Bitcode Assignment 2

1 Design Patterns

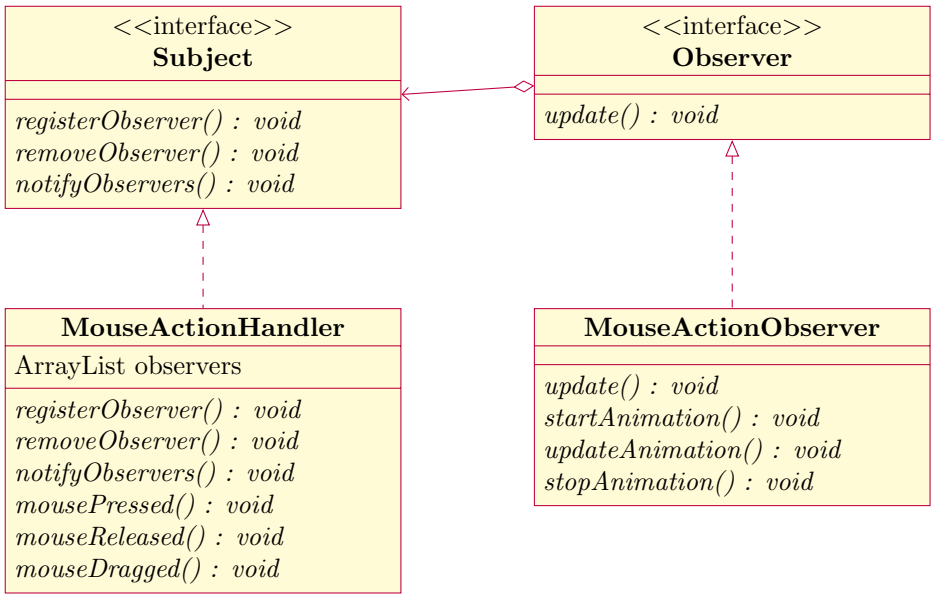
Using design patterns in software project is a good practice. It helps to make your software understandable, sustainable and expendable. We have chosen two design patterns and implemented them in our existing code, the observer pattern and the factory pattern.

1.1 The Observer Design Pattern

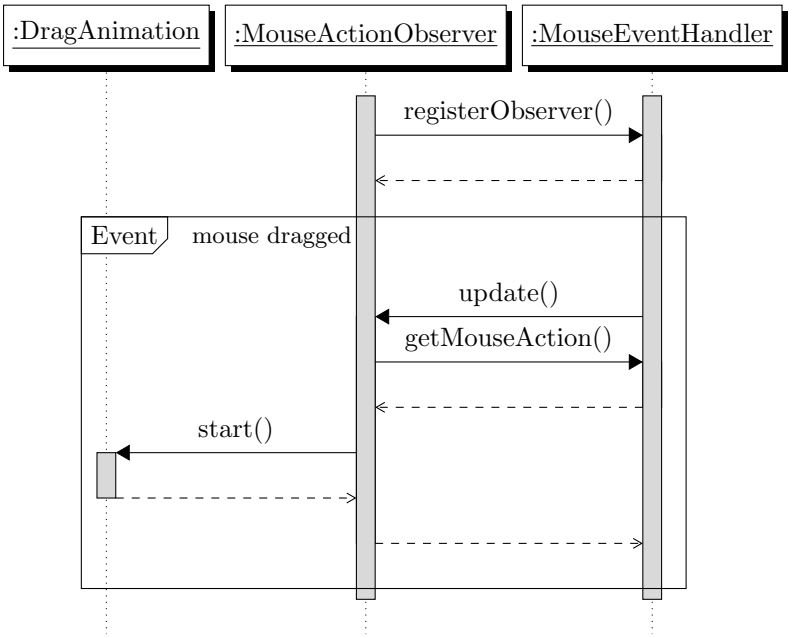
The observer pattern is implemented between the MouseActionHandler class (subject) and the DragAnimation class (observer). The MouseActionHandler listens for mouse input in a given window. A object of the DragAnimation class is created when a mouse drag is registered and updated when the position of the mouse pointer changes.

