Assignment 3: Buffy the Vampire Slayer. Exceptions and files

Submission date: January 11th, 9:00

Objective: Exception handling and file I/O

1. Introduction

In this assigment we extend the functionality of the *Plants vs Zombies* game of the previous assignment in two ways:

- **Exception handling**: errors that may occur during the execution of the application can be more effectively dealt with using the exception mechanism of the Java language. As well as making the program more robust, this mechanism enables the user to be informed about the occurrence of an error in whatever level of detail is considered appropriate, while at the same time providing a great deal of flexibility in regard to where the error is handled (and the error message printed).
- **File handling**: a useful addition to the application would be the facility to save the state of a game to file and load the state of a game from file. To this end, we add two new commands, one to write to a file and the other to read from a file. The use of the Command pattern introduced in the previous assignment greatly facilitates the addition of new commands.

2. Exception Handling

In this section, we present the execptions that should be handled by the application and give some information about their implementation, as well as providing a sample execution.

In the previous assignment, the `execute` method of each command simply invoked a method of the `Game` class which implemented the functionality of that command (or at least, it should have!). You will have observed that there are circumstances in which a

command may fail, either in its parsing or in its execution. The execution of the `add` command, for example, will fail if the player does not have enough coins, or if the position chosen by the user to add the slayer is already occupied by another game object or is not even on the board.

In the previous assignment, the occurrence of such an error when implementing the functionality of the `add` command was likely communicated back to the execute method of the AddCommand class via a boolean return value (or, if not, by some ad-hoc mechanism involving specific methods in the Controller class, obliging the game to know about the controller). This use of a boolean value to communicate the occurrence of an error is very limiting, since such binary communication cannot include any indication of the reason for the occurrence of the error, nor can it include any other data about the error which may be required to handle it adequately.

In this assignment, we deal with these issues using the Java exception mechanism. An exception mechanism provides a flexible communication channel between the location in the code where an error occurs and the location in the code where that error is handled, along which any required data concerning the occurrence of the error can be sent from the former to the latter[1]. In many cases, the data concerning the occurrence of the error that is transmitted from one code location to another via an exception consists simply of an error message, and the handling of this error consists simply of sending that message to the standard output to be displayed on the screen. In the general case, however, more data about the error and its context may be transmitted between code locations and the error-handling may require more complex actions than simply printing a message to the screen.

In particular, in this assignment, the exception mechanism enables us to ensure that all messages to be printed to the standard output can be printed from the controller. It should be pointed out that file handling inevitably involves exception handling, particularly when reading from files, which is why these two topics are often introduced to students at the same time.

2.1. Concrete Types of Exception

First, we discuss handling exceptions thrown by the system, that is, those that are not explicitly thrown by the programmer (most of which are also created by the programmer, though the programmer can also explicitly throw system exceptions). You should at least handle the following exception:

- NumberFormatException, which is thrown when an attempt is made to parse the String-representation of a number and convert it to the corresponding value of type int or long or float, etc., in the case where the input String does not, in fact, represent a number and cannot, therefore, be so converted.

Regarding exceptions thrown by the programmer, you should create the following three exception classes: GameException, CommandParseException and CommandExecuteException, where the latter two exceptions inherit from the former and are to be used when an error occurs on parsing a command / executing a command respectively. These two exceptions are to be considered high-level exceptions. You may also wish to create low-level exceptions but these should not reach the controller, that is, they should be

[1]The error code mechanism of C and C++ is somewhat primitive in comparison, though it is also much less computationally costly, which is why C++ retains it as well as having an exception mechanism (though this exception mechanism is less type-checked and more difficult to use than the Java equivalent).

caught in the methods of the command classes and then wrapped in one of the two high-level exceptions. An example of an error that should produce an exception that will reach the controller as a CommandParseException is an `add` command with incorrect number of parameters. An example of an error that should produce an exception that will reach the controller as a CommandExecuteException is an attempt to add a slayer in a position on the board that is already occupied by another game element. Any exceptions thrown by the system during parsing, e.g. NumberFormatException, should also be wrapped in a CommandParseException.

