Assignment 3
Bubble Spinner Group 20

Contents

1. Design Patterns

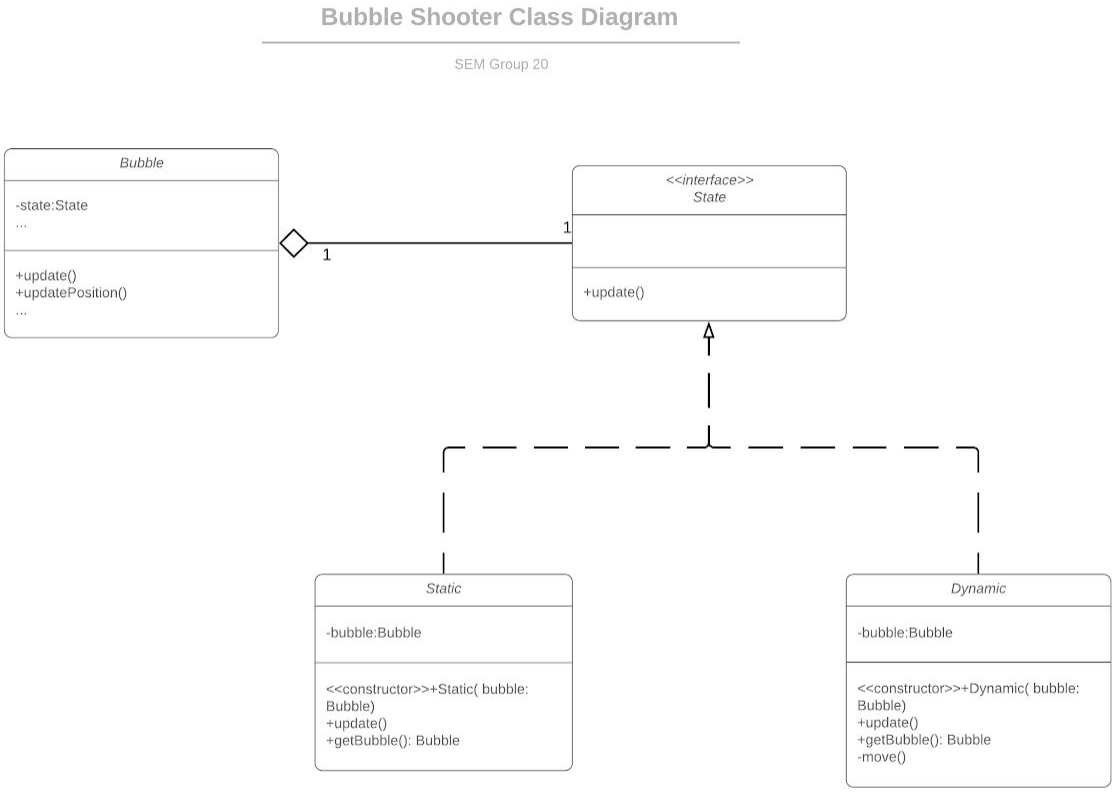
1.1 State

"The State Pattern allows an object to alter its behavior when its internal state changes.".[1]

The behaviour of Bubble objects in our game depends on their state, which can change at runtime. A Bubble instance shown at the bottom of the screen is static at first, but will then start moving after it has been shot by the player. After colliding with the BubbleBoard, it will become static again.

To integrate this pattern, a State interface has been created which defines the update() method. The classes Static and Dynamic both implement the State interface and provide different implementations to the update() method. Dynamic moves a Bubble according to its direction, whereas Static keeps the Bubble fixed. Both classes keep a reference to the Bubble instance they affect. The Bubble class has a State attribute, which is set to Static by the constructor. It is set to Dynamic when a Bubble is shot by the BubbleFactory. The PlayScreen sets it back to Static on collision with the BubbleBoard.

The implementation of the State Pattern allows the development team to create additional states without having to change the Bubble class. It also prevents large, conditional statements to arise when extra states are implemented.



