Assignment 3

Group 66, Bust-a-move

Exercise 1 - Design patterns (30 pts)

Choose two design patterns among those that we saw in class and that you did not use in a previous assignment. For each chosen design pattern, you must have a corresponding implementation in your code. If not, refactor your code to include it.

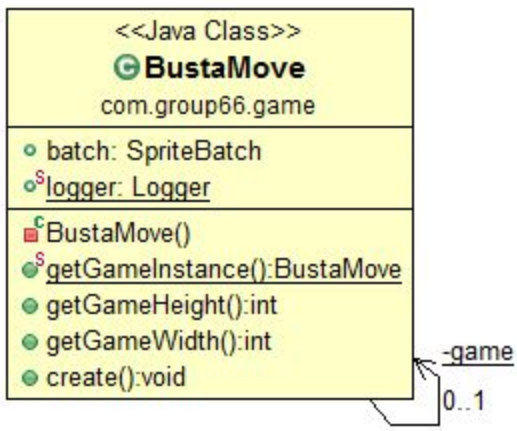
Design pattern 3: Singleton

1. Write a natural language description of why and how the pattern is implemented in your code.

This game is build around the libGDX Game Development Framework. Many operations, especially rendering batches, are methods of the BustaMove class that extends the Game class of libGDX. With different Screens available, each screen has to render its own components, and each screen does this using its own instance of the BustaMove class, which shouldn't be the case; as this is one game, it'd be more reasonable to have one instance of the BustaMove game that's used by each screen to render. This encouraged applying the Singleton design pattern to the BustaMove class. With this, only one instance of BustaMove is created when the game starts (eagerly created in a static initializer that's thread safe, following slide 11 of DPII lecture) and it can be get from any part of the game using the static getGameInstance() method.

2. Make a class diagram of how the pattern is structured statically in your code.

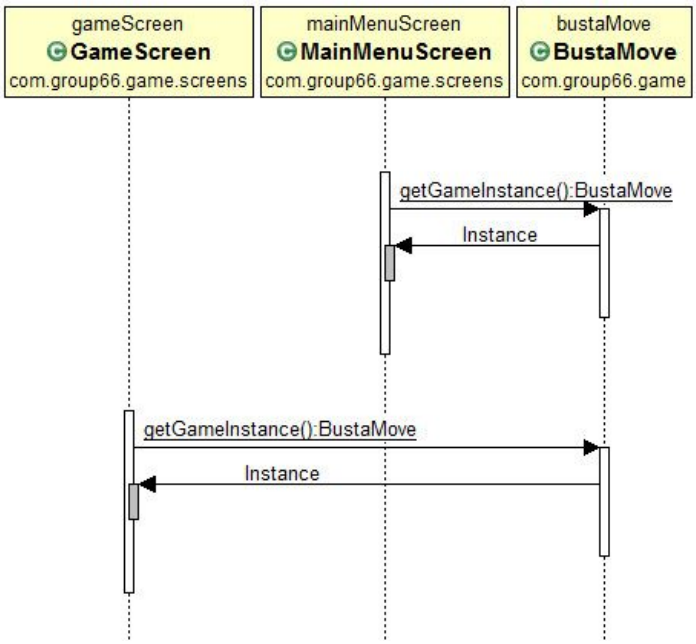
Again, the only instance was eagerly created in a static initializer that's thread safe, following slide 11 of DPII lecture.



3. Make a sequence diagram of how the pattern works dynamically in your code.

This sequence diagram shows an example of two classes using the getGameInstance() method from the BustaMove class, which have been refactored to use the Singleton design

pattern. In this, the Screen classes use the getGameInstance() methods to get the instance of the game and eventually use it to start related operations, like rendering.