In the Buffy the Vampire Slayer application, the exception mechanism can be used to ensure that all printing to the standard output is done from the controller (in the case of error messages, in a catch clause of the run method). It will now be much easier to use more specific error messages than in the previous assignment.

Summarising, you need to make the following changes to your application:

1. Define new exception classes: GameException, CommandParseException, CommandExecuteException as well as other low-level exceptions, see below.

2. The run method) method of the controller should catch the CommandParseException and CommandExecuteException thrown by the parse and execute methods of the Command class. It can do so by explicitly declaring that it catches exceptions of the type defined by the superclass of these two classes. Recall that the header of the textsfparse and execute methods must contain the appropriate throws clause (these exceptions should be *checked* exceptions).

3. Include the throwing and catching of exceptions in the appropriate classes, ensuring that they result in a CommandParseException or CommandExecuteException whose message can be printed by the controller, with the exception of any system exception that may occur in the parsing of the program parameters in the main method, which must be handled by the main method itself.

As already stated, all messages should now be printed from the controller, except those concerning the program arguments, which will be printed from the main, and those non-error messages concerning saving and loading files, which will be printed from the two new commands added for this purpose (see the following section). The controller code should now look something like the following[2]:

```
while (!game.isFinished()) {
    if (refreshDisplay) printGame();
        refreshDisplay = false;

    System.out.println(prompt);
    String s = scanner.nextLine();
    String[] parameters = s.toLowerCase().trim().split("  ");
    System.out.println("[DEBUG] Executing: " + s);
    try {
        Command command = CommandGenerator.parse(parameters);
        refreshDisplay = command.execute(game);
    } catch (GameException ex) {
```

[2]If you want to print both the message from the high-level exception and the message from the low-level exception it may contain, as in the examples given below, in the run method you should use the method getCause of the Exception class, first checking whether it returns null (the case where the high-level exception does not contain a low-level exception).

```
            System.out.format(ex.getMessage() + "%n%n");
        }
    }
```

2.2. Low-level exceptions

As well as the high-level exception classes CommandParseException and CommandExecuteException together with their superclass GameException you should create and use the following low-level exception classes (in the next section, we will add a few more):

- InvalidPositionException: thrown when a coordinates provided by the user denote a tile that is already occupied or that is not on the board.
- NotEnoughCoinsException: thrown when an attempt is made to execute any command which has a cost in coins but the user does not have enough coins to cover that cost.
- DraculaHasArisenException: thrown when an attempt is made to add Dracula to the board in spite of the fact that he is already present.
- NoMoreVampiresException: thrown when an attempt is made to add a vampire to the board when there are no more vampires left to come out.

2.3. Exception handling good practice

- An exception passes along a chain of method calls from the method that throws it to the method that catches it. The question arises: in which of the methods should it be thrown and in which should it be caught? For example, suppose that a method $m1$ calls a method $m2$ that calls a method $m3$ and an error may occur during the execution of method $m3$ which we want to handle in method $m1$. When the error occurs, there are two possible policies:
  - $m3$ returns an indication of error (without throwing an exception), say, the value false, to $m2$, $m2$ throws an exception on receiving the return value false from $m3$, $m1$ catches the exception.
  - $m3$ throws an exception when the error occurs, $m2$ doesn't catch the exception, $m1$ catches the exception.

  The implementation that is most coherent with the philosophy of exceptions is the second. Observe that, in consequence, methods should not return a boolean value when an error occurs but rather throw an exception.
- It is often good practice to catch a low-level exception to then thrown a high-level exception that wraps it (and which contains a less specific message than the low-level one). Assuming you have already implemented the second option of the above item, you should now ensure that low-level exceptions thrown during the execution of a command in the game, the board of the list class are caught in the execute method of the corresponding Command subclass which then throws a CommandExecuteException that wraps it. The high-level exception should contain a less specific error message than the low-level exception.

