && Lucie Kuchárová

Game overview

Player, as a hero of whole Galaxy, is supposed to save ships, which are in some kind of trouble, by repairing their engines. Player travels among Star Systems and searches for broken ships. In addition, player can trade with planets in every Star System and fulfill Questions, which are requests from certain planets to obtain certain goods.

Game start

Game starts by clicking on “New game” button. User is asked to fill in a nickname. This nickname has to contain at least one character, can’t be longer than 10 characters and mustn’t contain any special characters. Galaxy is then created, filled with Star Systems and randomly generated planets and ships in them. Player is put into randomly chosen Star System and game objects are displayed.



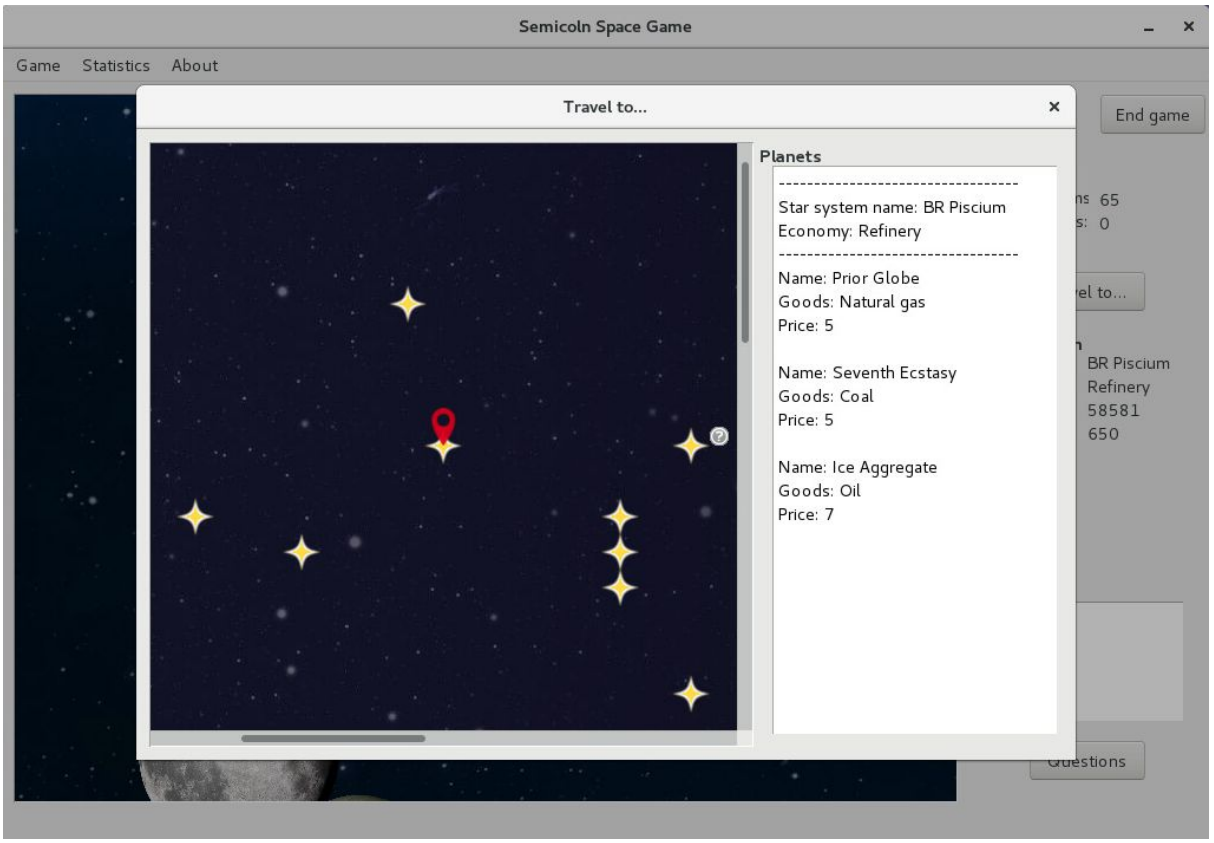
Screenshot 1 - Game Start

Game environment

Let's take a look at *Screenshot 1 - Game Start* above. Area on the left represents single star system view (one "zoomed-in" star system). There player, NPC ships and planets can be seen. On the right panel, information about game and action buttons are shown. There is info about whole galaxy (number of Star Systems in galaxy and ships which need help), info about current Star System in which player is located, info about current game object with which can player interact and player's inventory. "Travel to..." button displays map and "Questions" displays active questions in game, which player can complete or already completed.