Design pattern 4: Iterator

1. Write a natural language description of why and how the pattern is implemented in your code.

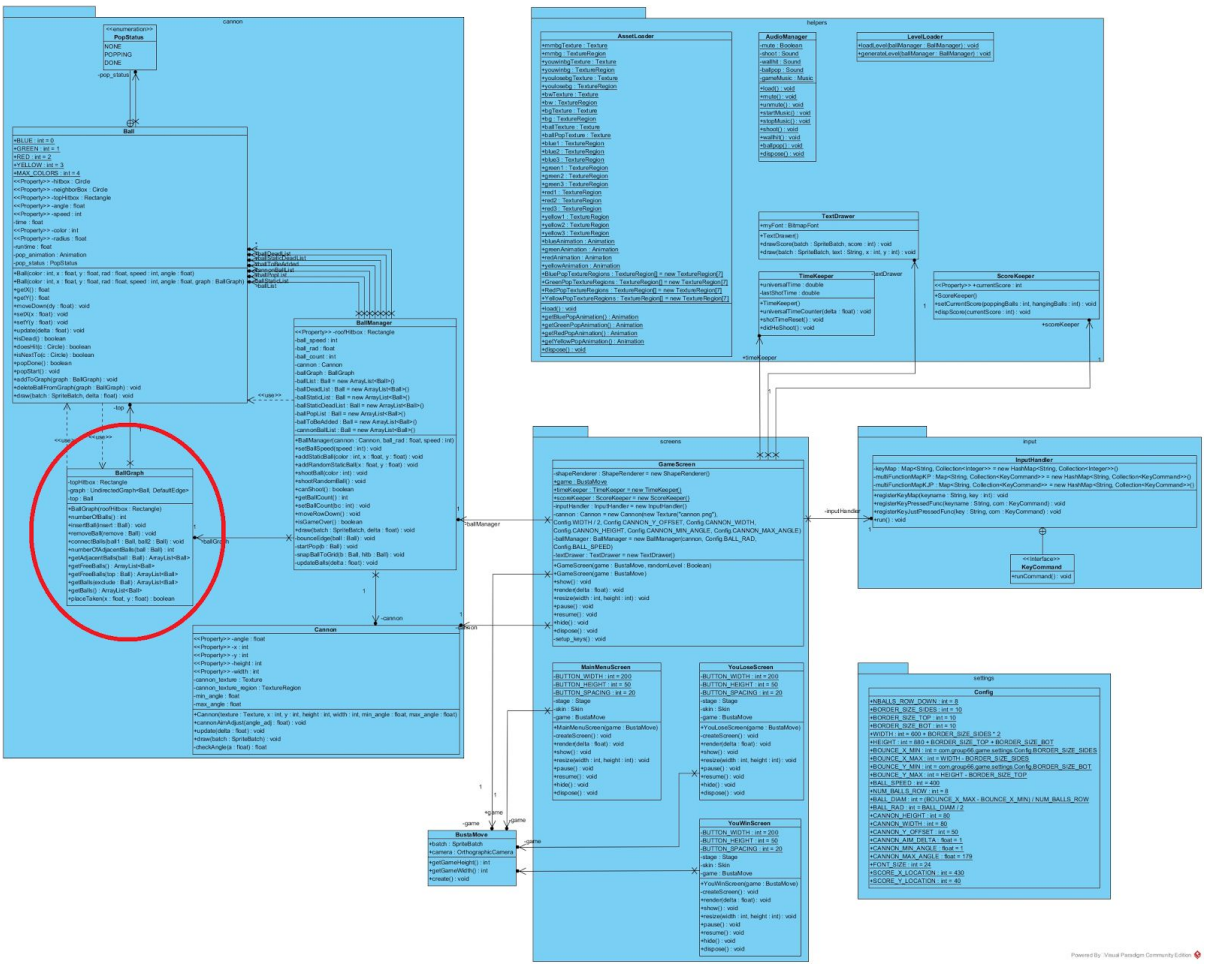
The BallGraph class stores all the balls of the game in an undirected graph as vertices. If the balls are next to each other in the grid, an edge will be created between the two vertexes. The BallGraph class had a complex algorithm to determine which balls of the same color are connected, so it can be determined if they should pop.

This week the iterator design pattern is applied to the BallGraph, two different iterators are implemented. One iterator is a breadth first iterator to which you can give a condition, so you can get for example all connected red balls. The other iterator iterates over the balls adjacent to a certain ball. The first iterator sometimes uses the adjacent iterator.

