# Pisces

# Purpose

The purpose of this document is to provide evidence for the different roles of the team members.

# Introduction

*WordSweeper* is a multi-player word puzzle game in which players can create or join games hosted on a server to compete with other players, by selecting letters from a board that form words to gain points. The game also implements a local practice mode for the player to get familiar with the rules. As a team focus on the client side, we went through the full development process of use cases definition, UML modeling, storyboarding, coding in collaboration with a server team, and testing to ensure correct all end-to-end gameplay scenarios.

# Team organization, members, and responsibilities

Throughout the semester, our team has held regular (at least two meeting per week) group meetings in labs or the library. Our meetings in the design stage of our project were used to discuss brainstorm and discuss the design specifics of the game, such as the user stories, use cases, UML and storyboards. Once the design is figured out, we used meetings in the coding and testing stages to report progress and gather feedback, and make sure coding tasks are assigned to different team members and executed in time.

At the beginning of each meeting, we clarified the goal and agenda of the meeting. During the meeting, every team member actively participated the brainstorming, and contributed to the generation and clarification of the idea. At the end of the meetings, each team member was assigned a task to record the discussion results on paper. We did not choose a team leader, because we wanted to encourage participation from everyone, and make sure we all agreed on the decisions we made.

In the design stage, each of the subtasks deliverables (i.e. use cases, storyboard, and UML class diagram) were the results of group discussion. However, we did assign different team members to write up the results. Specifically:

* **Use Cases**: Each team member wrote a few of the user cases, and Yen-Chang Hsieh helped finalized it.
* **Storyboard**: Drawn by Yuchen Liu and Kai Zou using WindowsBuilder. The rest of the team members helped validated and finalized it.
* **UML Class Diagram**: Drawn by Mi Tian using Astah. The rest of the team members helped validate and finalized it.

In the implementation stage, every team member contributed to coding and testing of our project. In general, our approach is to separate functionality into areas and assign to different team members to work in parallel. For example, the first features we implemented were the different JPanels (views) as the barebones of the project. Each team member worked on a different panel to bring the initial versions up. We then reviewed each other’s work and kept adding more features to complete each of them. In this stage, we also emphasized on real time communication using WeChat or meeting. This happened when we wanted to update the team of new code, felt confused of the responsibility, or when a merge conflict or bug was introduced. The responsibilities of each team member is listed as below:

(order is by first name…)

* **Kai Zhou:**
* **Mi Tian:** Help implement “Join Game”, update score, reset board and reposition board. Test implementation of most view classes and some controller classes. Implementation of “FindWord” request and response. Make tests fully automated and fix the bug for the tests. Help team ramp-up on Git, and help with bug fixes and merge conflict resolve.
* **Yanzhang Xie:**
* **Yuchen Liu:**

# Process

## Design Stage

In Phase 1 of the project, the team had two meetings per week, with each meeting lasted about three hours.

At first, each team member wrote several use cases. Then we combined all the use cases together as well as some questions we had during the process. By this stage, we had 14 use case according to the gold user stories that professor posted. Some of the question we can come up with a solution together during the meeting, as for other questions that we cannot solve, we asked during the scheduled meeting time with professor. After discussed with professor, we modified our use cases. Right now, we have 12 use cases, we spend a lot of time on this part and UML part below, because they plays a basic and vital role in the design process.

Besides use cases, we also have drawn UML class diagram. First, we identified the major classes together. We had a lot of discussion here about whether each class should exist, what is the relation between two classes. Then we drafted our first version of UML class diagram, and cross-compared it with user stories and use cases to see if there was anything we missed. After got the suggestions from professor, we modified the UML diagram many times. We believe this part is worth spending time on, since we need to write code based on this diagram.

Based on the use case and UML class diagram, we drew our GUI and wrote our storyboard. Because we had different versions of use case and UML diagram, we also wrote different versions of storyboard. To begin with, we utilized Lumzy, which is an online GUI design software. However, we found there are some disadvantages, such as it doesn’t support users edit it collaboratively. After got the suggestion of using WindowBuilder from professor, we changed our tool, and used WindowBuilder to mock up GUI.

We have cooperated very well by far. We figured out most of design specifies through brainstorms and discussions in the group meetings. Before every meeting, we set up an agenda about what design questions to discuss. And at the end of the meeting, we assigned take-home tasks to each team member to summarize the discussion result in the form of the documents. We shared our documents on Google Drive so everyone can view and edit, and we also used WeChat to ask questions and have discussions outside of the in-person meetings.

## Implementation Stage

In the implementation stage of the project, we started by converting our visual design into actual Java JPanels. The team members were assigned to different panels (CreateGame, JoinGame, Practice, MainMenu, etc.) and worked on coding the first version in parallel. Once the first version were checked in, we started reviewing each other’s work and added more functionalities to the code to make them more complete. This was also when we started implementing the models and controllers, and hook them up with the existing panels. Our work on the board UI in this stage were mostly tested in the practice mode, because we had not connected with a real server backend yet. We also wrote some placeholder code to simulate how to send request and process responses but they were not tested against real network data.