Map

Map is displayed through "Travel to..." action button. It is the only way for player to travel among Star Systems. In an area on the left (*Screenshot 2 - Map window*) every Star System is shown. **One click** on Star System icon shows information about it and it's planets, which are located there, and goods with price, which they offer, on the right panel of map window. **Double click** on Star System icon causes reallocation of player ship to chosen location and map window is hidden.



Screenshot 2 - Map window

Every Star System can be marked with icons:

- Red location mark = Player location
- Red "broken heart" mark = Star System contains ship in distress
- Grey question mark = Question destination (goods to be delivered there)

Controls

Player can move in Star System view using up, down, left and right **arrow keys**. When player ship reaches planet or ship, clicking on **spacebar** key starts interaction between them. Any other interaction with game is done by mouse clicks. If player ship loses focus and movement through arrow keys is not possible, mouse click on player ship returns focus to player and movement is possible again.

Statistics

In top menu under “Statistics -> My statistics” current player’s statistics can be found.
Player collects points during a game. Points are obtained by saving ships or completing Questions. Points indicate player’s overall score in game.
Credits are used for trading with planets. Are obtained by saving ships or completing trade Questions. Are reduced by buying goods from planets.
Statistics also keep track of number of Questions completed and saved and abandoned ships.

Saving ships

Player can repair ships during their movement, if they have their health decreased. Player is then rewarded by 1-2 credits. When NPC ship is broken so much that it cannot move, number in main window (next to a label “Ships in distress”) is increased. User can find location of this ship in map window. There, Star System, in which broken ship is located, is marked with red icon. Player has to travel to this Star System, find non-moving ship and locate himself right on the top of it. Ship is being repaired by hitting “space” key. User has to hit space more than once for ship to be fully repaired and therefore saved. Player is rewarded by points and credits.
Note that there can be more broken ships in one Star System, so repairing one ship doesn’t have to make the red warning icon disappear from map view. Red icon is not visible only if there are no broken ships in Star System.
When player fails to save broken ship in certain amount of time, ship is abandoned and player can’t interact with it anymore.

Trading with planets

When player arrives to planet icon, hitting “space” bar starts trade action. User can choose between sell and buy actions. Buy action transfers planet’s goods into player’s inventory (every planet has only one type of tradable goods) and reduces certain amount of credits from player’s account. Sell action is allowed only when planet’s request for goods is listed in Questions and player has certain goods in his inventory. Size of player’s inventory is limited.

Questions

Questions are listed in question window under “Questions“ button. Questions are divided into active (on the left) and completed (on the right). Player has to obtain certain goods and deliver it to certain planet. When question is completed, player is rewarded by points and

credits. Star systems, where a questions can be completed, is marked in map with “?” icon. Note that this icon stays on the map even when question is completed.

