Project requirements in terms of "technical" issues

The "technical" issues concern the coding and software development items that must be included in the final project. These are the indicators of various abilities that will be demonstrated. The following is the basic list of "technical" requirements or, if you like, ability indicators. Some of the items you may not recognize. An objective of the course is that by the end of the semester you will have some familiarity with all of them. These will be modified just a bit during the remainder semester, but they give you an idea of the scope of the technical components of the project.

Programming and software construction "technical" requirements overview

- Use of abstract and concrete classes (includes appropriate subclassing).
- Use of interfaces.
- Use of packages
- Use of visibility modifiers including appropriate visibility for both field and method members.
- Use of exception handling.
- Use of appropriate collection data structure and their iterators.
- Use of generics.
- Use of javadoc comments and documentation.
- Use of appropriate programming constructs to store and retrieve data (options include: simple files, object files, structured files, SQL style data base, NoSQL style database, remote files).
- Use of appropriate programming constructs to build a suitable user interface including at least display areas (text and graphics), buttons, lists, and icons (Other items are optional).
- Use of appropriate programming constructs to build simple 2d graphics with simple animation or other effects.
- Demonstration of design knowledge including CRC cards and use cases.
- Demonstration of knowledge of design patterns.
- Construction of at least some appropriate UML design diagrams for use cases and classes.
- Constructions of some appropriate unit tests and an indication of regression testing.
- Construction of at least one package that will be evaluated and will be shareable by other teams.

Required Components

- Graphical user interface
- Text formatting and processing (html or other)
- Graphics sampled or constructive with manipulation or animation (could be all four).
- Storage and retrieval of information (flat file, database, XML or all of these).
- Editing and configuring the software product.
- MVC architecture/design

Documentation "technical" requirements

You will need to provide appropriate documentation with your project. Although the construction of appropriate javadoc documentation for the code is required, this is not the same as the project documentation. Remember that you can use javadocs through the design phase to document the design and, in cooperation with the NetBeans IDE, provide instructions for implementation. The material that follows will indicate what your project documentation should contain.

Cover Page
The cover page should indicate the name of your project, the members of your team, that this is a course requirement and the date of the release of the document.

Intellectual property
Provide a simple assignment of a copyright and intellectual property rights. For example,
Copyright 2017 AUTHORS.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3, (3 November 2008) or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license can be found at https://www.gnu.org/licenses/fdl.html.

1.0 Project description
This section should provide your basic description of the project. These will be very brief. You will have a chance to expand latter. It should be based on the project description that you have submitted earlier.

2.0 Project management
This is the description of the work that went into the project.

> *History*
> This section will provide a description of what you have done during the semester. You should provide a brief description of the previous submissions and what improvements were made over the semester.
>
> *Personnel*
> Identify the members of your team. Briefly indicate the educational background of the member and any additional professional or experiential detail, if relevant. If a member of the team had the lead on a particular part of your project, clearly indicate this.
>
> *Effort*
> Indicate approximately how much time the team as a whole as spent on the project. Also indicate the number of meeting that occurred during the semester. A time line would be helpful.

3.0 Use Cases
Place your principle use case or cases here. You should wrap these in appropriate text so that they can be easily understood. If there is more than one, number them so that they can be referenced in other sections. Other use cases that you have developed for previous parts of the project, or use

cases for special conditions should be placed in an appendix.

4.0 Requirements

The description of the requirements should be based on your project description and the use cases.

Functional requirements

If you use special words with specific meanings, define them first. This should be from the user or client point of view. Use the use cases as a guide.
Describe:

- what the functions of the software are, not how these are done
- what is expected from the program, not what the program looks like
- the main flow of events

Each individual requirement should be numbered in a standard way. These will be referenced in your testing section. Example:

R1 The software shall allow for the selection of a game from a collection of games.
Rn The software will keep an accurate accounting of the user's available money.

User interface requirements

If you use special words with specific meanings, define them first.
Describe:

- what the program should look like
- what the input elements are
- what the output elements are

Remember to discuss how invalid user input is handled, i.e., what happens if the information supplied by the user is incorrect. Do not describe how the error handling is actually realized, just indicate what follows if some input is invalid.

Future modification and extensions

Describe how the program could be extended in the near future. These are the things that will not appear in your implementation, but you will take these things into consideration in your design. You may use additional scenarios, window sketches, etc., to illustrate the future extension.