[1] dr. Annibale Panichella, TU Delft, Software Engineering Methods lecture 7, slide 31

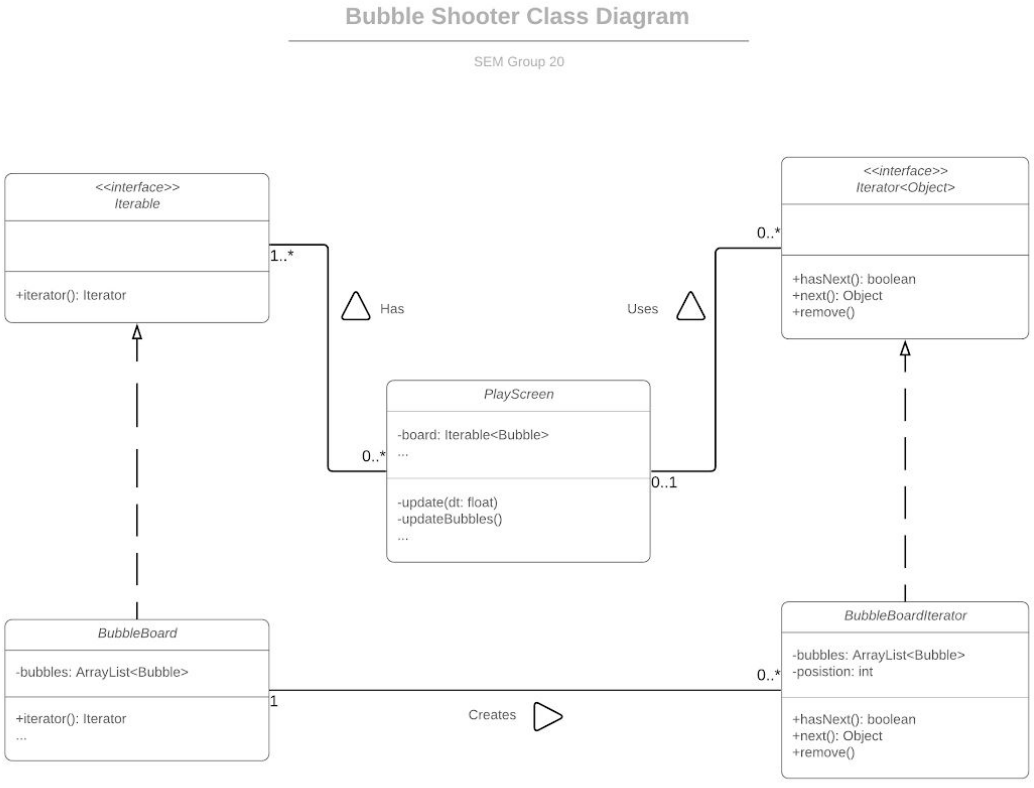
1.2 Iterator

"The Iterator Pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation."[2]

The PlayScreen in our Bubble Shooter game has a BubbleBoard, which contains Bubble objects. In order to update the screen, the PlayScreen needs to iterate through the Bubbles in the BubbleBoard upon execution of the method updateBubbles(). However, we do not want to expose the underlying representation of the board. In previous versions of the game, the Playscreen kept its own duplicate ArrayList of Bubbles. This solution was not optimal, since it wasted space.