While waiting to pair with a server team, we also started writing test cases to verify our models, views, and controllers. Our eventual goal of testing was to catch code defects, and reach 80%+ coverage for all classes. We were able to achieve most of these using the mock server and hard-coded response messages.

In the next step, we paired with another team that provided us a backend server using the same protocol. We spent time testing the connection and made sure our requests were sent using the right format, and verified the response string were the correct format and content. As the communication between client and server are implemented and tested, we replaced our placeholder code in various controller classes to send real request based on user input, and processed the real response to update the UI. We kept iterating this and fixing bugs until we reached a good state for the game to play correctly in all use case scenarios, such as joining public, private or locked games, and request reset board to update UI for all clients, and so on.

# Tools

**Lumzy**: We used Lumzy in the beginning to mock up our GUI. It was a very helpful and easy-to-use tool. The only downside was that online synchronization was not available so we could not edit it collaboratively.

**Astah**: We used Astah to draft the UML class diagram. It is a simple tool and very easy to learn and use. And it served our purpose well enough as a free UML editor. It has very friendly UI to draw the classes and the relationships between them.

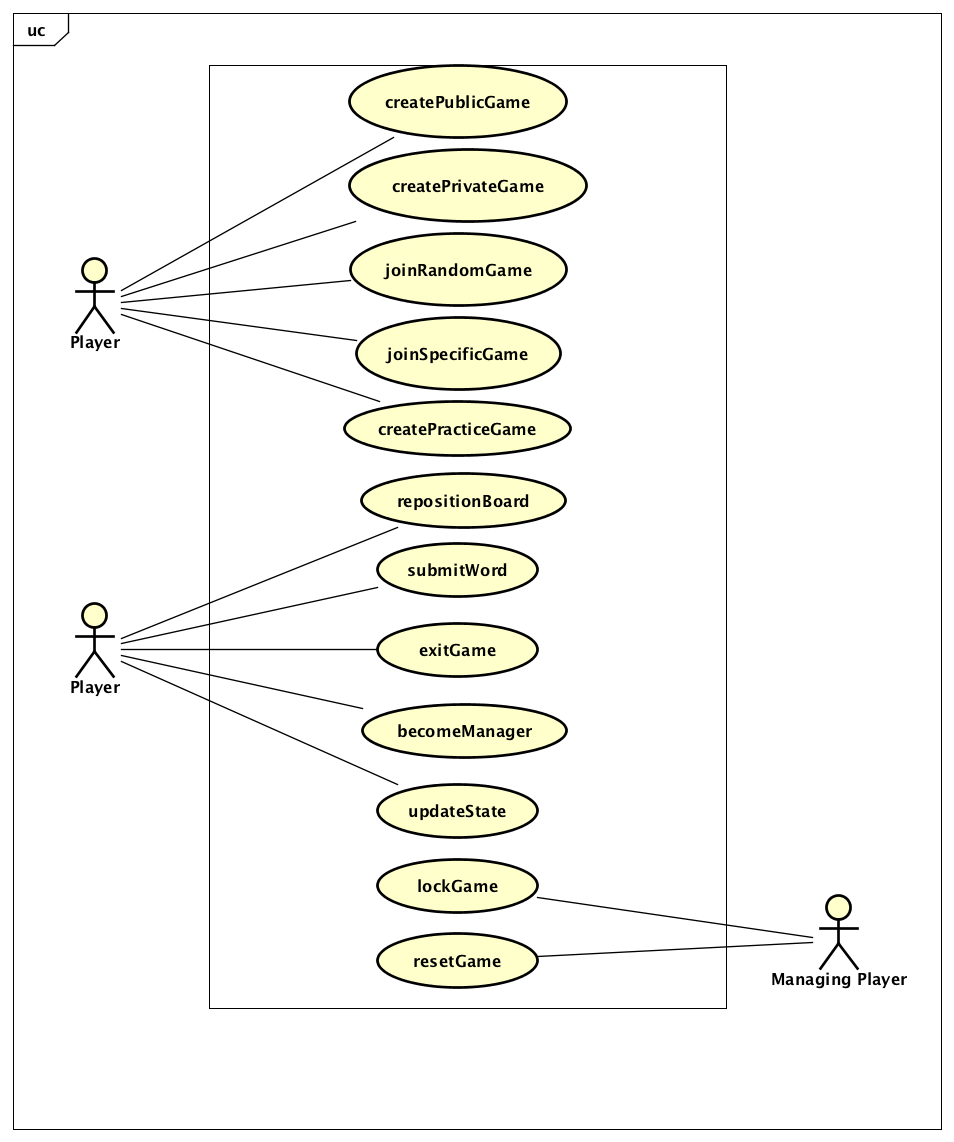
**WindowsBuilder**: Finally, we chose WindowBuilder in Eclipse to mock up our GUI. We expect it to provide all the components we needed for WordSweeper, including the list, the canvas and buttons, and it did meet the expectation.  By far, we think WindowBuilder is better than Lumzy, because as a plug-in to Eclipse, WindowBuilder simplifies the development of a Java GUI application.

