Bitcode Assignment 3

1 Design Patterns

Using design patterns in software project is a good practice. It helps to make your software understandable, sustainable and expendable. We have chosen two design patterns and implemented them in our existing code, the state pattern and the strategy pattern.

1.1 The State Design Pattern

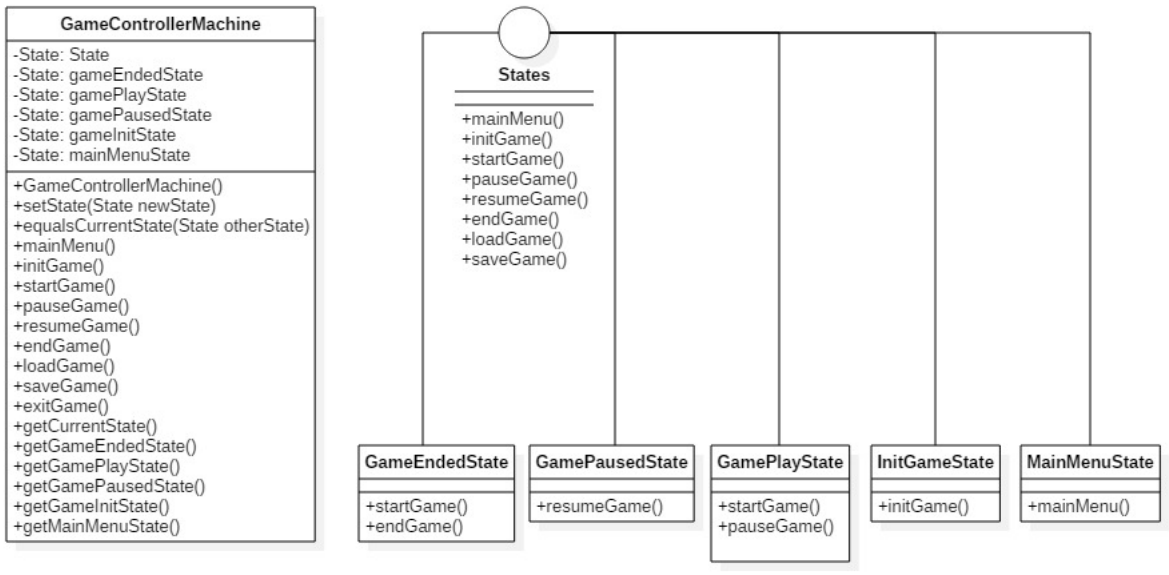
We chose the state design pattern for the game because the game has become very complex and various objects can and will change a lot of the games behavior and variables.

With the state design pattern we are able to dictate when certain parts of the game's logic can be executed and when not. This reduces the chances of accidentally changing the games behavior or variables when it is not wanted.

For example disabling mouse input when the user is in the main menu state makes no sense. Without the state design pattern the disable mouse input (pause game) could be called when it is unwanted. By integrating the pause and resume,into the state design pattern we can choose in which states it is allowed to execute this command. We obviously only want this to happen when the game is running. Otherwhise we throw an exception, allowing easy traceability to where the command was invalidly called, allowing for an easy fix without having to debug the whole program.

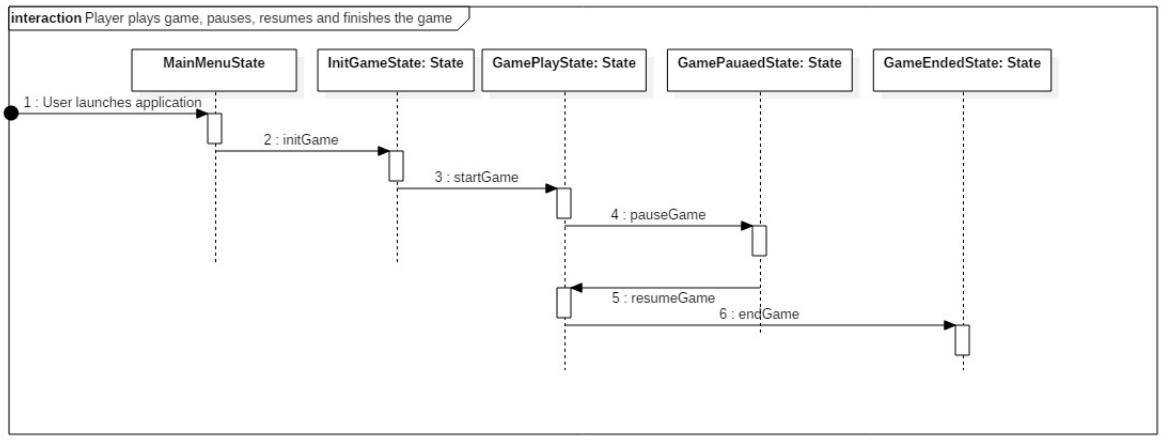
1.1.1 State Class Diagram

The class diagram of the implemented state design mattern is shown in the figure below.