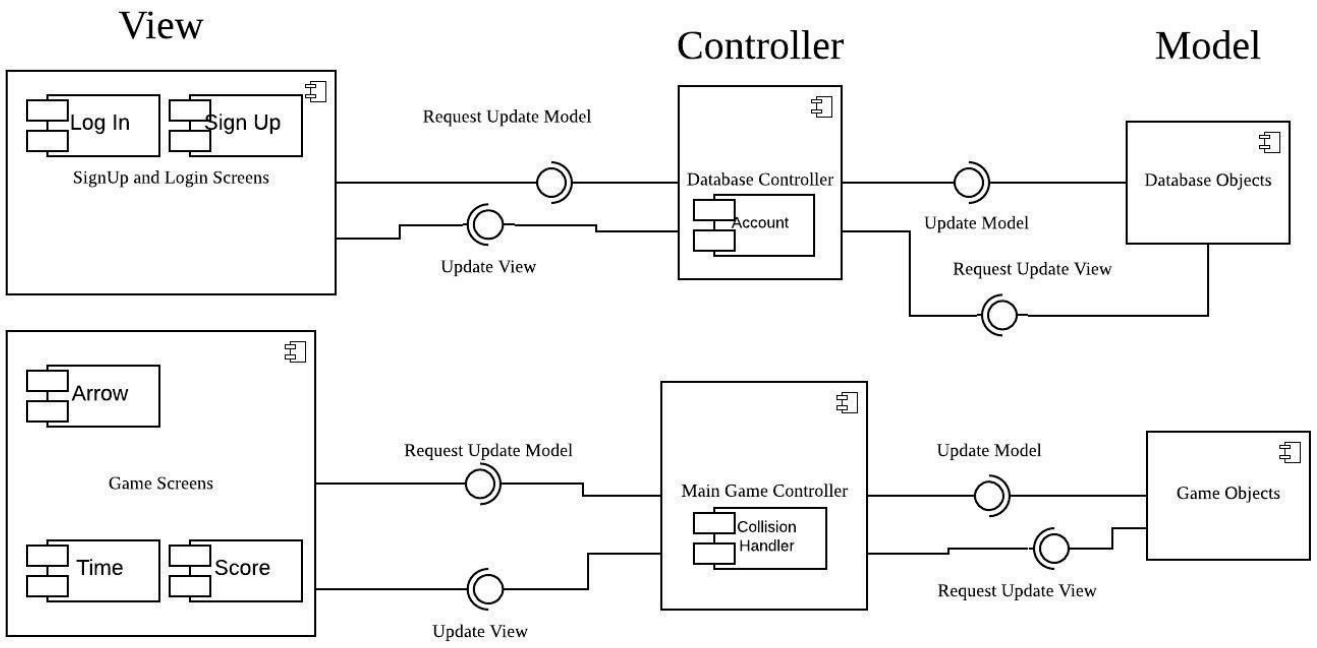
To hide the underlying data structures of the BubbleBoard, a BubbleBoardIterator has been created. This class implements the Iterator interface provided by Java. The concrete Iterator can go over the ArrayList of Bubbles that a BubbleBoard has, without giving other classes direct access to this ArrayList. Now the PlayScreen does not need to keep its own ArrayList of Bubbles anymore.

If other aggregate classes (like different types of boards) were to be created in future versions of the game, they should also be accompanied by custom concrete iterators. Those aggregate classes should implement the Iterable interface, which forces the iterator() method to be implemented that returns an Iterator of that class. The custom iterators should implement the Iterator interface and provide the methods hasNext(), next() and remove(). Also, if the BubbleBoard were to be replaced by another iterable board (in other levels of the game), then the way PlayScreen iterates over the board would not have to be changed.



[2] dr. Annibale Panichella, TU Delft, Software Engineering Methods lecture 6, slide 55

2. Software Architecture



We used the Model-View-Controller software architecture to develop our Bubble Spinner game. We divided the related game logic into three interconnected layers: model, view, controller. Therefore a player can use the controller, which manipulates the model, and the view would get updated, which can be seen by the player. This architecture groups related actions in one controller, and related views in one model, which gives a high cohesion among classes overall. This architecture also allows our group members to develop different components at the same time due to the loose coupling between classes.

2.1 View

The view contains two components: SignUp and Login Screens, and Game Screens

1. **SignUp and Login Screens:** contains classes 1) SignUpScreen; 2) LogInScreen; 3) MainScreen; 4) userDaoController

   This component is to allow users to sign up for a new account and log in with their existing account.