We have chosen this pattern on this location because it enables us to create more observers for the MouseActionHandler in the future. Therefore it will save time to implement new features that need the MouseActionHandler class.

1.1.1 Observer Class Diagram



1.1.2 Observer Sequence Diagram



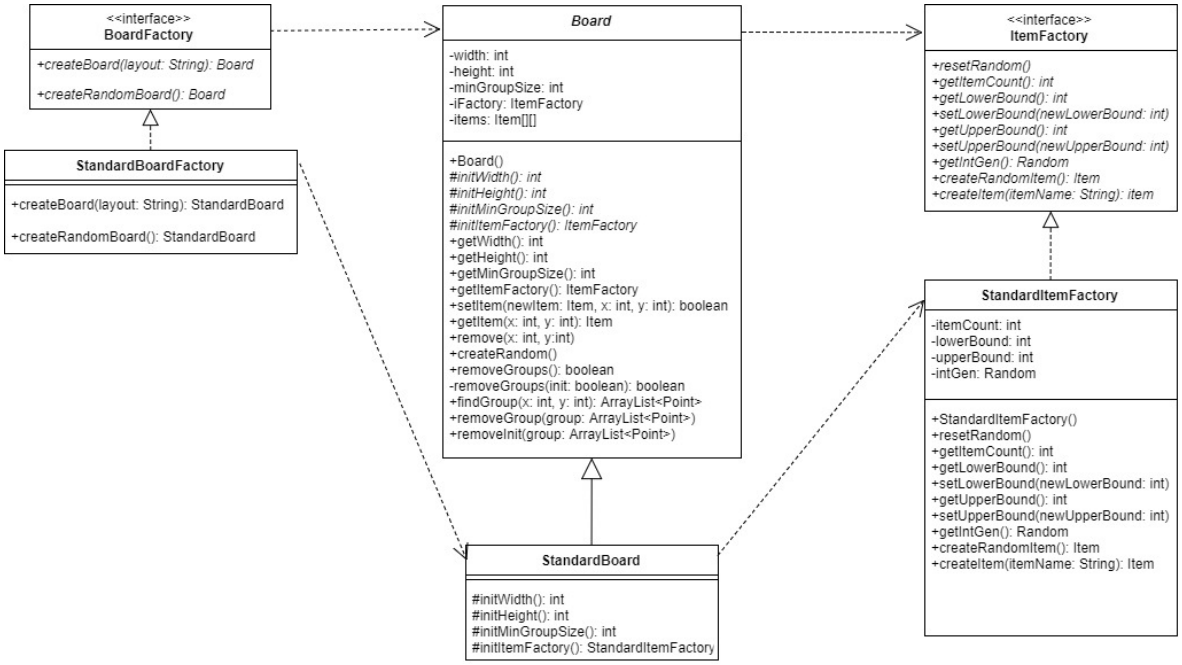
1.2 The Factory Design Pattern

The factory design patterns are implemented for the BoardFactory and the ItemFactory. On launch the board is created via the StandardBoardFactory which will construct a board using the StandardItemFactory.

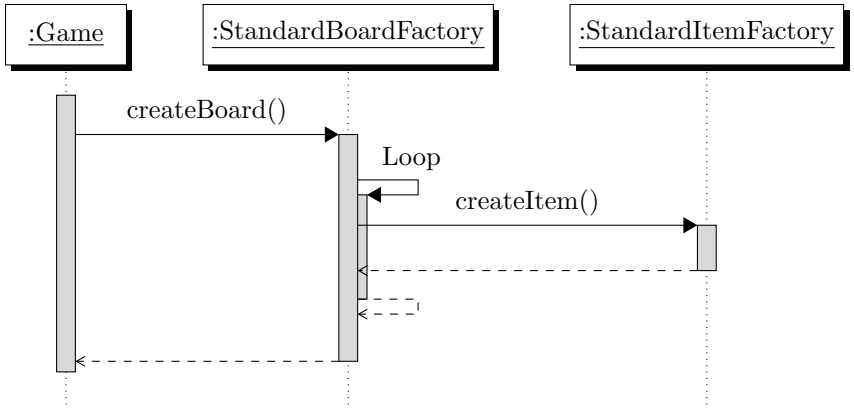
The reason that we have implemented a factory pattern for the Board class is that we can use this pattern in the future for creating different types of board. For instance creating boards with different sizes based on difficulty.

