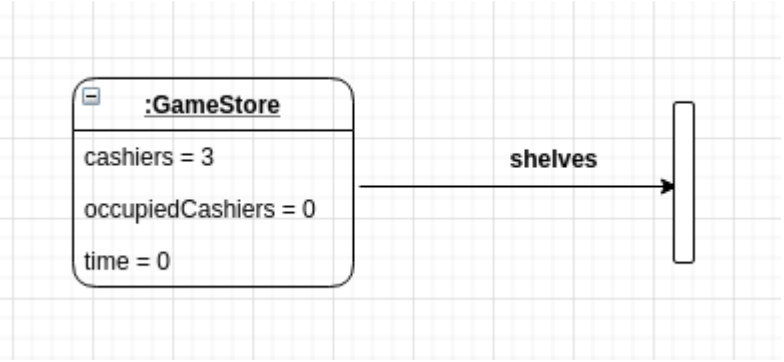
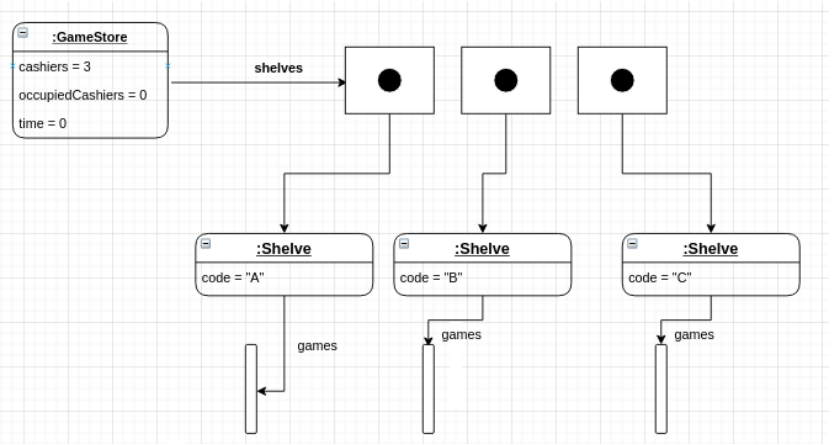
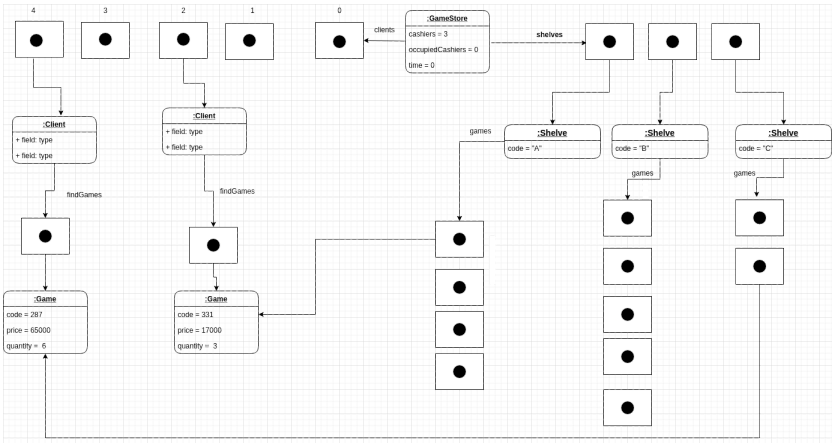
UNIT TEST DESIGNS

Name	Class	Stage
setupScenario1	GameStore	

Name	Class	Stage
setupScenario2	GameStore	

Name	Class	Stage
setupScenario3	GameStore	

Test goal: Validade the correct process of adding new shelves (3) to the game store				
Class	**Method**	**Stage**	**Input values**	**Result**

GameStore	addShelve	setupScenario1	sh1 = "A";
			sh2 = "B";
			sh3 = "C";
			
			qSh1 = 4;
			qSh2 = 5;
			qSh3 = 2;
				The list “shelves” of the class GameStore has three new elements shelves whose attributes are the same as the input ones, proving the correct addition of the objects.

