

CSCI 3100 Software Engineering

Project Requirement Specification

16 January 2023

1. Objective

The objective of this course project is to practice what you are learning in this CSCI3100 Software Engineering course by specifying, designing, implementing, testing, and documenting a typical software engineering project (e.g., a web-based client-server application, or a software game application). The project serves as a vehicle to sharpen your knowledge in Software Engineering and to develop your relevant skills. This course project also introduces students to teamwork and project management, which are keys for successful large-scale software development. Besides, you can take the project opportunity to develop a modern software product (such as a mobile-web application), as if you are working for a major high-tech firm or founding an IT start-up company.

Generally, the project consists of two parts: (1) software engineering project documentation, and (2) software production (including specification, design, coding, and demonstration). On the one hand, the documentation reflects the process of your designing, refining, and testing your project. The documentation is required to be conducted and submitted progressively according to the grading criteria. On the other hand, the software production reflects how well you produce your system. Your implementation should be managed by a GitHub repository and graded on the Demo Day, and your complete code should be submitted after the demo.

2. Project Grouping

Each project group is composed of 5 students for the whole duration of the project. Some groups may have four members, but no group should have six or more members. All students in a group work together on the same project based on the project requirements defined in the remainder of this document. By now, you may have chosen group members by yourselves. If so, please visit the front page of the course website, where there is a link for Project Group Registration. Sign up your project group accordingly. Note that you can sign up for a group only when you have *three or more* members in the group already. At the end of project grouping date (20 Jan), we will randomly assign the remaining students to form additional groups or assign them to existing groups which include less than 5 (i.e., 3 or 4) members. Once the group is assigned, you should remain in the group and work with your team members closely for the entire project duration.

3. Project Requirements:

The goal of the project is to go through the whole process of software development with software engineering techniques. We provide four application choices. Each group can only choose *one* application as the topic of the project. The whole project is composed of two parts: (1) documenting

the application, and (2) implementing the application.

3.1. Documenting the Application

You are required to use specification and design techniques (e.g., requirement analysis, DFD, UML) taught in the lectures to provide a comprehensive documentation for your design and implementation. In this project, you will provide four documents, a high-level design document, a DFD specification document, a UML specification and UI design document, and a testing document.

- **High-Level Design Document**
The high-level design document should contain two parts, i.e., project overview and **system architecture**. In the project overview part, you should describe basic requirements and all advanced features of your applications. In the system architecture part, you should describe your vision on the techniques that you will employ in your project. Please refer to **Appendix 2** for more detail.
- **DFD Specification Document**
In the DFD specification document, you should use data flow diagram (DFD) techniques taught in the class to specify your application.
- **UML Specification and UI Design Document**
In the UML specification and UI design document, you should use UMLs to describe key components of your system. The UML technique will also be described in the class. Meanwhile, for the UI design part, you should design several views for your application and describe the objects and actions of the view. Please refer to **Appendix 2** for more information.
- **Testing Document**
The testing document should contain two parts: (1) a test plan, and (2) multiple test cases. First, you should describe the components covered in the testing process in your test plan. Second, multiple detailed test cases should be described to test the key functionalities of your project. The design of test cases includes black box testing (required) and white box testing (optional), which will be covered in the class.

3.2. Implementing the Application

We provide four applications for your selection, i.e., a *simplified Twitter*, a *course selection system*, a *Pac-Man*, and a *Gobang*. For each application, we provide several basic requirements and multiple advanced functionality suggestions. The detailed application description and requirements can be found in **Appendix 1**.

The basic requirements account for 70% of the implementation part of the project. While the detailed basic requirements are different among different applications, there are still several common points. For example, you need to provide a clear user interface (UI) for users to easily understand and use the application. You also need to employ database techniques to store user and application data. You

are expected to implement all the basic requirements and show your implementation in your demo presentation. We provide checklists in **Appendix 4** containing the grading details of basic requirements.

The advanced requirements account for 30% of the implementation part of the project. We provide some advanced suggestions for each application in **Appendix 1**. You can try to implement these advanced features. You can also put as many fancy and interesting features in your applications as possible. For instance, you could consider making your application more comprehensive by providing more application-specific features. You can also consider integrating more advanced techniques in your application by integrating cloud techniques or integrating with mobile devices.