2.4. Examples of error states

The messages produced by the game are of three types: `[DEBUG]`, `[ERROR]`, `[GAMEOVER]`; each message is preceded by the appropriate tag. Here are some examples of error handling:

1. The `b 0 0 20` command (to add a bloodbank costing 20 coins) has executed correctly.

```
b 0 0 20
[DEBUG] Executing: b 0 0 20
Cycle number: 1
Coins: 42
Remaining vampires: 9
Vampires on the board: 1

 ------------------------------------------
|B-B[20] |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        | V^V[5] |
 -------- -------- -------- -------- --------
|        |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        |        |
 ------------------------------------------

Command >
```

2. If a vampire leaves the board via the l.h.s., the game stops and the following message is printed:

```
[GAME OVER] Vampires win!
```

3. The command `b 0 7 60` has failed due to the fact that the position `(0,7)` is invalid. Note that the player does not have enough coins either but the first error detected is the one shown to the user. After the occurrence of the error, which has not advanced the cycle number and after which the board has not been reprinted, the player tries to release a light flash to destroy the vampire present on the board. But this command also fails for lack of sufficient coins. Suitable error messages are printed for each situation, preceded by the tag `[ERROR]`. Note also the low-level exception messages preceded by the tag `[DEBUG]`

```
Cycle number: 1
Coins: 42
Remaining vampires: 9
Vampires on the board: 1

 ------------------------------------------
|B-B[20] |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        |        |
 -------- -------- -------- -------- --------
|        |        |        |        | V^V[5] |
 -------- -------- -------- -------- --------
```

```
   |        |        |        |        |        |
    -------- -------- -------- -------- --------
   |        |        |        |        |        |
    -------- -------- -------- -------- --------
   |        |        |        |        |        |
    ----------------------------------------------

Command >
b 0 7 60
[DEBUG] Executing: b 0 7 60
[ERROR] Failed to add bloodbank
[DEBUG] Position (0, 7): invalid position

Command >
l
[DEBUG] Executing: l
[ERROR] Failed to release light flash
[DEBUG] Light Flash cost is 50: Not enough coins

Command >
```

4. The player adds Dracula at position `(5,2)` and then attempts to add an explosive vampire at the same position. The second command fails and a suitable error message is printed:

```
Command >
v d 5 2
[DEBUG] Executing: v d 5 2
Cycle number: 0
Coins: 50
Remaining vampires: 2
Vampires on the board: 1
Dracula has arisen

    ------------------------------------------------------------------------
   |        |        |        |        |        |        |        |        |
    -------- -------- -------- -------- -------- -------- -------- --------
   |        |        |        |        |        |        |        |        |
    -------- -------- -------- -------- -------- -------- -------- --------
   |        |        |        |        |        |        |        |        |
    -------- -------- -------- -------- -------- -------- -------- --------
   |        |        |        |        | V-V[5] |        |        |        |
    -------- -------- -------- -------- -------- -------- -------- --------
   |        |        |        |        |        |        |        |        |
    ------------------------------------------------------------------------

Command >
v e 5 2
[DEBUG] Executing: v e 5 2
[ERROR] Failed to add vampire
[DEBUG] Position (5, 2): invalid position

Command >
```

Suppose the game evolves and arrives at the following state:

```
[DEBUG] Executing:
Cycle number: 6
Coins: 70
Remaining vampires: 2
Vampires on the board: 1
Dracula has arisen
```

```
 ------------------------------------------------------------------------
|        |        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        | V-V[5] |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        |        |        |        |
 ------------------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 7
Coins: 80
Remaining vampires: 1
Vampires on the board: 2
Dracula has arisen

 ------------------------------------------------------------------------
|        |        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        | V-V[5] |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        |        |        | V*V[5] |
 ------------------------------------------------------------------------

Command >
```

Note that the cycle has advanced and an explosive vampire has been added. No regular vampires was added so it must have been due to the fact that the random behaviour did not come out in favour. But for Dracula, it could also have been due to the fact that Dracula is already on the board. You must enure that if this situation occurs, a DraculaHasArisenException is thrown but is handled in such a way that the player is not informed of this failed attempt to add Dracula (in an expanded implementation, information may be written to a log but we will not implement any such treatment).

2.5. Execution examples

Next we provide two sample executions. The first shows the occurrence of parsing errors and the second shows the occurrence of execution errors.