Test goal: Validade the correct process of adding new games (11) to the game store and to its respective shelve. For this test the input values will be given as a console input but they are saved in a text file for technical reasons of this automatic test				
Class	**Method**	**Stage**	**Input values**	**Result**
Game Store	shelveAddGame	setupScenario2	4 331 17000 3 465 60000 6 612 80000 2 971 70000 6 5 441 30000 3 112 22000 6 229 28000 6 281 38000 2 333 43000 6 2 767 40000 2 287 65000 6	The list “games” of each shelve are not empty, which means that the program has been able to read the input and create the objects of type game and add them to the list of their respective shelve

Test goal: Validade the process of finding a game for a client which has its game list not empty				
Class	**Method**	**Stage**	**Input values**	**Result**
GameStore	procccesFindGame	setupScenario3	*Client client = gameStore.getClients.get(3)*	True, which means that the program is able to search the games that are required by the client

Test goal: Validate that a client can not pay its games because he hasn't found them yet in the store				
Class	**Method**	**Stage**	**Input values**	**Result**
GameStore	procccessPayGame	setupScenario3	*Client client = gameStore.getClients.get(2)*	False, which means that the program is able to validate that a client can not pay its games because he hasn't found them yet

Solution of two test cases

Test goal: Validade the correct process of adding new games (11) to the game store and to its respective shelve. For this test the input values will be given as a console input but they are saved in a text file for technical reasons of this automatic test				
Class	**Method**	**Stage**	**Input values**	**Result**
Game Store	shelveAddGame	setupScenario2	4 331 17000 3 465 60000 6 612 80000 2 971 70000 6 5 441 30000 3 112 22000 6 229 28000 6 281 38000 2 333 43000 6 2 767 40000 2 287 65000 6	The list "games" of each shelve are not empty, which means that the program has been able to read the input and create the objects of type game and add them to the list of their respective shelve

Steps: After setting the Stage

1.

we started to iterate for each shelve

```
for (int i = 0; i <3 ; i++)
```

in this case we have 3 shelves

2.

We crate an object of type Scanner (sc) and read the first line of the input, then we save the value in a variable

```
int quantity = sc.nextInt()
```

```
4
331 17000 3
465 60000 6
612 80000 2
971 70000 6
5
441 30000 3
112 22000 6
229 28000 6
281 38000 2
333 43000 6
2
767 40000 2
287 65000 6
```

This first line represents the amount of games

3.

we started to iterate for each game

```
for (int j = 0; j <quantity ; j++)
```

In the iteration we read the lines below and we save them in the following variables using our object scanner. Each line contains the code , pricea and cuantity of each game

```
int codeGame = sc.nextInt();
int priceGame = sc.nextInt();
int quantityGame = sc.nextInt();
```

```
331 17000 3
465 60000 6
612 80000 2
971 70000 6

441 30000 3
112 22000 6
229 28000 6
281 38000 2
333 43000 6

767 40000 2
287 65000 6
```

4.

Then we use our object of the class GameStore and me use the method shelveAddGame(). Proviting the parameters needed.

```
test.shelveAddGame(test.getShelves().get(i).getCode(), codeGame, priceGame, quantityGame);
```

We use the i index in order to call the information of the shelve , saved in the list of shelves of the class GameStore, that will save the games that we create

5.

Inside the class GameStore we made use of the method findShelve in order to get the index in of the shelve in the list of shelves of the class

```
shelves.get(findShelve(code)).addGame(game);
```

Gives the index of the shelve with that code

an object of type Game created before whith the parameters given and got from the input

Finally we made a test doing 3 times assertFalse() for the 3 shelves we have.

The boolean condition will be if the list of game of the shelves is empty, and it will false because , as we have seen , the games were added but the test will fail

```
assertFalse(test.getShelves().get(0).getGames().isEmpty());
assertFalse(test.getShelves().get(1).getGames().isEmpty());
assertFalse(test.getShelves().get(2).getGames().isEmpty());
```

Test goal: Validade the process of finding a game for a client which has its game list not empty				
Class	**Method**	**Stage**	**Input values**	**Result**
GameStore	procccessFindGame	setupScenario3	*Client client = gameStore.getClients.get(3)*	True, which means that the program is able to search the games that are required by the client and add it to the stack of games of the client

Steps: Before we set Scenario3:

Step 1:
We create an object of type Client by getting it in the list of clients , set before in the game store.

```
Client client = test.getClients().get(4);
```

Step 2:
We use the method "proccessFindGame" of the class GameStore and assertTrue() , beacuse the method returns true if it could find the game that the client is looking for.

```
assertTrue(test.proccessFindGame(client));
```

OBJ