1.1.2 State Sequence Diagram

In the figure below the sequence diagram of the implemented state deign pattern is shown.

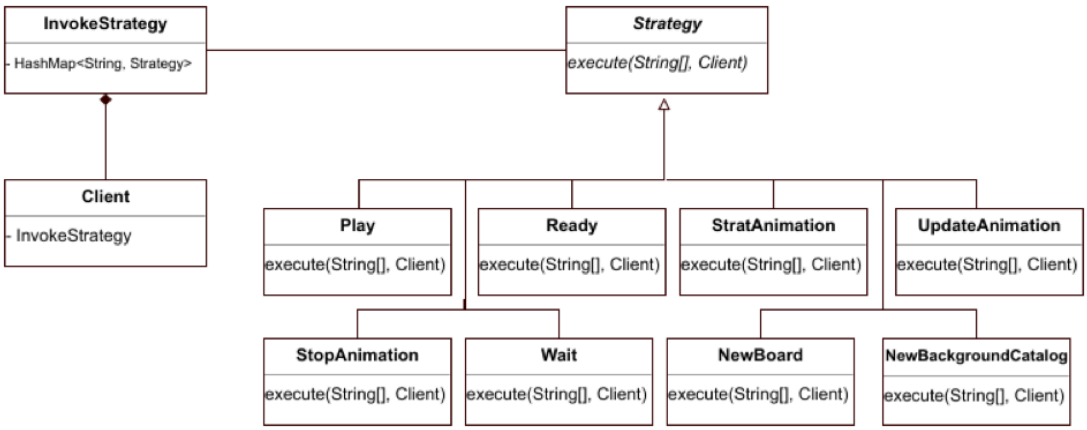


1.2 The Strategy Design Pattern

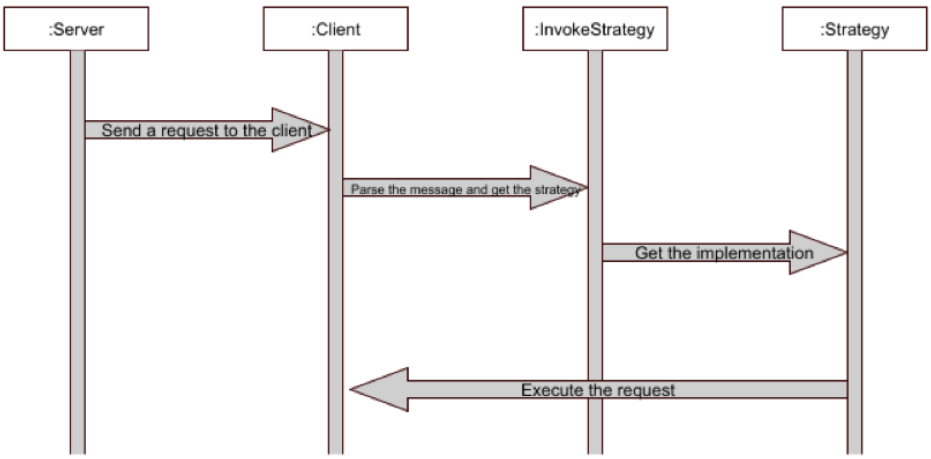
The multi-player feature of our game requires handling multiple requests from server. This is the responsibility of the client object to listen to the server and receive these requests and parse them. Based on each request it requires doing an action but there are multiple ways of performing that action and that is why Strategy design pattern comes into play because it helps to encapsulate each request in a separate class and define each class with the same interface so they can be interchangeable.

1.2.1 Strategy Class Diagram

There is a general interface called Strategy with one method which accepts two arguments. These arguments will be treated differently based on the implantation. For each request from the server we create a separate class which implements the Strategy interface. And also there is another class called invokeStrategy which is used to map each request to the appropriate implementation. This class has also a method to add a strategy dynamically, so whenever we want to extend our server functionality we don't need to change the base code, we need only to implement a class for handling the request and add it to the invokeStrategy class.



1.2.2 Strategy Sequence Diagram



2 Software Metrics

keeping track of software metrics helps to keep code clean, understandable, sustainable and expendable. We have chosen to use a intellij plugin called "MetricsReloaded" that helps of measuring software metrics. By running this tool we could compile the following lists of classes and methods that could be improved.