**JUnit**: We used JUnit to implement test for our model, view, and controller classes. We found JUnit to be very intuitive to use, and very helpful to indicate the code coverage.

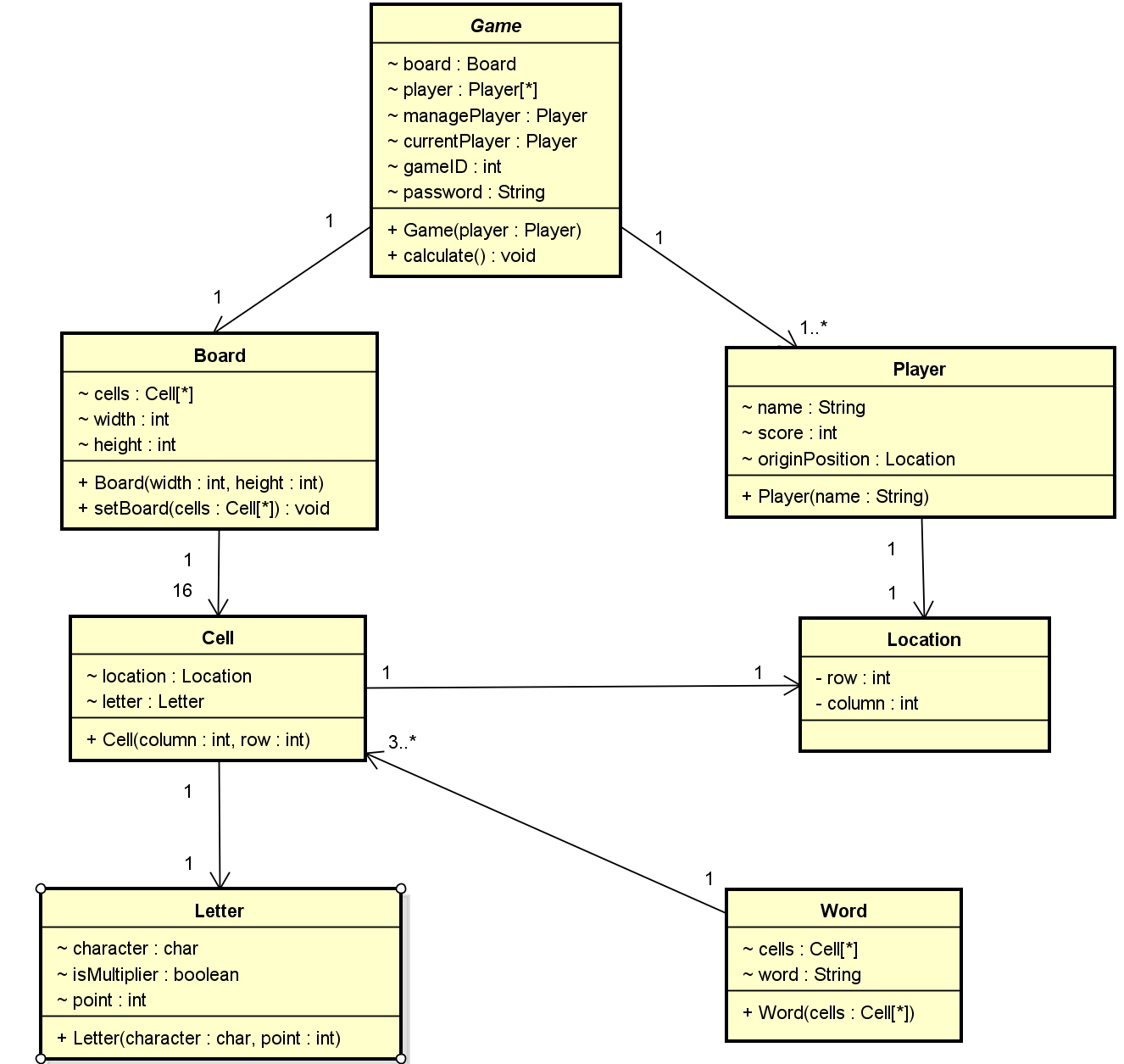
**Git and GitHub**: We used GitHub as the host of our code and Git as our version control system. We had some difficulty in the beginning for managing the collaborating code flows as we were not very familiar with it. We had to sit together once or twice to get code conflicts resolved. But once we are ramped up it became very intuitive to use.