Game end

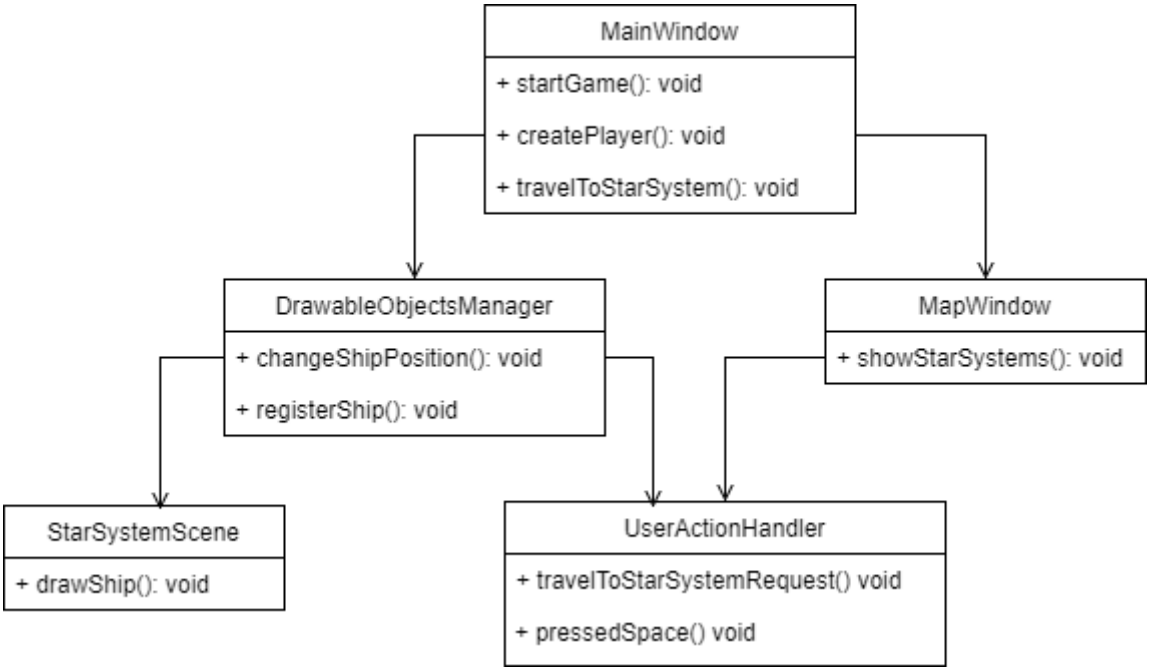
Game ends by hitting “End game” button **only**. Player’s statistics are saved and added to overall statistics accessible from Statistics ->Top list. Order of players is descending according to acquired points (player with the most points is on the top of the list). Note that if player uses the same nickname that was used and saved before, previous result are overwritten by the new one.

Implementation

All classes, their methods and attributes, invariants, pre and post conditions can be inspected via Doxygen documentation, following Design by contract approach. Here is a brief overview of important classes and their behavior.

Important classes

As can be seen from screenshots above, there are two windows used for interacting with game objects - main window with it’s single star system view (StarSystemScene) and map of all star systems (MapWindow). Both of them (and all other windows in program) are created in **MainWindow** object, which keeps track of game events, connects events from handlers to objects that deals with them and manages initialization of new game.



StarSystemScene is an area where main part of the game is played. Because of certain incompatibility between course side coordinates (where every object in one star system has

the same coordinates) and our intentions (every object in one star system has own coordinates according to its position inside star system, so that player can move between them), we somehow needed to simulate non-existing movement of NPC ships inside star system. That is done simply by adding a random movement to these ships. This means that every object in StarSystemScene has one coordinates given by its star system location and other “extra” coordinates, which determine exact position is star system. When NPC ship decides to travel to different Star System, it is removed from the view.

Everything that happens in single star system view (in StarSystemScene object) is managed by **DrawableObjectsManager**. This class is responsible for creating UI representations of base Ship objects, pairing right UI representations with right base objects, catching signals from EventHandler and manage every change which causes a need to redraw objects in StarSystemScene. Simplified example of it’s behavior:

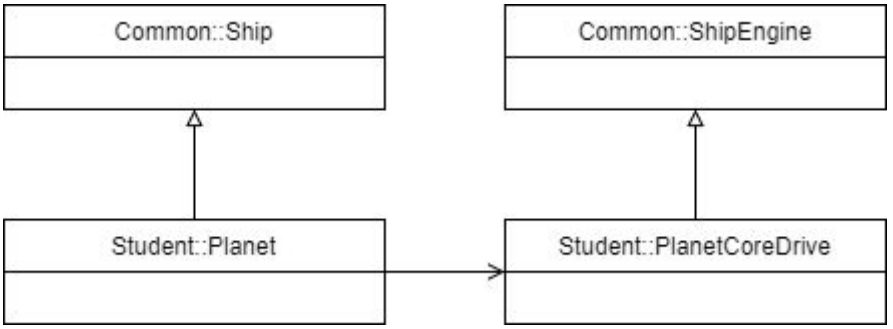
“New Ship is spawned -> EventHandler emits signal -> signal is caught in registerShip(ship) slot in DrawableObjectsManager -> DrawableObjectsManager creates new UI object (which will be drawn to scene as visualisation of the ship) and stores both of them as a pair -> if the ship is in the same star system as player and is in it’s range of view, ship UI object is drawn to scene.”

Similarly to DrawableObjectsManager, **MapWindow** works as “connector” to base StarSystem object and it’s UI representation in map.

