Project Evaluation Point Justification Guide

Please see original code and other materials for further information

1. **Object Oriented Design**

Proper class design and organization *Refer to original code. *Refer to UML diagram in documentation folder for class structure.	*Mythical Maze* meets the general specifications for object oriented programming (OOP). All object classes (shapes, characters, blocks, etc.) in the *Mythical Maze* game contain fields for storing information such as location and methods for performing simple procedures. Classes interact with each other through passing parameters.
Code Reuse *Refer to original code.	One of the key goals of the *Mythical Maze* game was to cut code reuse to minimal levels to allow for the process of code reading and editing easier. Thus, the code in *Mythical Maze* has been repeatedly checked to eliminate reuse, with the implementation of several superclasses and interaction between methods.
Use of encapsulation *Refer to original code.	The *Mythical Maze* game uses encapsulation as per the guidelines of basic object oriented programming. Class fields are made globally accessible, or public, only when absolutely necessary. Instead, most fields are restricted, or private. Data is bundled into instance methods for all objects of the game.
Use of inheritance *Refer to original code and UML diagram, located in source code and documentation folders, respectively.	In the *Mythical Maze* game, we discovered that many objects of the game were related to each other as they shared similar attributes. We therefore used inheritance for several object classes. For example, many of the shapes in the *Mythical Maze* game therefore were coded as subclasses of a parent Shape superclass. The subclasses shared the same fields, constructors, and instance methods.
Use of software design patterns	*Mythical Maze* follows object oriented programming for software design.

2. **Design Analysis**

Interface Design *Refer to actual game.	For the purpose of creating a user friendly game, the makers of the *Mythical Maze* game implemented several graphical interface frames for the menu, player selection, and gameplay. For example, the menu has buttons to navigate around to scores, credits, and to play the game. The game itself has a heads up display which features user statistics and buttons for saving and leaving the game.
Data Flow Diagrams *Refer to documentation folder for diagram.	Before the coding began, the design of the *Mythical Maze* game included the creation of a data flow diagram. For the diagram, please refer to the Documentation folder. The diagram features how information is moved around throughout game play. It also analyzes where data must be extracted and written into local database files.
Adherence to Software Development Methodology *Refer to Project Plan in documentation folder.	The waterfall methodology was followed in the development of the *Mythical Maze* game. For details of the waterfall methodology, please refer to the diagram located in the documentation folder or in the project plan. The methodology allowed for a streamlined creation of the game without any major backtracks due to poor planning.

3. **Code Documentation**

Comment blocks explaining classes, methods, and complex sections of logic *Refer to Documentation folder, Javadoc API, and original code.	For the sake of reading code, the makers of *Mythical Maze* have included an entire interactive Javadoc API, located in the documentation folder of the game. The API contains descriptions of all classes and methods. The *Mythical Maze* code also contains the same material, along with line by line commenting for harder sections of logic and algorithms.
In-game tutorial or walkthrough *Refer to actual game and user manual, located in documentation folder.	The *Mythical Maze* game contains a basic tutorial level along with a story plot introduction for new players. A detailed description of the game with rules is also provided in the user manual.

4. **Crash Reporting**

Generate crash report on application crash *Refer to ErrorLogger.java, CrashHandler.java classes for original code. Though the existing code should be errorless, errors can be created by editing the source code. Errors and crash reports will be logged in text files.	To help with editing code during debugging, the *Mythical Maze* game utilizes error logging and crash handling classes. One function is to generate crash reports on unrecoverable errors. The *Mythical Maze* game creates an in depth report log on the type of error with custom messages and finds the location of the error in the code. Error messages are also provided to users to help with debugging.

5. **Data Driven Design**

Application makes use of data driven design: runtime settings are adjustable via text file or database *Refer to settings text file and original code.	Settings for the *Mythical Maze* game can be changed by the user in an pullup options menu. The menu allows the user to change music, sound effects, and other game settings. These settings are stored in a local text file.
Session data (saved games, high scores, etc.) are stored via flat file or database for later reuse *Refer to profile files in profiles folder and code. Files are created after playing the game and creating profiles.	The *Mythical Maze* game stores player information into individual files. These files correspond to each player and contain critical information such as score, player name, and level. High scores are also available in a separate file.

6. **Error Handling**

Proper use of error/exception handling techniques *See ErrorLogger.java, log files, and original code. Logs will only be created in the event of an error, so source code must be edited to result in an error for viewing.	For debugging purposes, the *Mythical Maze* has error/exception handling code that logs all incidents to text files. Code is in place to provide custom messages, including incident descriptions and locations. Users will be able to view special messages for when an error occurs; this will also help with error reporting during testing.
Clear user alerts on recoverable and non-recoverable error conditions *See ErrorLogger.java, CrashHandler.java for original code and log files for logged messages. Again, source code must be changed in order for a crash to occur that can be logged.	When an error occurs in the *Mythical Maze* game, the user is notified of the incident. The messages are custom fitted to the specific problem. For example, if images are unable to load, a specific message will describe the error. The user can then decide how to proceed. The user can exit the game or continue playing.

7. **Logging**

Log system events to dedicated text file for debugging *See EventLogger.java and log files. Event log is created once the game is played.	For easy debugging, system events are logged to provide insight into actions that occur. The *Mythical Maze* game has a special error logging system that logs custom messages. For example, load messages are logged to help determine I/O errors. User actions are documented to allow debugging to allow for a better user experience.
Log system errors to dedicated text file *See ErrorLogger.java, CrashHandler.java, SystemEvents.java, and log files. Log files for errors and crashes will be nonexistent until an error appears. To create an error, please edit the source code and run with faulty code.	Again, for easy debugging, all system errors, no matter how small or large, are logged to a dedicated text file. Items logged include: messages, error locations, user prompts and actions, descriptions, and much more.