# CS 2340 Sprint 2

Rooms, Scoring, Singleton Design, and SSDs
*Due Date: See Canvas assignment*

## Background

For the iteration of this project, your team will be implementing some of the core functionalities of the 2D Dungeon Crawler game: player scores and tiles. Scores for each player's attempt should be shown on a leaderboard. For the tiles, there should be at least three distinct types representing different rooms.

## Purpose

This sprint expands upon the work started in Sprint 1. Your team will be able to work together on creating the basic game screens for the project, where future sprint requirements can be implemented.

When designing the System Sequence Diagram (SSD), your team will learn how important analysis is when developing large systems. Diagrams like SSDs are very helpful tools for making decisions about how to inspect your code.

Additionally, following software engineering principles like adhering to a specific architecture for application development and incorporating design patterns will allow your team to learn the importance of logical code organization, especially for larger-scale projects. Testing will occur alongside development, and you will be responsible for writing unit tests to verify your implementation's functionality.

## Tasks

For Sprint 2, you are asked to create and submit to Canvas multiple design deliverables along with demoing the functionality implemented. For the design deliverables, you will perform object-oriented analysis (OOA) on the new sprint features. This will include updating the previous sprint's domain model, producing a set of System Sequence Diagrams (SSDs), and providing evidence of your team using a design pattern and implementing the Model-View-ViewModel (MVVM) Architecture. For the implementation portion of this sprint, you will create three new tiles corresponding to the rooms in your game. You will also create a score that will temporarily be based on time and a leaderboard on the ending screen to display all scores recorded so far. Other than the requirements outlined below, and that you must use Java (not Kotlin), the details of your implementation are up to you. Your application implementation and functionality will be graded during a demo, which will occur the week after sprints are due; see the class schedule for specific dates.

### Design Deliverables

There are five design deliverables that are required in this sprint: a revised Domain Model, System Sequence Diagrams (SSDs), Design Pattern Evidence, and MVVM Architecture Evidence. These are due **as a combined single PDF via Canvas submission (check course page for official date)**. For diagramming, we recommend using draw.io.

#### Revised Domain Model

1. Update the list of nouns from your previous domain model to include the features from this sprint (an example might be leaderboard). You must include at least two new nouns. Highlight the new nouns in some way to distinguish them from Sprint 1 nouns.
2. Update classes and attributes with your new list of nouns and list the identified classes and attributes on your submission PDF.
3. Revise your Domain model with the newly added classes and attributes.
4. Add the new classes to your Domain model and connect them with at least one other class using associations.
    a. Include multiplicities for each association, one on each side of the association.

Please explicitly categorize any new nouns as either classes or attributes somewhere in your deliverable *in addition* to including the domain model. The submission of the Domain Model itself does **not** suffice for the inclusion of the listed and categorized nouns.

## System Sequence Diagrams

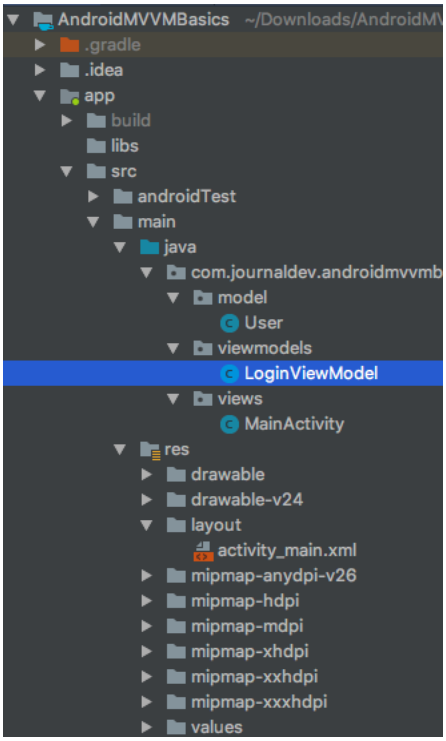
1. Each team member should take a use case from Sprint 1's use case diagram or create a new use case based on Sprint 2 features. The team member should then turn this use case into a UML System Sequence Diagram (SSD). Even though teams weren't required to write out the use case in Sprint 1, make sure to consider both the main success scenario and alternate success scenarios when creating the SSD. Please indicate the name of the team member who wrote the given SSD.
2. For each SSD, ensure that it illustrates the dynamic interactions between the user of the use case and your project's system. Keep in mind that you should treat the system (application) as a black box.
3. Each diagram must contain **at least one ALT, LOOP, or OPT structure**, so choose your use case carefully. You may optionally reference another SSD using the Ref fragment.
4. Collect all team members' SSDs and append them to the same pdf used for the domain model.