3.3. Other requirements

When designing your application, the most important project feature to keep in mind is that the software product you will develop should require a reasonable programming effort. No joint work over any technical aspects of the project is allowed between any two groups/teams. Any problem about the project requirements should be directed to the tutors through electronic mails, newsgroup discussions, or tutorial sessions. The reason for this policy is to enforce team separation for proper credits. This project should be considered as if there is only one single team, namely your team, being responsible for your whole project development. No plagiarism is allowed regarding any aspect of the project. Reusing existing designs or codes as part of your project (such as those from the open-source projects) is allowed, but you need to document the reused code clearly. We will also employ professional tools to verify the percentage of existing codes in your project. Note that you should not reuse existing code excessively. Your project mark will be deduced significantly if the percentage of reused existing code is excessive.

4. Project Phases:

There are five project phases described as follows:

(1) High-Level Design Phase (2 weeks)

In this phase, each project group will prepare and submit a high-level design document to provide high-level descriptions of the functionalities, features, and architectural design of your application. Project introduction, architecture diagrams, and brief descriptions of the key system components should be provided. Feedbacks will be provided on your high-level design, and you should consider and possibly revise the project goals before going on to the next phase. You are required to submit your project high-level design document by **23:59:59 of 4 February (Saturday)**.

(2) DFD Specification and GitHub Repository Creation Phase (3 weeks)

You need to complete two tasks in this phase. First, you need to specify your application functionalities with data flow diagrams (DFDs). You need to specify all basic requirements and the advanced features you want to implement in this document. Second, you will work as programmers to implement your own design and collaborate using the **git** version control system. You need to get familiar with the **git** version control system. At the end of this phase, you are required to 1) submit the DFD Specification Document, and 2) create a code repository on GitHub. You should

accomplish these two tasks by **23:59:59 of 25 February (Saturday)**. Your code repository will be handed in by providing the URL of your GitHub repository. No implementation is required. For more information on the **git** version control system and GitHub, please refer to **Appendix 3**.

(3) UML Specification and UI Design Phase (4 weeks)

In this phase, you should be implementing your application. We expect you to use UML diagrams to specify your application and refine your UML diagrams during your implementation. Also, you should provide a UI design document that describes the user interfaces of your application. You are required to submit a UML Specification and UI Design Document by the end of this phase. The deadline is scheduled on **23:59:59 of 25 March (Saturday)**.

(4) Project Demo Phase (2.5 weeks)

In this phase, you are completing your project. You will need to make a demonstration of your complete application. Project Demo Day is scheduled on **13 and 14 April**. Detailed project demo arrangements will be announced on the course website, and the demo schedule (15 minutes per group) will be signed up accordingly (please note the news on the course website).

(5) Testing and Final Commented Code Phase (3 weeks)

In this phase, you are expected to conduct testing on your application. You should describe the test plan in your testing document. Detailed test cases should be included to test key components of your application. The final code is also required. Your final code should be self-contained and working. The code should also be commented as detailed as possible. A README should be included to describe your code repository. You are required to prepare and submit your testing document and final code by **23:59:59 of 6 May (Saturday)**.

5. Grading Criteria:

The followings are the project schedule of different phases:

Phase Deliverables	Weightings	Durations	Due Date
0. Project Assignment Team Formation	--	--	16 Jan. (on Web) 20 Jan. (23:59:59)
1. High-Level Design Document	5%	2 weeks	4 Feb. (23:59:59)
2. DFD Specification Document and GitHub Repository Creation	10% (8%: DFD Specification Document, 2%: GitHub Repository Creation)	3 weeks	25 Feb. (23:59:59)
3. UML Specification and UI Design Document	15%	4 weeks	25 Mar. (23:59:59)
4. Project Demo	60%	2.5 weeks	Demo Day: 13 and 14 Apr. (two full days)
5. Testing Document and Final	10%	3 weeks	6 May (23:59:59)

Commented Code	(8%: Testing Document, 2%: Final Commented Code)		
Total	100%	15 weeks	

5.1. High-Level Design Document (No more than 5 pages)

The high-level design document should be written in text font Times New Roman and size 11. The main body of the design document should be **no more than 5** pages. Marks will be deduced if the format requirements are not met. The document will be graded upon the clarity of the documentation. Please refer to **Appendix 2** for more information.

5.1. DFD Specification Document and GitHub Repository Creation

The DFD specification document should contain both basic and advanced features of your application. The document should be written in text font Times New Roman and size 11. The main body of the document should be **no more than 10** pages. The creation of your GitHub code repository will be graded on the availability of your GitHub repository. The implementation of your project is **not** required.