1. Level easy with seed value 1.

```
Buffy the Vampire Slayer 3.0
Random generator initialized with seed: 1
Cycle number: 0
Coins: 50
Remaining vampires: 3
Vampires on the board: 0

 ------------------------------------------------------------------------
|        |        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- -------- --------
```

```
  |        |        |        |        |        |        |        |        |
   ------------------------------------------------------------------------
  |        |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   ------------------------------------------------------------------------

Command >
help me
[DEBUG] Executing: help me
[ERROR] Command help :Incorrect number of arguments

Command >
help
[DEBUG] Executing: help
Available commands:
[DEBUG] Executing: help
Available commands:
[a]dd <x> <y>: add a slayer in position x, y
[h]elp: show this help
[r]eset: reset game
[e]xit: exit game
[n]one | []: update
[g]arlic : push back vampires
[l]ight: kill all the vampires except Dracula
[b]ank <x> <y> <z>: add a blood bank with cost z in position x, y
[c]oins: give player 1000 coins (debugging command)
[v]ampire [<type>] <x> <y>; type = {""|"D"|"E"}: add a vampire in position x, y (debugging

Command >
add 0 vampiro
[DEBUG] Executing: add 0 vampiro
[ERROR] Invalid argument for add slayer command, number expected: [a]dd <x> <y>

Command >
v 0 0
[DEBUG] Executing: v 0 0
Cycle number: 0
Coins: 50
Remaining vampires: 2
Vampires on the board: 1

   ------------------------------------------------------------------------
  | V^V[5] |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   ------------------------------------------------------------------------

Command >
v w 2 2
[DEBUG] Executing: v w 2 2
[ERROR] Invalid type: [v]ampire [<type>] <x> <y>. Type = {""|"D"|"E"}

Command >
reseto
[DEBUG] Executing: reseto
[ERROR] Unknown command
```

```
Command >
reset
[DEBUG] Executing: reset
Cycle number: 0
Coins: 50
Remaining vampires: 3
Vampires on the board: 0

   ------------------------------------------------------------------------
  |        |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   ------------------------------------------------------------------------
  |        |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |        |
   ------------------------------------------------------------------------


Command >
exit
[DEBUG] Executing: exit
[GAME OVER] Nobody wins...
```