## Design Pattern Evidence

1. Submit screenshots of your code abiding by the Singleton pattern for the implementation of the leaderboard feature.
2. The screenshots should capture a clear view of how the singleton design pattern was used for the leaderboard, including all relevant classes.
3. Write a paragraph description of how your code properly utilizes the singleton design pattern.

## MVVM Architecture Evidence

1. Your project architecture should follow the Model-View-ViewModel (MVVM) architecture pattern. Classes should be distinctly split into Model, View, and ViewModel classes. Submit screenshots of how your project follows this architecture pattern.
2. The screenshots should capture a clear view of the Model-View-ViewModel Architecture in your project's file structure (in other words, your corresponding Model, View, and ViewModel classes). An example screenshot is displayed below. Your folder structure does not have to follow the example exactly.




Ref1

3. List out your classes and xml files and categorize each of them as Model, View, or ViewModel.
4. Write a paragraph description of how the team utilized the MVVM architecture in Sprint 2.

## Project Management

We will no longer require you to submit screenshots of the project management tool (Trello, Jira, Rally, etc.) you have been using in Sprints 0.5 and Sprint 1. However, you should continue using the project management tool for the remaining sprints to manage sprint tasks among your team.

## Implementation

In this Sprint, you will be setting up basic tile structure representing different rooms in your game map. You will also create a scoring system for the player, including displays on the game screen and on a leaderboard on the ending screen. In addition to these features, you will be required to implement the Singleton Design pattern for the leaderboard. The following are the requirements:

1. The game screen should now show a map with the starting room and a corresponding tile set.
    a. There must be at least 3 visually distinct and different rooms.
    b. Each of these rooms should be implemented as separate screens.
    c. Have a temporary next button to navigate to the next screen. This button is temporary as it will be removed once more game functionality is implemented in future sprints.
2. Implement score feature that tracks player performance.
    a. The score should be visibly displayed on the game screen.
    b. The score should be updated in real time as it changes.

    c. For now, temporarily base the score off time (ex: score decreases from a set starting amount until it reaches 0). How score should be implemented in the future will be left to the team.
        i. In the future, scores can be based off multiple factors like damage done to enemies, damage taken from enemies, number of missed attacks, time spent to clear the game, etc.
        ii. Score can increase (ex: increase score if player destroys an enemy), decrease (ex: decrease score the longer the player takes to clear the game), or both.
        iii. Teams are free to choose how exactly they implement scores as long as it takes multiple factors into account and can be explained in Sprint 5.
3. Implement a leaderboard that keeps a history of previous scores using the singleton design pattern. This leaderboard should:
    a. Display scores in descending order
    b. Display each attempt made by the player
        i. Each entry on the leaderboard should visually display the player's name, score, and the date/time of the attempt.
        ii. Attempts do not need to be saved across game instances (i.e. closing and reopening the app). The leaderboard should just display attempts made during the current instance through playing through the game, restarting the game using the restart button, and repeating.
    c. Be located on the ending screen
4. Add a restart button to the ending screen to bring the user back to the start screen
    a. The player should be able to play the game again from the start like normal
5. *(Optional, but highly recommended!)* Make the player a singleton. This will make development for future sprints more manageable as the project's complexity grows.

As mentioned in Sprint 1, artistic liberty is permitted for groups to customize their project theme. For example, the game can be implemented in a Sci-fi theme where the player traverses through rooms such as the medical bay, helm, and crew quarters of a spaceship. You are also free to add functionality as long as you meet the requirements.