Summary

This should be a list of statements that indicate what the software shall do. These should have a clear numbering and be clearly linked to the scenarios. For example, "The software shall accept a user selection of direction and produce the appropriate scene as presented in Use Case 1.4". You might consider a table since you will need to show how the requirements are linked to the scenarios. Consider this to be the general contract for your system. You will satisfy the contract if you satisfy each requirement. This does not mean that you have satisfied the contract well or that the contract was interesting. These last two items will also be used in evaluating your performance

Associated tests

In this section describe the test you will use to determine whether the requirement is

satisfied. There will probably be a bit of text but there should also be a clear summary (perhaps a table) that indicates the requirement and the test.

5.0 Design

This section describes the static and dynamic elements of your system and the relation of the model to the user interface (views and controls). Descriptions of the final CRC cards may be used here. The actual CRC cards should be placed in an appendix. At least some of the design should be documented with appropriate UML class diagrams.

Since the MVC architecture is being used in the project, describe the model, the views, and the controls. Indicate the communication between the parts. Once again this should not have a great amount of detail. This section is about design and not implementation.

Model design
What is in the model and what are the ways in which it can change?

View design
What are the views of the data and how are they generated?

Control design
What are the elements by which the user can request a change to the mode?

Communication design
How do the model, views, and controls communicate?

6.0 Implementation

This section provides descriptions of how the design is implemented. Do not provide excessive detail. The details should be in the javadocs and other code comments. You should indicate which, if any, design patterns you have implemented.

Packages and classes
Describe the packages, classes, components, interfaces, and the relationships that implement the design. Make sure you explicitly indicate how a package, class or group of classes relates to or satisfies a design specification in the previous section. Use text here, but make clear references to the javadocs. For each public class indicate in what package it is contained. For each public class clearly indicate if it is abstract or concrete, whether it extends another class, and whether it implements an interface. For each public class describe the API for that class.

Utility classes and packages
Describe the packages and classes that are used to satisfy design specifications but are not in the actual design. This would be the placed to indicate the way in which you satisfied the four component requirements that are not explicitly examined above. You can consider the packages and classes that deal with the views and controls as satisfying the GUI component.

Test plan
Describe what you will attempt to test and how you will attempt to test it. These tests should be clearly related to the requirements. Note that one package or specific collection of classes must have associated JUnit tests. It would be better if there were test for all relevant packages and classes. Describe the tests and the results.

Tested functionality
Describe functionalities and the outcome of the tests, i.e., list functionalities that have been tested and present the results of the tests. Note that all the use cases/functionality listed in the requirements document should be tested.

Untested functionality
Describe the functionalities that have not been tested, and include an explanation of why the tests have not been completed.

7.0 Discussion
Provide a discussion of any items that you think are important. This should include anything that you want to say about trade-offs that you have made, particular implemented features that you want to highlight, and any lessons learned.

Suggestions

- Use the text as a reference source. You should use the text to help explain your decisions.
- Start the documentation part early. Begin by filling in the basics.
- Make sure that you have the high-level, more abstract parts written early.
- As the project proceeds, check against the more high-level descriptions. If necessary, change the description.
- Develop a uniform numbering system for the various parts of the document. The numbering should provide clear reference points between sections. Write each particular section as it is completed. If things change someone will need to fix the documentation for that section.
- Keep a 'check sheet' for the programming and software construction 'technical' requirements. Make sure that the documentation is shared.
- If possible, make a picture.
- Develop a time line and keep meeting records. Use these for the project management section

Scoring

Item	Points
The program launches and runs correctly	10
Javadocs are complete and informative	10
- GUI component: Includes views, controls, and communication	10
- Graphics component: Graphics sampled or constructive with manipulation or animation (could be all four).	5
- Text component: Text formatting and processing (html or other)	5
- Configure component: Editing and configuring the software product	5
- Persistence component: Storage and retrieval of information (flat file, database, XML or all of these)	5
Structure of packages and classes incorporates object oriented principles. Inheritance, interfaces, polymorphism, encapsulation.	15
Code quality: Includes appropriate use of language features including visibility, collections, and so on. Demonstrates appropriate use of design patterns.	15
Project documentation	10
General quality of project	10