   The MainScreen is the screen you see when you first start up the game. The SignUpScreen and LogInScreen take the players username and password, which are then given to the database controller for authentication. From here you can either go to the sign up screen or the login screen. Although UserDaoController is a gui class for notifying the users about their status of signing up and logging-in, and communicates with the controller UserDao.

1. **Game Screens:** contains classes 1) StartGameScreen; 2) BubbleSpinner 3) Hud 4) Arrow

The component contains the gui that a user would see when he/she starts playing the game after signing up or logging-in.

StartGameScreen is the screen that gets displayed after logging in successfully with the correct credentials. Hud takes care of showing the score, the time, displays the name of the game and communicates with the controller called Playscreen. The arrow class displays the arrow on the screen and takes in the mouse/touch input and passes it to the PlayScreen controller, which is responsible for shooting bubbles in the correct direction.

2.2 Controller

The controller contains two components: Database controller, Main game controller. Within the controllers, Database controller will use the scores calculated from the main game controller and main game controller needs the scores ranking stored in the database to display the ranks to the user when a game is won or lost. The implementations of score calculation and score storing in database are still under development, and therefore we did not specify the class names and subcomponents in the diagram.

1. **DataBase controller:** contains class: 1) UserDao

Database controller is the component responsible for handling users' requests for registering with a new account and login with an existing account.

UserDao is a controller class that controls everything database related. Since this class is a controller, it needs to communicate with both model and view classes. Therefore UserDao communicates with model classes that take care of modifying the database and creating an instance of a user as well as view classes that display error messages or retrieved content from querying the database.

2. **Main game controller:** contains classes: 1) CollisionHandler; 2) PlayScreen

This component has the main control over the interaction between the gui classes in the View and the game objects in the Model.

It keeps track of events and acts as the physics engine of the game. The CollisionHandler detects every collision that occurred and calls the correct methods in PlayScreen to handle that collision. There are two types of collisions possible: bubble X bubble and bubble X wall. PlayScreen is the brain that links all of the game components. It takes the user input from the arrow class and calls the bubble factory in order to shoot a bubble and to prepare the next bubble to be shot. Once a collision with the bubble board occurs, it calls the bubbleBoard class in order to rotate the bubble structure. It prevents a bubble from floating indefinitely by keeping track of its age and states.

2.3 Model

The model contains two components: Game objects , Database objects.

1. **Game objects:** contains classes: 1) Bubble; 2) BubbleBoard; 3) BubbleFactory; 4) Color; 5) State; 6) BoardIterator

This component responds to the commands from the controllers when a user starts playing the game.

The Bubble class defines a regular bubble in the game. The BubbleBoard class creates bubbles for the central bubble cluster in the screen when a new game or a new level of the game gets initialized. It also checks the neighboring bubbles with the same color and stores the clusters of same-color bubbles. BubbleFactory regulates how the shooting bubbles in the bottom of the screen get generated. The Color class is in charge of randomizing a color for each bubble in the screen. And currently, we have two states, dynamic and static classes, which implement the State interface. The BoardIterator class is for iterating each bubble inside the bubble board. It retrieves each bubble from the BubbleBoard and give it to the PlayScreen, while hiding the internal data structure how BubbleBoard stores the bubbles.

2. **Database objects:** contains classes: 1) DatabaseConnection; 2) DatabaseProperties; 3) LogInRequest; 4) SignUpRequest; 5) UserRequest

This component takes care of the commands from the database controller whenever we need to query the database for information.

DatabaseConnection allows the system to query the database and retrieve the data needed. DatabaseProperties provides the credentials needed to connect to the database. UserRequest is an abstract class that creates an instance of a user, while SignUpRequest and LogInRequest extend from that class.

3. Coding

While working on the first exercise of this assignment, we implemented two design patterns. In addition to that, new buttons have been added to the game. The user is able to click a button which leads to the game manual. Pausing an ongoing game and resuming a paused one are also possible now. On top of that, the player can choose to restart a game. Another added feature is the possibility to resize the screen.