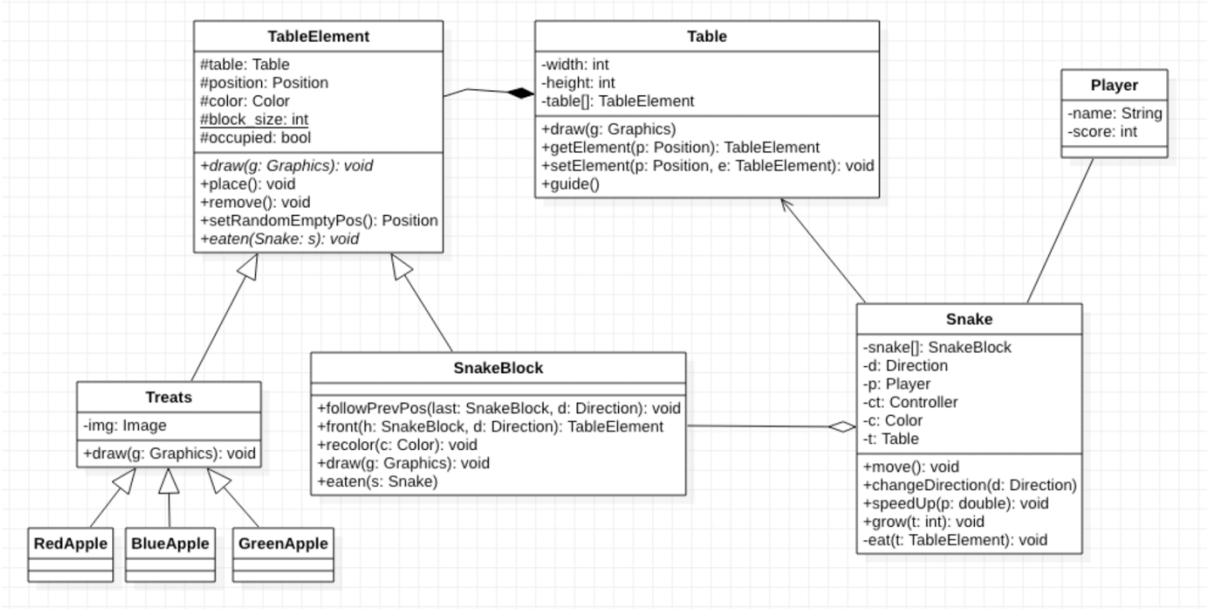
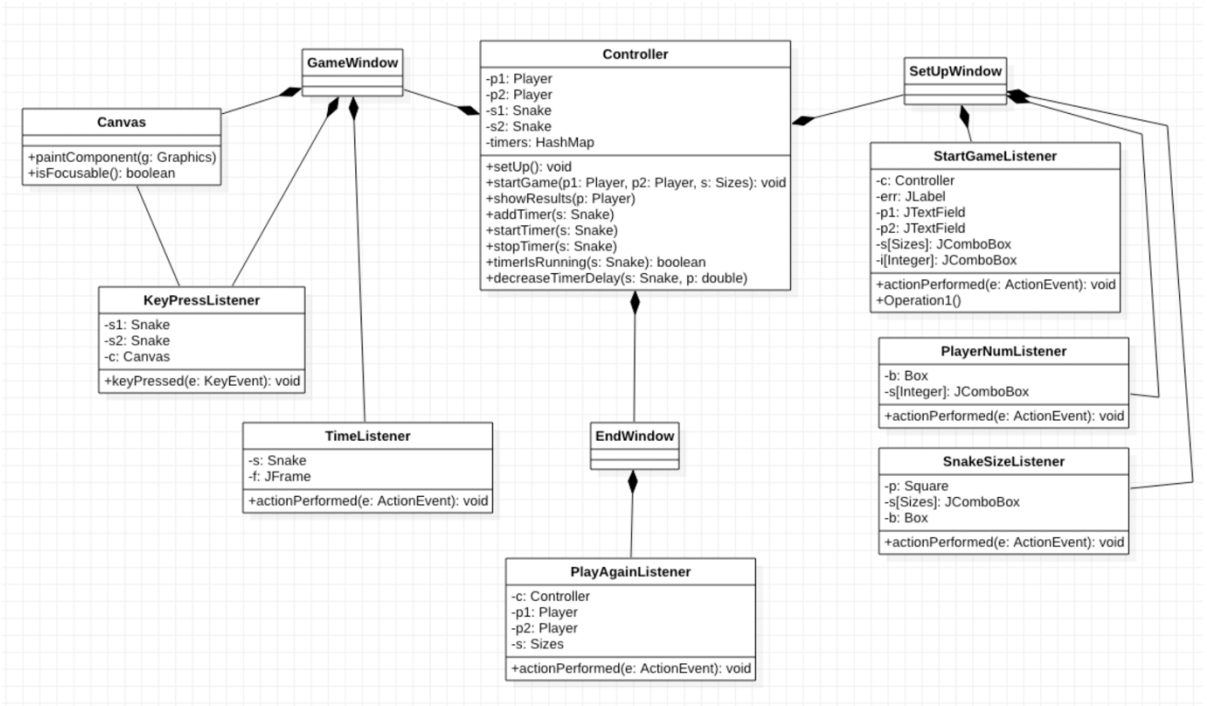
Architectural design

UML diagrams

Model: *gamemotor* package



Controller, view: *gui* package



Note: getters/setters are not displayed for brevity.

Design aspects, comments

The program was created according to OOP principles, in Model-View-Controller resolution.

Game Motor

- The snake moves to its next position on an event generated by a timer, the game is called OView-Control. The start of the game and the movement of the snake are controlled by button press events. Such events are are monitored by various EventHandlers. Here we observe separately that the snake cannot turn in the opposite direction at once, and 2 user eventsbetween two events (antighosting).
- The fields of the table (as a 2D array) are stored as a heterogeneous collection of table elements [empty, snake, apple]
- Individual table elements communicate by calling (partly virtual) methods, e.g.: a when drawing, the table calls the virtual draw() method on all its elements, and they themselves draw themselves, or on a step, or when two elements collide, etc..
- The snake is stored as a sequence of blocks in a linked list (LinkedList), movement is procedure as follows:
- Calculate the position where the head would be at the next moment, step there, then check if there is any other element in the table.
    - If it is empty, we enter a snake element in the table at that position, add the snake head to the list, and delete the end element.
    - If there is an apple in that position, then after appending the head, we do not delete the last snake element (thus increasing its length) and generate a new apple.
    - But if you hit a snake or a poison apple, the game is over.
- We scan the ranklist.json values line by line and store them in a list. We update the list:
    - if the winner has a better score than someone already in the list, we enter
    - if there is none, but the list is not complete (10 elements), we add it to the end
    - delete the last element of the list if > 10 finally, collapse the list into JSON.

Aspects of the GUI

Controller
Responds to input from the user interface and controls the game accordingly and it switches between menus.
SetUpWindow
Displays the user interface of the Setup menu. Adds the appropriate ActionListeners (SnakeSizeListener: shows which value has which snake size, PlayerNumListener: by number of players, displays the corresponding number of JTextFields, StartGameListener: validates and saves the data set so far, sets the next state of the controller).
GameWindow
Displays the user interface of the Game menu, plots the table elements. Adds the appropriate ActionListeners (TimerListener: moves the snake on time ticks, KeyPressListener:

changes the direction of the snake on corresponding key presses.)

EndGameWindow

Displays the End Game menu user interface, performs file operations (serialization), displays the ranking and results.