Classes

1. StandardItemFactory, WMC: 28
2. MainScreen, DIT: 5
3. Board, WMC: 40
4. Game, WMC: 28
5. BackgroundTileCatalog, WMC: 26
6. ConnectionScreen, DIT: 6
7. JPanelTile, DIT: 5
8. Client, CBO: 23
9. Game, CBO: 21
10. MainScreen, CBO: 20

Methods

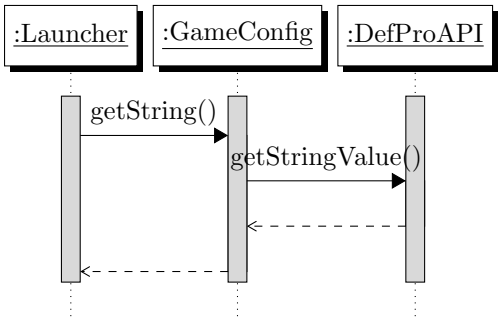
1. pooralien.Controller.Board.removeGroups, ECC: 6
2. HighScoreTable.HighScoreEnterNameDialog.HighScoreEnterNameDialog, ECC: 4
3. Controller.HighScore.TopXTableModel.getValueAt, ECC: 8
4. pooralien.Controller.BackgroundTileCatalog.BackgroundTileCatalog, ECC: 4
5. pooralien.Controller.BackgroundTile.BackgroundTile, ECC: 4
6. pooralien.Controller.BackgroundTile.equals, ECC:
7. item.StandardItemFactory.createItem, ECC: 8
8. item.StandardItemFactory.createRandomItem, ECC: 8

For each improvable class or method a issue is created on our github repository so that everyone can help improve the code.

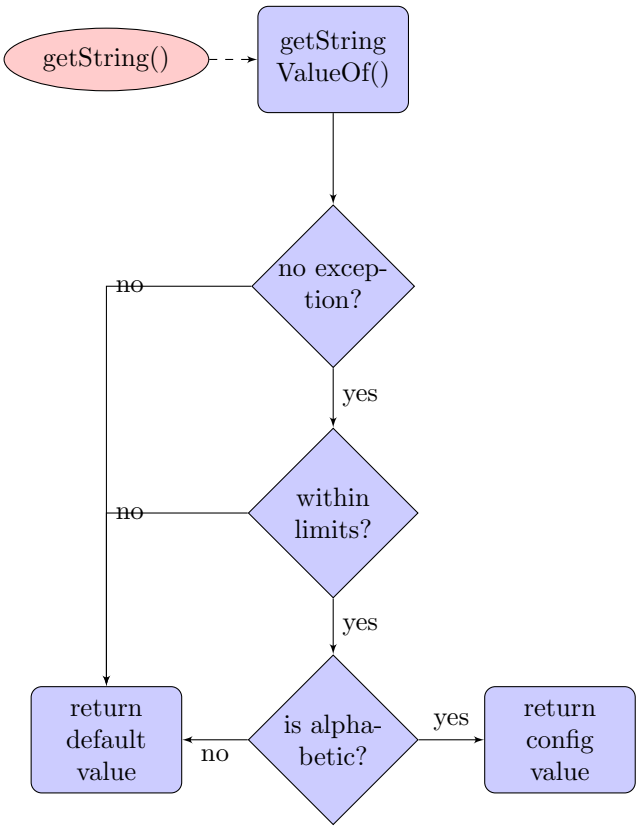
3 Defensive Programming

In the previous assignments it was required to use a game configuration file for initializing variables using a provided library. However, the provided library introduced bugs in the game.

To solve the bugs we have created a wrapper class (CameConfig) around the library's API that catches all the bugs and checks if the variables are within limits. This means that for every different API call we created a method in the GameConfig class. In the sequence diagram below is shown how the wrapper class works.



In every method in the CameConfig class that implements the API checks are build in to verify the data that is returned by the API. Also if the API throws an exception it is catched in the method. If an exception occurs or the data returned by the API is not within the defined boundaries the method will return the defined default value. The flowchart below shows how the method getString() is implemented.



In this particular example the following conditions makes the method return the defined default value.

- If the library thorows an exception.
- If the amount of characters in the sting is less or more then expected.
- If the string is not alphabetic.

4 Selectable Difficulty

One way of making the game more competitive is to implement selectable difficulty. The idea is that the user can select a desired difficultly between easy, normal and hard before the game starts. The selected level of difficulty will noticeably influence the difficulty of the game.

4.1 Requirements

The requirements are ordered by the MoSCoW model.

Must Have:

- The game must have three levels of difficulty, easy, normal and hard.
- The player must be able to select the desired difficulty befor the game starts.
- The game must have a start screen.

Should Have:

- The player should get more points when removing a tile or a background tile when the game is more difficult.

Could Have:

- Higher difficulty could mean that the player should find four of the same tiles on a row.
- Higher difficulty could mean that the player has less moves to remove all background tiles.
- Higher difficulty could mean that there are more background tiles.
- Lower difficulty could mean six different tiles instead of 7 tiles.

4.2 Software Design

The figure below shows the class diagram of the implementation.