****ALL OF THE FOLLOWING INFORMATION CAN BE FOUND IN THE USERMANUAL AND THE GAME'S ORIGINAL CODE****

Evaluation Item	Justification and Location
Proper class design and organization	Proper class design and organization is used throughout the program. Classes are organized into methods that each perform a single function to allowing for maximum code reuse and readability. Examples of this can be found throughout the program.
Code Reuse (minimize code duplication)	Code is divided into methods that each preforms a single function to maximize code reuse. For instance the getImageKey() method in the GameObject class preforms a single function and is used and inherited by many other methods throughout the program. One of the primary methods used to promote code reuse is Inheritance which allows subclasses to all access and use methods in their parent classes. An example of this is the NPC class which provides the basic movement and attack functions used in all of the various monster classes.
Use of encapsulation	Private variables and objects are used to encapsulate program objects and functions into the classes in which they are declared. This protects them from unintentional outside modification and increases the simplicity and reusability of classes within the program. Encapsulation is used throughout the entire program however an example would be the private variables including 'drawables' and 'money' and that are used in the GameRunner class.
Use of inheritance	Inheritance is used by many classes to allow for code reuse and simple interaction between objects. The GameObject class contains basic methods used to move, draw, and modify objects on screen. These methods are inherited by all of the objects and abstract classes that rely on drawing objects on screen

	including the PhysicsObject, Player, and Coin classes to reduce unnecessary code duplication. The use of these uniform classes and properties also makes interactions easier. For example objects that extend the PhysicsObject class gain access to code that allows them to collide and exchange information with one another.
Use of software design patterns	Standard software design practices were used in the development of Pyramid Plunder to insure the finished game was functional and met the provided specifications. From a project prospective we adhered to our waterfall design process for the development cycle of our game to insure it met specifications. Before developing the game we determined a timeline of events necessary to complete the project on time and account for potential setbacks. From a software perspective we started by creating a storyboard of our game and determined the necessary class structures through the creation of a preliminary UML diagram which we continued to revise and modify throughout the development process. We consolidated objects and functions into distinct classes and methods to maximize code usability and facilitate a smooth and timely development process. (The UML diagram, timeline, and related documentation can be found in the documentation folder)
Interface Design	When designing the interface we relied on our *Dara Flow Diagram* and *Storyboard* both of which can be found in the documentation folder. We fine-tuned our interface design based on feedback we got during the testing phase of development. The relationships between classes and methods in our program and the overall organization and structure of the program were determined based on our *UML Class Diagram* which is also available in the Documentation folder.
Data Flow Diagram(s)	Our *Data Flow Diagram* and related UML Class Diagram can be found within our Documentation folder.
Selected and adhered to a software development methodology	We initially determined to develop our project based on a classic waterfall development cycle. A diagram

	of the projects lifecycle and development can be found in the Documents folder. We followed strict timelines detailing the completion dates of various aspects of the project. These can also be found in the Documentation folder.
Comment blocks explaining classes, methods and complex sections of logic	The comments were done as headings on each class or method detailing its function and the parameters. JavaDocs detailing each method have been provided within the Documentation folder and can be opened using any standard web browser.
Provide an in-game tutorial or walkthrough for instructional purpose	A fully playable tutorial can be accessed through the games main menu screen. It details the basic controls and features of the game and prepares the player to take on their first pyramids.
Generation of crash reports (via text file or dialog box) on application failure	The "LogHandling" class is used to display an error message detailing the reason for the crash. The details also are saved to an error logging file titled 'errorreport.txt'.
Application makes use of data driven design: runtime settings are adjustable via text file or database	Data is saved and can be edited in a user specific file. Some settings, including level generation settings, can also be accessed from menus within the game.
Session data (saved games, high scores, etc.) are stored via flat file or database for later reuse	Player's session data is saved to a user specific .dat flat file when the save game option is selected form the in game menu and at various auto save points by the "SaveLoad" method. This data includes the locations of objects within a level, a player's status including health and gold, enemy's status, and available weapons. High scores are saved to as separate non-user specific file automatically upon a user's completion of a level and passwords are saved in a numerically encrypted flat data file.
Game session is resumable from saved session data	Players saved game will be resumed automatically from their user specific data file. If they saved during the middle of a level, that level will automatically be resumed. Player's stats, inventory, coins, and weapons will be restored to them when they sign back in regardless of whether they saved during or after completing a level.
Proper use of error/exception handling techniques	Error handling is present throughout the program. The dedicated ErrorLogger class will alert the user of any errors, log those errors to a file, and determine if the program can continue running based on the severity of

	the error.
Clear user alerts on recoverable and non-recoverable error conditions	The ErrorLogger class responds differently depending on the type of error and the location in the program if occurs at. Recoverable and non-recoverable errors are handled by separate methods in the ErrorLogger class. When errors are logged in the error log file, their severity is indicated. The type of error determines the way it is handled. A major error that could impact the play of the game will result in the user being alerted by an onscreen prompt. A non-recoverable error will cause a more severe warning message and the program closes itself.
Log system events to dedicated text file for debugging	System events including starting and saving games are handled by the “ErrorLogger” class and are saved in a separate dedicated text file for debugging.
Log system errors to dedicated text file	Error Logging is handled by the “ErrorLogger” class which notifies the user of errors and there severity and saves information about them to a dedicated text file for debugging.
Project application submission provides directions for compiling/building on judging hardware, including requirements and dependencies.	This information can be found in the readme and user manual.
Project application compiles successfully.	If you have trouble compiling the application please read the source code readme. A pre-compiled jar file is provided.
Project application runs successfully	If you have trouble compiling running application please consult the provided redme file.
Project application is a playable game	Please consult the readme and user manual.
Installer included for project application.	The game is provided in the form of a self-extracting jar file. It will automatically extract and run. Please consult the readme for further information.