## Testing Requirement

Starting this sprint, your team will be required to create 2 unit tests per team member. Make sure you create and develop your project with unit testing in mind. It may be wise to write some unit tests for the implementation requirements to practice for future sprints. For Sprint 2 **only**, you are allowed to construct unit tests that test functionalities from Sprint 1 and/or Sprint 2. Some example unit tests that you could prepare from Sprint 2 requirements are:

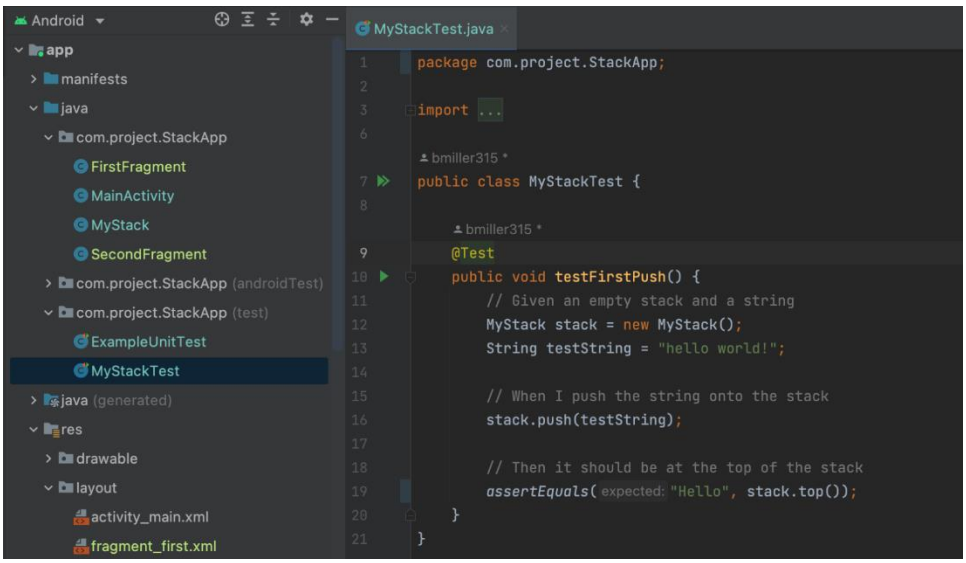
1. The player's score does not decrease any further once it reaches 0 (if your team decides to implement score this way).
2. Leaderboard updates the descending order of scores when a new score is added.

Similarly, example unit tests from Sprint 1 are:

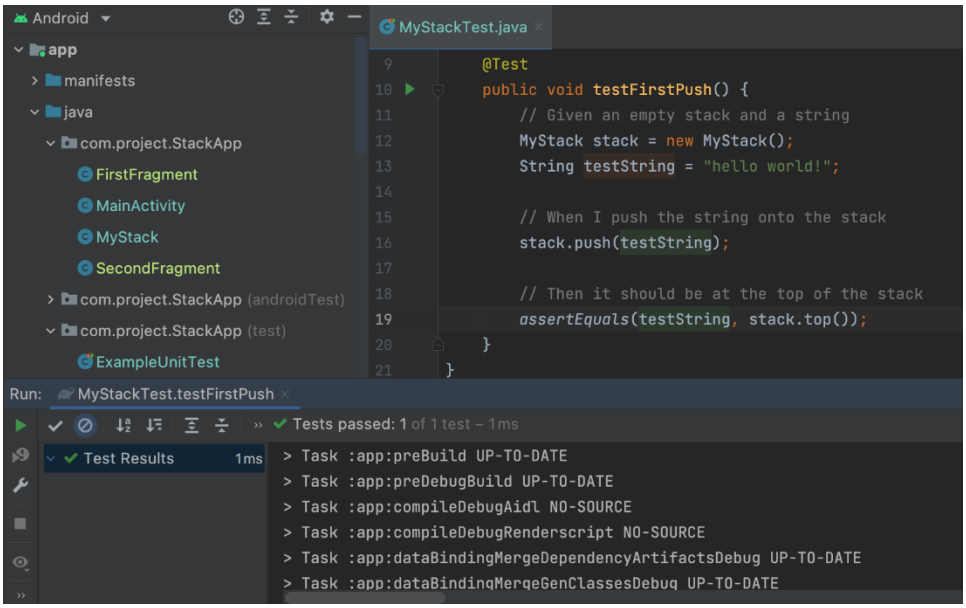
1. Detection of whitespace-only, null, and empty names.

