Final Group Project Milestones

The final group project will be divided into several milestones as described below. Deliverables and Rubrics are provided below.

# Milestone Deliverable 1

## Team Formation and Contract

### Weight – 5%

### Due: End of Week 4

The purpose of the first milestone is to:

1. Form groups of 4 students, with other numbers permitted when class size/4 has a remainder.
2. Complete a contract indicating each students commitment and contribution to the group.
3. Scheduling meeting times with and without the professor.
4. An indication of which game is going to be the subject of the project and an explanation of the game rules in the student’s own words, whether this is a student choice or assigned by the professor.
5. Setup of GitHub Version Control – shared with all group members and the professor.

## Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Incomplete or Unacceptable | Major Improvement Required | Complete but with little thought or analysis | Complete with minor issues or inconsistencies | Outstanding |
| Contract | 0 | 1 | 2 | 3 | 4 |
| Meeting Schedule | 0 | 0.5 | 1 | 1.5 | 2 |
| Game Details | 0 | 1 | 2 | 3 | 4 |
| GitHub | 0 | 0.5 | 1 | 1.5 | 2 |
|  |  |  |  |  | /10 |

# Milestone Deliverable 2

## Requirements Specification and Project Management

In this milestone, students will utilize their experience from Systems Development courses and previous OOP courses to determine the application requirements.

### Weight – 10%

### Due: End of Week 6

### Requirements

Requirements will be very detailed and specific. Requirements should be organized by major application components; such that similar requirements are grouped together. Each requirement should then be given a priority level using the [MoSCoW](https://en.wikipedia.org/wiki/MoSCoW_method) prioritization of: Must Have (M), Should Have (S), Could Have (C), or Will not have (W).

#### Must Have (M):

are requirements that will be part of the determination of a successful project or not. All Must Have requirements must be complete for a successful project.

#### Should Have (S):

are requirements that absolutely should be included in the project, but if significant changes need to be made, these requirements might be able to be dropped.

#### Could Have (C):

are requirements that will only be implemented if time permits. These requirements are often satisfied in future updates and subsequent versions if time does not permit. This list also means that the decision of which features will be dropped under time constraints is made early in the project, versus near the end when time runs out.

#### Will not Have (W):

Are requirements that may be considered by some but will not be included in the final deliverable for the current version. These requirements are included as it is very important in determining the scope of the deliverable and make expectations clear and defined.

More information can be found at: [MoSCoW method - Wikipedia](https://en.wikipedia.org/wiki/MoSCoW_method)

### Project Management

The list of requirements, and their priorities, will often lead developers to a list of tasks that will need to be completed through the design and development process.

Each group will be required to setup a GitHub Project linked to their repository, where they will create a Can Ban view with the following statuses: ToDo, InProgress, InTesting, ForReview, Complete.

* **ToDo** – tasks that are for future completion that have not yet been started
* **InProgress** – a task has been assigned to a developer and they have started the task
* **InTesting** – the developer has initially completed the task, but is still testing it themselves
* **ForReview** – the developer has completed and tested the task and is submitting the task for group review in the next agile meeting. Tasks accepted by the group are moved to Complete, tasks that are not accepted are moved back to InProgress and either assigned back to the original developer, or a new one is appropriate.
* **Complete** – Tasks have been completed, internally tested, and accepted by the group as complete. From this point forward, if mistakes are found, it is now everyone’s responsibility and not that of only the original developer.

Coming up with tasks that will be entered into the project and initially set as ToDo should be completed. Groups may choose to create high level milestone for the project (such as front end, game logic, graphics, documentation, etc..) and place tasks into the appropriate milestones.

Thinking of Agile development practices, tasks should be very detailed and short in nature. A student should be able to complete the task in one sitting and not be spread over several days. This leads to a great amount of efficiency as well as coinciding with object-oriented programming philosophies of independence and modularity.

The initial tasks in the tasks list should be around the subject of design tasks about the setup of the application, architecture, data flow, meeting schedules, and looking ahead to milestone 3.

GitHub project will be expected to be continually used from this point forward for the remainder of the project. Further grades will be obtained in the final submission for the continued use of GitHub project management throughout the project.

## Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Incomplete or Unacceptable | Major Improvement Required | Complete but with little thought or analysis | Complete with minor issues or inconsistencies | Outstanding |
| Requirements List | 0 | 1 | 2 | 3-4 | 5 |
| Requirements Prioritization /Categorization | 0 | 0.5 | 1-1.5 | 2-2.5 | 3 |
| GitHub Project Setup | 0 | 0.5 | 1 | 1.5 | 2 |
| Initial Task Determination | 0 | 1 | 2 | 3-4 | 5 |
|  |  |  |  |  | /15 |

# Milestone Deliverable 3

## Application Design and Data Specification

This is the group’s chance to put some thought and analysis into the design of the system and to document how they plan to follow OOP practices. Groups will create class diagrams, data structure, and data flow diagrams.

### Weight – 15%

### Due: End of Week 8

## System Architecture and Assets

Groups will determine the design architecture for their system. Will they be producing an n-tier application, using MVC, which libraires will be utilized, what custom libraries will be created as part of the project, in addition to strategy to be used for code organization. This can be presented in a manor appropriate for the groups project, with the suggestion that the professor be given the chance to review the presentation before submission (during week 8 meetings).

A list of required external assets is to be gathered, and appropriate citations or references gathered for both moral and legal obligations to accredit the original creators. This will include any graphical images, code algorithms, external libraries, or written documentation citations.

## Class Diagrams

Groups will create one, or more, class diagrams that display that proposed code architecture and data structures. Each class should include:

* Name of each class
* Properties, for object classes
* Included methods (static and instance)
* Inheritance, relationships, associations, dependencies, aggregations, and compositions where appropriate.
* Indication of access levels (private, public, shared)

The purpose, and advantages, of creating these class diagrams in advance is:

* all developers know the design in advance and can create code that should integrate without refactoring, this is extremely important in multi-developer environments,
* tasks can be assigned to individual developers at a micro-task level and completed in short amounts of time such that problems such as file conflicts, code synchronization, and project delays are minimized,
* Code refactoring, and continued development does not stray from the design concept and result in spaghetti coding making long term maintainability difficult,
* And ultimately, and repetitively, to maintain best practices in object-oriented programming concepts.

## Data Structures and Data Flow

Each project will be required to have data flow. This may be in the form of save game files, user preferences, system settings, data logging, etc. Each group will be required to design this part of the project in advance. The following considerations must be made:

* What type of data source will be used (flat file, JSON, XML, Relational database, no-sql database)?
* What are the fields, or objects, that will be stored in the data source?
* What are the access permissions or other security considerations?
* How is the file connected to, accessed, read, and written to from within the application?
* How are data sources linked to coded classes and how is OOP concepts maintained during data access.
* How is the data stored within the project (generics, collections, … etc.)

The data structure and data flow should be presented using both a written document in addition to level 1 and/or level 2 data flow diagrams.

## Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Incomplete or Unacceptable | Major Improvement Required | Complete but with little thought or analysis | Complete with minor issues or inconsistencies | Outstanding |
| System Architecture and Asset list | 0 | 1 | 2 | 3-4 | 5 |
| Citations and References | 0 | 0.5 | 1 | 1.5 | 2 |
| Class Design | 0 | 1 | 2 | 3 | 4 |
| Class Diagrams | 0 | 1 | 2 | 3 | 4 |
| Data Sources | 0 | 0.5 | 1 | 1.5 | 2 |
| Data Flow Daigrams | 0 | 0.5-1 | 1-2 | 2-2.5 | 3 |
|  |  |  |  |  | /20 |