We have implemented the factory pattern for the Item class because that allows us to create different types of items in the board, such as special items that will remove all items from the board.

1.2.1 Factory Class Diagram



1.2.2 Factory Sequence Diagram



2 Your wish is my command

Our client wanted to have two new features implemented, which consist of a high score page and maximum amount of moves per level. For each each feature we have created new requirements and a software design.

2.1 High Score Page

tbd

2.1.1 Requirements

tbd

2.1.2 Software Design

tbd use UML

2.2 Maximum Moves per Level

Adding a constraint on the amount of moves in each level makes the game more immersive. The player needs to clear all items with a square background around them within a given number of moves. If the player manages to clear all these items within the maximum amount of moves the game will continue until the maximum amount of move are reached. When the player passes the amount of moves the player and the player did not manage to clear all background items the game will end. When the player passes the amount of moves and the player was able to clear all background items, the game will generate a new board and the amount of moves are reset to the default value.

2.2.1 Requirements

The requirement that are sorted using the MoSCoW model.

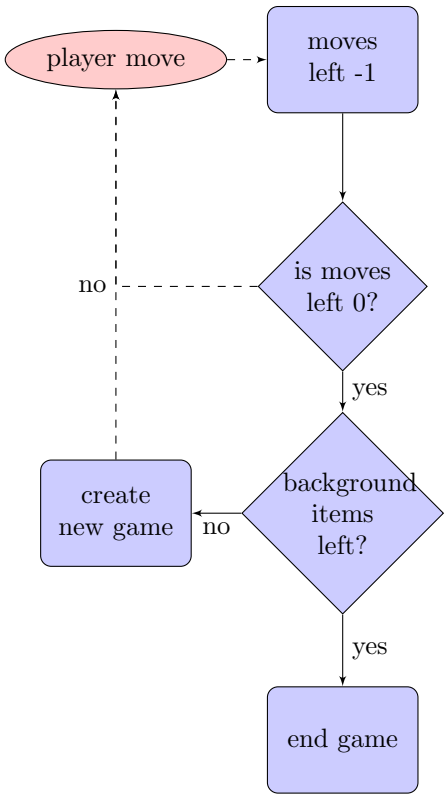
Must Have

- There must be a maximum amount of moves per level.
- The game must end when the maximum amount of moves are reached and not all backgound items are cleared.
- The game must continue when the maximum amount of moves are reached and all backgound items are cleared.
- When the game continues after the maximum amount of moves are reached the amount of moves left must be reset to the default value.

Should Have

- The game should generate a new board when the game continues after the maximum amount of moves are reached.
- The default amount of moves should be configurable from the configuration file.
- The player should be able to see the amount of moves left.
- The score should not be reset once the game continues after the maximum amount of moves are reached.

2.2.2 Software Design



3 Turn-based Multiplayer

In exercise three we have been asked to implement a feature that we wanted to implement. We have chosen to implement turn-based Multiplayer. For this feature we also created new requirements and a software design.

3.1 Requirements

The requirement that are sorted using the MoSCoW model.

Must Have

- There must exists a central server to handle communication between players.
- There must be a open port that listening to the player who wants to connect to the network.
- A player must be able to connect to a server using an address.
- It must be possible to send data to all or some of the player via the server.
- When a player move something on the screen all other players must be able to see it live on their screen and sync everything.
- There must exists a communication protocol to exchange data between players (via server).
- The server must decide who's turn it is to play the game.
- When the player connects to the network their game must be put in a pause mode waiting for the server to initiate the game.
- The players must be able to see on the screen if they should play the game or wait for other players.

Should Have

- The player should see the errors graphically instead of on terminal errors.
- A Command design pattern should be implemented to handle the communication protocol.
- The server should announce the winner.

Could Have

- The player could send their names and see the score of each other on their screen.

3.2 Software Design