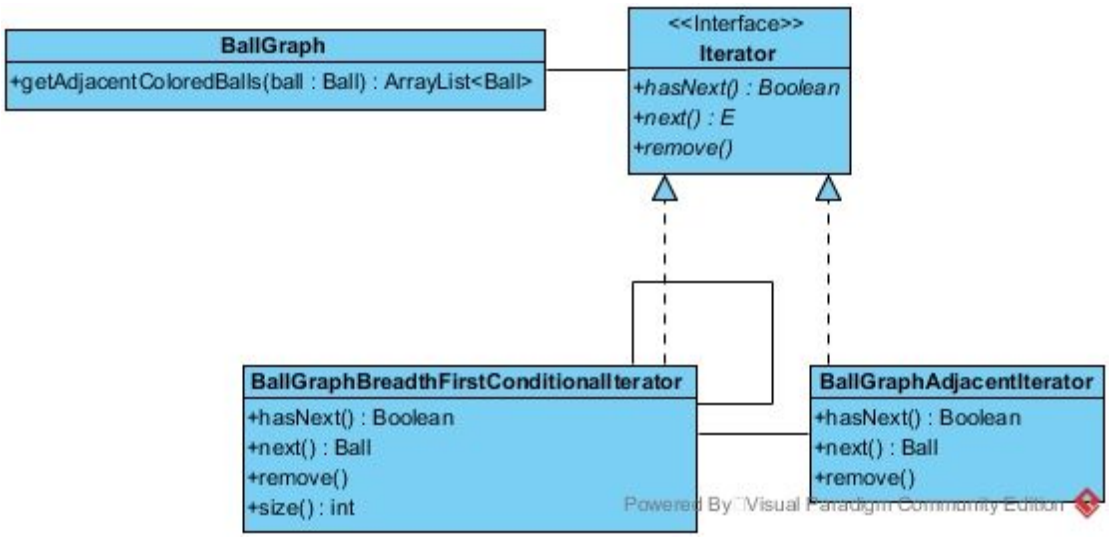
By using the iterator pattern the BallGraph class becomes less complex and more readable.

2. Make a class diagram of how the pattern is structured statically in your code.

The diagram below shows an overview of all the classes in our game. It is quite big, but you can zoom in. The place where the design pattern in implemented is at the red circle.

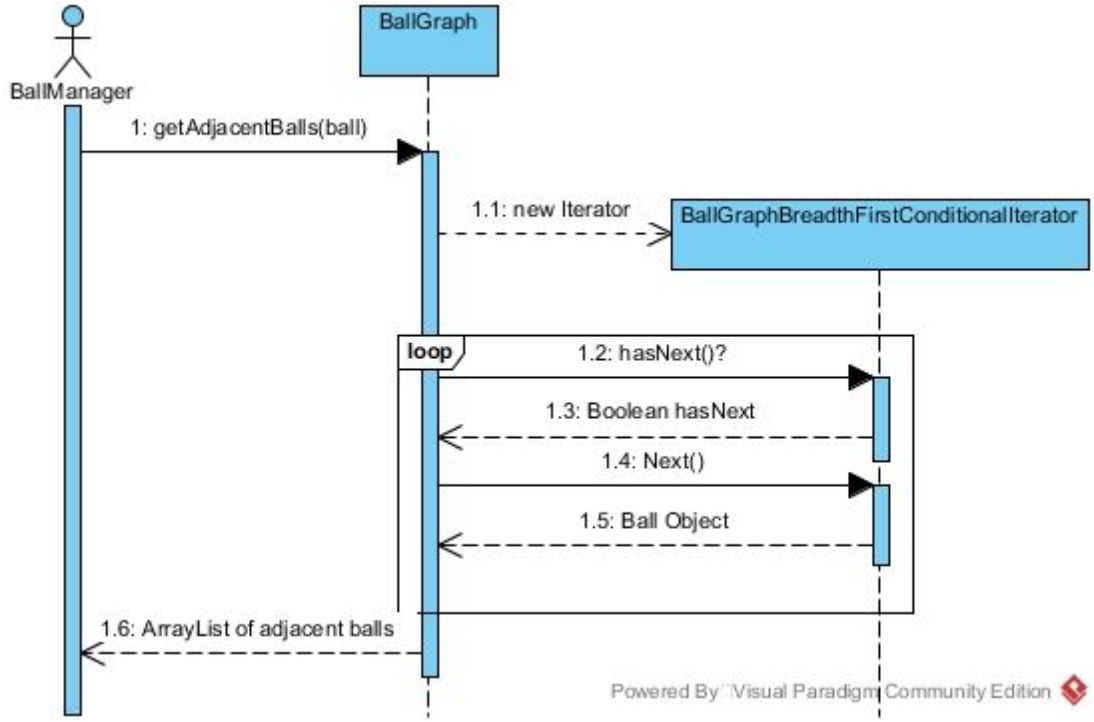


A detailed UML of the implemented design pattern is given below. Two iterators have been implemented which follow the interface of Iterator. The BallGraph class is able to use them by only knowing how the Iterator interface looks like.