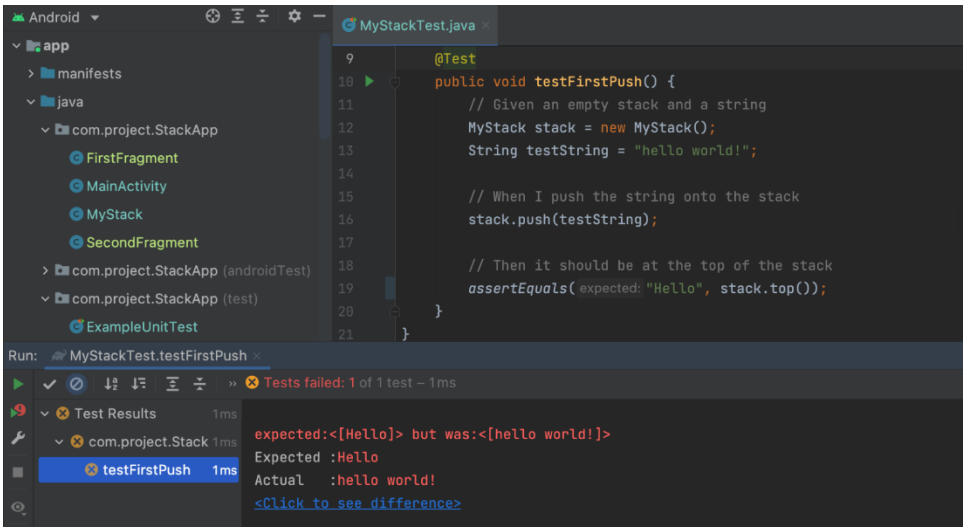
2. Player starting lives are different based on the chosen difficulty.

JUnit 4 has already been added to the dependencies in the Android-Gradle template we provide for Sprint 0.5. To write a unit test, navigate to the subpackage marked as “(test)” in Android Studio. Annotate each unit test with “@Test” as shown:




Unit tests can be run individually or altogether using the green play button near each test method/class:






For further knowledge on Android testing, refer to the documentation.

***Note:*** The UI of your team's game should be separate from the game's functionality/logic. These tests should not be composed of UI mocking, but instead, your team's tests should be validating your game's logic and event handling. **If you create mock test cases, you will receive 0 credit for those tests.**

## CheckStyle

For checkstyle requirements for this sprint, you can refer to the details mentioned in Sprint 1 for the set up. Make sure you follow the rules for the correct styling of your project's code.

## Sprint Tagging

Tags are a way of marking a specific commit and are typically used to mark new versions of software. To do this, use "git tag" to list tags and "git tag –a tag name –m description" to create a tag on the current commit. **Important: To push tags, use "git push origin –tags"**. Pushing changes normally will not push tags to GitHub. You will be asked to checkout this tagged commit during demo. This is done with "git fetch --all --tags" and "git checkout tag name". **You will be required to pull this tag during the demo. If you forget to tag your commit or tag your commit late, you will be penalized with –5 points for each day late.**

## Additional Resources

Under this section are some resources we think will help you and your team streamline the game creation process. If you have any questions about these resources, contact your TAs through Ed or office hours.

1. TA website: https://github.gatech.edu/pages/gtobdes/obdes/
2. Ed: https://edstem.org/us/courses/42802/discussion/
3. Android Studio Video (located on your section's canvas lecture page).

## Demos

Your implementation of sprint features will be graded through an in-person TA demo. Demos will be held in CCB 267 the week after the sprint is due. Please sign up for a demo slot in Microsoft Bookings. Note that not all demo slots may be open at sprint release.

## Summary

You will be graded according to the successful and correct completion of the design deliverables and implementation requirements above. Groups are required to demo to receive credit for the features they have implemented. This will be done during TA office hours after the due date of the design deliverables. TA demos will be able to be booked through Microsoft Bookings**.** You will submit your design deliverables as a combined PDF via the Canvas assignment. **You should also include a link to your GitHub project repository**. Your repository must be set to private. When you sign up for a demo, ensure that the shared TA GitHub account has been added to your repository **and you have tagged the code you intend to demo.** Points may be deducted if these guidelines are not followed.

## Academic Integrity

### Note on Plagiarism and the Use of AI Tools

We understand that technology and tools, including Artificial Intelligence platforms, have become readily accessible and can be valuable in various scenarios. However, students must be cautious about their usage in academic settings.

Please refer to the course policy regarding the use of AI tools for your projects. Using AI to generate or copy content for your project is not considered collaboration. Leveraging AI to produce work that you present as your own, without proper citation, is likely crossing the line into academic misconduct. It's essential to maintain integrity in all academic undertakings. If in doubt, always consult the course guidelines or reach out to the instructor on Ed Discussion for clarification.