Every user action is caught by **UserActionHandler** - in single star system view, if player presses space bar, UserActionHandler gets to know about this and sends signal to others. Another example is double click in MapWindow, which stands as request for player travel and again, signal is emitted about this.

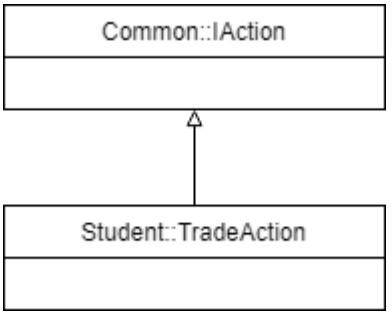
Custom ship and engine

Planet object is actually child of Ship object. **Planet**s are static in our game, therefore they use custom **PlanetCoreDrive** engine. This engine isn’t supposed to travel and keeps Planet in StarSystem where it was spawned. This “ships” are spawned randomly via GameRunner (together with CargoShips).



Custom action

TradeAction in game is implemented as child object of IAction class. TradeAction is possible only with instances of class Planet.



Implemented features and work division

As this project is a teamwork of two participants, code review, refactoring, architecture changes or bug fixing caused implemented parts of the code to blend in and following list is not strictly determined by single person. Division is done approximately by “most of the work done” or “initial idea and initial implementation”.

Minimum:

- The program contains a main user interface where at least the star systems and ships in the game are shown.
    - Andrej = single star system view (ships)
    - Lucie = map view (star systems)
- The location of the player's ship can be changed without delay or animation (e.g. click of the cursor indicates the new location or the selected star system)
    - Andrej = player ship movement in single star system view
    - Lucie = travel done via map window
- The player's ship works in some kind of interaction with the non-player ships. The non-player ships react to the interaction with the player's ship.
    - Andrej = player ship interacts with Planets (trading)
    - Lucie = player ship interacts with NPC ships (saving)
- The Galaxy class inherited from iGalaxy passes its unit tests.
    - Andrej
- The game is turn based. The game keeps score of the players actions and possible the game is won or lost based on the score i.e. there are goals to aim for in the game.
    - Andrej = ending the game and winning by highest score
    - Lucie = keeping score of player’s actions

- The statistics class implemented by the student team has unit tests implemented to it. The functionality required in defined in the interface iStatistics.
    - Andrej = unit tests update
    - Lucie = implementation of Statistics and basic unit tests

Basic:

- The game can handle errors that occur in handling game setting files.
    - Andrej = handling star systems with the same position, handling problems in Assets folder, settings file
    - Lucie = game items file
- The game implementation contains at least two separate windows and/or dialogs.
    - Andrej = Main window, Help, About, Top list
    - Lucie = Map window, current Statistics
- Game state update. The statistics collected during the game are shown to the player. The data is updated as the game is played.
    - Lucie
- Star system information. On the map, or otherwise is a player readable way, information on the star systems is shown. E.g. size of the population, financial system etc.
    - Andrej

Extra:

- Real time game play.
    - Andrej = real time movement of player ship, periodic game events
    - Lucie = real time movement of NPC ships
- Even screen update. The game area is updated in even intervals with the changed information instead of updating each change separately. The QTimer-class may be useful.
    - Andrej = refreshTimer_ handling UI, collisionTimer_ handling collisions between objects, gameTimer_ handling game acitons
- Minimal screen update. Changes in the state of the actor only its immediate surroundings are updated instead of drawing the entire map again.
    - Lucie = visibility of objects changes according to player’s position in single star system view
- Scrollable map. The game area is not tied to a fixed view but changes according to the movement of the player's ship.
    - Andrej = movable single star system view centered on player

    - Lucie = scrollable map
- Even movement of the player's ship. Instead of an immediate jump, the player's ship flies to its indicated destination according to its engine.
    - Both = smooth animation of player ship in single star system view
- Top10-list. A Top10-list is implemented. The list maintains its contents over closing and restarting the game.
    - Andrej
- Own engine type and a ship that uses it.
    - Lucie = Planet + PlanetCoreDrive
- Integrating the student team's unit tests as a part of the CI environment. The tests are run as a part of CI.
    - Andrej = added rest of the tests
    - Lucie = added configuration file with statistics test

Own Extra:

- "ZOOM" feature of single star system
    - Andrej = base implementation of scene, player ship movement
    - Lucie = managing coordinates, positioning and spawning items on scene