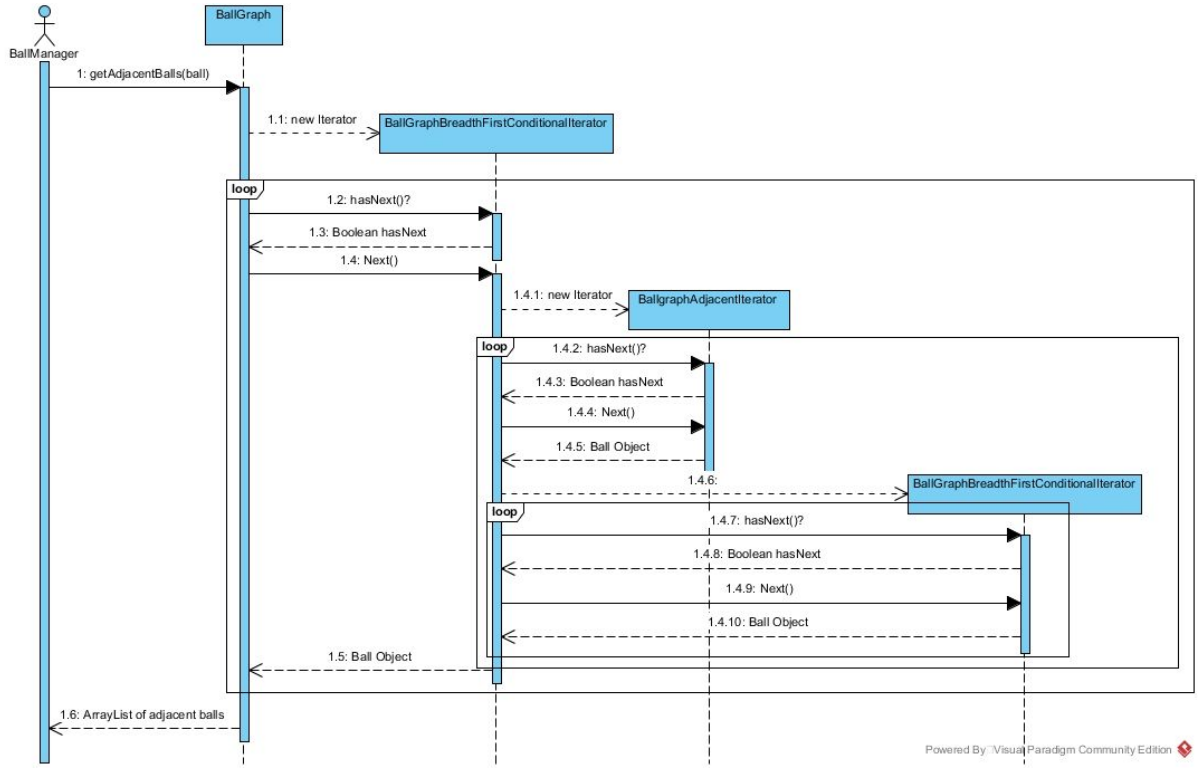


3. Make a sequence diagram of how the pattern works dynamically in your code.
There are two possibilities either the iterator is used for getting the connected balls of a colored ball or it is used to get the connected balls of a special ball.

The first situation is shown below. The ballmanager will ask the ballGraph to give a list of the adjacent balls. The ballgraph then creates an iterator and as long as there are balls, they will be added to the list. After that the ballgraph will return the list of balls.



The second situation will happen if the ballmanager will ask for a list of adjacentballs of a special ball. It will first get all connected special balls. After that is will get all the adjacent balls of each special ball. For each adjacent ball all connected balls are retrieved using a new breadth first conditional iterator. Eventually all the balls in a way connected to the special ball will be added to the list, which is in the end returned to the ball manager.



Exercise 2 - Your wish is my command (30 pts)

1. Each project and group is heading to a specific direction and there are specific things that should be implemented and improved. Who knows it better than your TA?
In this exercise, you have to talk to your TA (during the group meeting on Monday) and (s)he will give you what to do next to your game.
After you receive the task from your TA, write a requirements document, which will be evaluated in the same way as for the requirements document of the initial version.
Afterwards you must implement the requirements.

Your wish is my command related requirements can be found within the assignment 3 Your wish is my command directory.

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced).

Your wish is my command related requirements can be found within the assignment 3 Your wish is my command directory.

Exercise 3 - 20-Time (30 pts)

1. Google asks its employees to spend 20% of their time at Google to a project that their job description does not cover. As a result of the 20% Project at Google, we now have Gmail, AdSense, and Google News, among the others.
This is your occasion to have similar freedom. You can decide what to do next to your game: It can be an extension/improvement from any perspective, such as improved code quality, or novel features.
Define your own requirements and get them approved by your teaching assistant. Afterwards you must implement the requirements.

20-Time related requirements can be found within the assignment 3 20-Time directory.

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced).

20-Time related requirements can be found within the assignment 3 20-Time directory.