2. Level hard with seed value 5.

```
Buffy the Vampire Slayer 3.0
Random generator initialized with seed: 5
Cycle number: 0
Coins: 50
Remaining vampires: 5
Vampires on the board: 0

   ------------------------------------------------------------
  |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |
   ------------------------------------------------------------


Command >
b 6 0 20
[DEBUG] Executing: b 6 0 20
[ERROR] Failed to add bloodbank
[DEBUG] Position (6, 0): invalid position

Command >
b 5 0 30
[DEBUG] Executing: b 5 0 30
Cycle number: 1
Coins: 33
Remaining vampires: 4
Vampires on the board: 1

   ------------------------------------------------------------
  |        |        |        |        |        |B-B[30] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
```

```
  |        |        |        |        |        | V^V[5] |
   ------------------------------------------------------------

Command >
a 0 2
[DEBUG] Executing: a 0 2
[ERROR] Failed to add slayer
[DEBUG] Defender cost is 50: Not enough coins

Command >
b 3 2 40
[DEBUG] Executing: b 3 2 40
[ERROR] Failed to add bloodbank
[DEBUG] Defender cost is 40: Not enough coins

Command >

[DEBUG] Executing:
Cycle number: 2
Coins: 36
Remaining vampires: 4
Vampires on the board: 1

   ------------------------------------------------------------
  |        |        |        |        |B-B[30] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] |
   ------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 3
Coins: 49
Remaining vampires: 4
Vampires on the board: 1

   ------------------------------------------------------------
  |        |        |        |        |B-B[30] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        | V^V[5] |        |
   ------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 4
Coins: 52
Remaining vampires: 3
Vampires on the board: 2
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        |B-B[30] | V-V[5] |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
```

```
  |        |        |        |        | V^V[5] |        |
   ------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 5
Coins: 65
Remaining vampires: 2
Vampires on the board: 3
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        |        | V-V[5] |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        | V^V[5] |        | V*V[5] |
   ------------------------------------------------------------

Command >
v d 6 1
[DEBUG] Executing: v d 6 1
[ERROR] Failed to add vampire
[DEBUG] Dracula is already on board

Command >
a 0 2
[DEBUG] Executing: a 0 2
Cycle number: 6
Coins: 15
Remaining vampires: 1
Vampires on the board: 4
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        | V-V[5] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] |
   -------- -------- -------- -------- -------- -------- --------
  | <->[3] |        |        | V^V[4] |        | V*V[5] |
   ------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 7
Coins: 15
Remaining vampires: 1
Vampires on the board: 4
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        | V-V[5] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] |
   -------- -------- -------- -------- -------- -------- --------
  | <->[3] |        | V^V[3] |        | V*V[5] |        |
   ------------------------------------------------------------

Command >
```

```
[DEBUG] Executing:
Cycle number: 8
Coins: 25
Remaining vampires: 1
Vampires on the board: 4
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        | V-V[5] |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] |        |
   -------- -------- -------- -------- -------- -------- --------
  | <->[3] |        |        | V^V[2] |        | V*V[5] |        |
   ------------------------------------------------------------

Command >

[DEBUG] Executing:
Cycle number: 9
Coins: 25
Remaining vampires: 0
Vampires on the board: 5
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        | V-V[5] |        |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] | V*V[5] |
   -------- -------- -------- -------- -------- -------- --------
  | <->[3] |        | V^V[1] |        | V*V[5] |        |        |
   ------------------------------------------------------------

Command >
g
[DEBUG] Executing: g
Cycle number: 10
Coins: 25
Remaining vampires: 0
Vampires on the board: 3
Dracula has arisen

   ------------------------------------------------------------
  |        |        |        |        |        | V-V[5] |        |
   -------- -------- -------- -------- -------- -------- --------
  |        |        |        |        |        | V^V[5] |        |
   -------- -------- -------- -------- -------- -------- --------
  | <->[3] |        |        |        |        | V*V[5] |        |
   ------------------------------------------------------------

Command >
v 4 1
[DEBUG] Executing: v 4 1
[ERROR] Failed to add vampire
[DEBUG] No remaining vampires

Command >
l
[DEBUG] Executing: l
[ERROR] Failed to release light flash
[DEBUG] Light Flash cost is 50: Not enough coins
```

```
Command >
g
[DEBUG] Executing: g
Cycle number: 11
Coins: 25
Remaining vampires: 0
Vampires on the board: 2
Dracula has arisen

 ------------------------------------------------------------
|        |        |        |        |        | V-V[5] |
 -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        | V^V[5] |
 -------- -------- -------- -------- -------- -------- --------
| <->[3] |        |        |        |        | V*V[4] |
 ------------------------------------------------------------

Command >
g
[DEBUG] Executing: g
Cycle number: 11
Coins: 15
Remaining vampires: 0
Vampires on the board: 0

 ------------------------------------------------------------
|        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- --------
|        |        |        |        |        |        |        |
 -------- -------- -------- -------- -------- -------- --------
| <->[3] |        |        |        |        |        |        |
 ------------------------------------------------------------

[GAME OVER] Player wins
```

3. Serializing and saving to file

3.1. Serializing the game

In computing, the term *serialization* refers to converting the state of an executing program, or of part of an executing program, into a stream of bytes, usually with the objective of saving it to a file or transmitting it on a network. The term *deserialization* refers to the inverse process of reconstructing the state of an executing program, or part of an executing program, from a stream of bytes. Serialization/deserialization in which the generated stream is a text stream is sometimes referred to as *stringification*/*destringification*. Clearly, the format used for serialization/stringification should be designed in such a way as to facilitate deserialization/destringification.