# Accomplishments

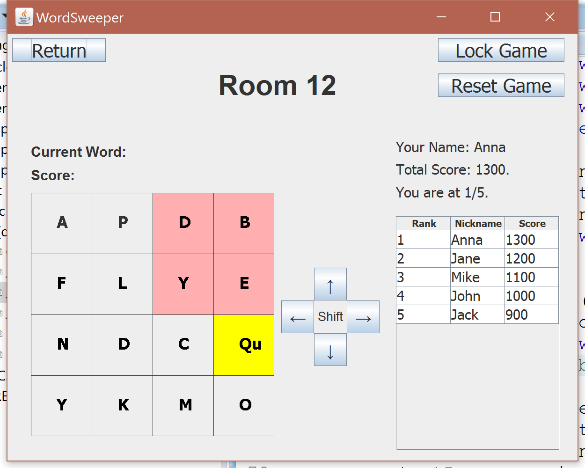
We came up with 12 use cases (see Picture 1 below) satisfying the gold user stories, and drafted the UML class diagram (see Picture 2 below) with entities class, variables, operations and the relationship between the classes. We also drew the storyboards (see Picture 3 below) based on our design. These will be posted in the deliverables section.



Picture 1 Use Cases Model



Picture 2 UML



Picture 3 Game View of a Managing Player in a Public Game

**We should summarize what our game can do in the final version, and put most recent pictures of the game here….**

## Deliverables

1. **Use cases:** We wrote 12 use cases for the game’s client side, which represents all possible sequences of the interactions between *WordSweeper* game and player.
2. **UML:** The UML includes the entities class, the classes’ variables, operations and the relationship between the entities.
3. **Storyboards:** We wrote storyboard based on the 12 use cases from the first deliverable. In the report of storyboards, for each use case, we gave a brief description of how does player interact with GUI together with a snapshot of GUI for each step.
4. **Code implementation:** We wrote 8 mode classes, 7 view classes, and 25 controller classes to implement our game functionality. We also wrote X test cases to a final code coverage of 100% for models, 85.2% for views, and 85.9% for controllers. Our code communicates with the server backend and allow the user to play in practice mode, and create or join game hosted on the server. Our game also allow managing users to lock the room or reset the board, and every user to shift their board and see the score of every other player.

# Reflection

## What worked, what didn’t work

All the things worked well for our team, because we made every decision as a team. We may had different opinions in the beginning of the discussion, but since we are a team that knows how to work together, our final decision will be a decision that combines all the advantages of our opinions, which is agreed by all the team members. In the coding implementation stage, our collaboration worked out well. We were able to separate the work and make progress together in parallel, and also review and help each other’s work by communicating in real time. We engaged the server team a little late, but since we have coded the client according to the communication protocols, the adaption was pretty smooth and did not make us miss the deliverable. We also started writing test cases very early and kept adding more tests, this helped us to verify our product as we add more functionality. The only thing we think that slowed us down a bit was code merge in the beginning, but once we became familiar with Git and GitHub it went very well.

## Our biggest mistake

Our progress has been good and we did not make much mistake by far. Probably there are a few small mistakes caused by us working ahead a little bit. For example, we started drawing GUI using Lumzy before the professor suggested WindowsBuilder on the course website. We did repeat the process in WindowsBuilder afterwards which cost a bit of extra time. But drawing it early also gave us a better understanding of what we are going after and helped making sure everyone was on the same page.

Another mistake we made was that we cared too much for unnecessary implementation details at this stage. The UML class diagram included parts that was suggested to remove by the professor because of this. We learnt that at the early stage of software design, we should focus on user stories and high-level scenarios, and leave implementation thinking for later.

**What do you guys think…?**

## Changes we would make

At this point, our UML just hold entities which we cannot form the flow of the process base on the uses cases. We think it probably will work after we add control objects and boundary objects in next task. Our final code implementation created a fully functional game as specified by the course project. However, our GUI was not very artistically polished. Maybe we can replace our standard Java UI elements with beautiful pictures if we have more time. Our current implementation also does not support resizing the window to resize GUI elements. That can also be improved if we have more time.

# Lessons learned

## Things we learned

At this point, all of us are enjoying the class a lot. We learnt to use Astah and WindowsBuilder, and how to reach agreement in brainstorms and discussions. The project helped us improved our teamwork skill. And it was also a good way for us to practice creative thinking and communicating ideas with others. A good lesson we learnt is not to be shy to ask questions, which can really help you to stay in the correct direction, if you do not clearly understand the design of the project. In the coding stage of our project, we learnt a solid understanding of how MVC and other design patterns work, about communicating with backend servers with protocols, and about the importance of writing test cases. We also learnt very practical knowledge of how to use Git to do version control, especially about how to handle code merge conflicts.