Here, our interest is to produce a text stream that represents the current state of the game, rather than the complete current state of the executing program, with a view to writing this state to, and reading this state from, a text file. Note that the textual representation produced by the game printer is not a suitable format since deserialization/destringification of text conforming to this format would be rather complicated. We therefore define a *stringified* format, in which the state of the game is represented as a sequence of game elements, each of which is represented as a sequence of values. This format is useful for debugging purposes and also for saving the state of the game to a text

file. The details of the stringified format are as follows:

- The first line contains the following information: `Cycles: <cycleCount>`
- The next line contains the following information: `Level: <level name>`
- The next line contains the following information: `Coins: <number of coins>`
- The next line contains the following information: `Remaining vampires: <number of vampires remaining>`
- The next line contains the following information: `Vampires on board: <number of vampires on the board>`
- The next line is blank
- The next line contains the header text: `GameElement objects:`
- The following lines contain the stringification of each game element on the board. The value cyclesToMove refers to the number of cycles that need to pass before the game element can move (0, if it can move on the next cycle):
  - For each slayer: `S;x;y;lives`
  - For each regular vampire: `V;x;y;lives;cyclesTillMove`
  - For Dracula: `D;x;y;lives;cyclesToMove`
  - For each explosive vampire: `EV;x;y;lives;cyclesTillMove`
  - For each bank blood: `B;x;y;lives;cost`

For example, if the state of the game is as follows:

```
Number of cycles: 7
Coins: 990
Remaining vampires: 1
Vampires on the board: 3
Dracula is alive

 ------------------------------------------------------------
|B-B[100]|        |        | V-V[4] |        |        |        |
 -------- -------- -------- -------- -------- -------- --------
|        | <->[3] |        |        |        | V*V[5] |        |
 -------- -------- -------- -------- -------- -------- --------
|        |        |        | V^V[4] |        |        |        |
 ------------------------------------------------------------
```

its stringification will be as follows:

```
Cycles: 7
Coins: 990
Level: HARD
Remaining vampires: 1
Vampires on board: 3

GameElement objects:
B;0;0;1;100
V;3;2;4;1
D;3;0;4;1
S;1;1;3
EV;5;1;5;1
```

We define a new command `stringify` (in the short version, the letter `t`) that sends the state of the game in stringified format to the standard output. Note that the `stringify` command does not change the state of the game. We then define another new command `save` (in the short version, the letter `s`) that saves the state of the game in stringified format to a file.

3.2. Implementation

We first define the StringifyCommand class and then the SaveCommand class. The execute method of the former class results in the stringification of the game state being sent to the screen by calling a stringify method of the Game class. The execute method of the latter class results in the stringification of the game state being saved to a file, after a suitable file header and a blank line, by calling a save method of the Game class that, in turn, calls this stringify method.

To save the state of a game to a file, you should take into account the following:

- The parse() of the SaveCommand should throw CommandParseException if the number of arguments is incorrect.
- The execute() method of the SaveCommand class should take into account the following:
  - To simplify, you do not need to check whether the text provided by the user can be a valid file name, this notion being operating-system dependent, nor whether the program has write permissions for the file, if it already exists. If these conditions are not met, on attempting to open the file, the JVM will throw an exception that you should catch and wrap in a CommandExecuteException.
  - Add the extension `'.dat'` to the file name provided by the user. If a file with this name already exists it will be overwritten and if not does not, it will be created (you do not need to do anything to get this behaviour since it is the default behaviour in Java).
  - In Java there are many different ways to connect a program to a file; we propose the following: create a FileWriter object and a BufferedWriter object that wraps it. Place the code which opens the file in a *try-with-resources* construct, catching any possible IOExceptions.
  - If the state of the game has been succesfully saved to file, print the following message on the screen (from execute() method of the SaveCommand class; no need to print it from the controller):
    "Game successfully saved in file <*filename_provided_by_the_user*>.dat".

For example, after executing the command save myData, the file `myData.dat` will contain the following text:

```
Buffy the Vampire Slayer v3.0

Cycles: 7
Coins: 990
Level: HARD
Remaining Vampires: 1
Vampires on Board: 3
```

```
GameElement objects:
B;0;0;1;100
V;3;2;4;1
D;3;0;4;1
S;1;1;3
EV;5;1;5;1
```