5.2. UML Specification and UI Design Document

The UML specification and UI design document should contain UML diagrams and UI design descriptions for your application. The document should be written in text font Times New Roman and size 11. The main body of the document should be **no more than 20** pages. Please refer to **Appendix 2** for more information.

5.3. Project Demo

The project demo will be graded upon the functionalities of your application. The grading criteria are listed as follows:

- Basic Project Features (70%):
Different applications have different basic feature requirements. The grading of the basic requirements is based on the checklist provided in **Appendix 4**. For each item in the checklist, you can get all the marks of the item as long as you have completed its requirements.
- Advanced Project Features (30%):
The advanced project features will be graded on the comprehensiveness of your application and the advanced techniques you have employed. You can show as many interesting features in the demo as possible and your grades will be considered accordingly.

5.4. Testing Document and Final Commented Code

The testing document should contain a general test plan and detailed test cases for your application. The document should be written in text font Times New Roman and size 11. The main body should be **no more than 15** pages. Please refer to **Appendix 4** for more detail. The final commented code should be handed in by pushing the code to your GitHub repository. You should include a detailed README on your GitHub repository describing your application and the requirements of running

your application. The code will be graded based upon the availability of the README and the readability of your code.

Although generally, the project grade is for the whole team and will not be assigned individually to the members, each team member must be aware that a major part of his or her final project grade depends on teamwork. Failures to cooperate with other team members and to invest an equitable amount of effort can lead to undesirable outcomes, particularly when other team members raise complaints about the non-participating members. On the project Demo Day, if there is any complaint about free-rider(s), please raise the case right after your project demo is done. We will verify with all your team members regarding the validity of the complaint.

6. Submission

There are four report submissions (i.e., High-Level Design Documentation, DFD Specification Document, UML Specification and UI Design Document, and Testing Document) and two code submissions (i.e., GitHub Repository Creation and Final Commented Code). The submissions need to meet the following requirements:

6.1. Report submission

Each project group should submit the softcopy of the report and the VeriGuide recipient to Blackboard System before the deadlines. The followings are the required names of the attached documents of different phases:

- “Group** High-Level Design Document”
- “Group** High-Level Design Document VeriGuide”
- “Group** DFD Specification Document”
- “Group** DFD Specification Document VeriGuide”
- “Group** UML Specification and UI Design Document”
- “Group** UML Specification and UI Design Document VeriGuide”
- “Group** Testing Document”
- “Group** Testing Document VeriGuide”

Please replace the “**” with your group ID.

6.2. Code submission

ALL your project stuff (including source code, images, flashes, databases files, etc.) should be maintained with **Git**. Git actions play an important role in evaluating your project coding phases. You should take advantage of the version control system to support the development and documentation of your project. You **MUST** submit your project via **Git**, and faithfully record your coding activities. We will NOT accept any code submissions via other approaches. Moreover, the tutors will check your version control logs when marking your coding efforts. Please find more information in **Appendix 3** on code submission. Further information will be provided in the related emails or information on the website.

6.3. Late Submission Policies

The late submission and missing Veriguide receipt follow the same policy as assignments. You can find the whole policy on the course [website](#).

Appendix 1: Application Requirements

1.1. Twitter

Twitter is a microblogging and social networking service. Users can post and interact with messages known as “tweets”. The following are the basic requirements and advanced suggestions for a simplified version of Twitter.

Basic Requirements

- **Client-server architecture**
The application should follow a client-server architecture. The server should hold tweets generated by users. Users interact with the application through the client. Note that the techniques of implementing the client-server architecture are not limited. For example, you can design your server as a single process and let clients interact with servers through inter-process communication mechanisms (IPCs).
- **Global Database**
Either SQL database (e.g., MySQL, or SQLite) or NoSQL database (e.g., MongoDB, or Redis) must be employed by your application for storing data.
- **User Interface**
Your application should at least have a clear graphical UI design. The UI should be consistent and easy to understand. Users should be able to use the application without the help of developers.
- **User Management**
Your application should also have basic user management functionalities. Specifically, you should let users sign up and login/logout.
- **Admin User**
Your application should have an admin user that can view all user information and add or delete a user if needed.
- **User Operations**
Your application should allow users to conduct the following operations like a real Twitter:
 1. **Search for users**
A user should be able to search for other users based on usernames or unique userIDs.
 2. **Follow other users**
A user can follow another user by clicking a “follow” bottom. New posts of the followed user will be pushed to the user.
 3. **Like/dislike a tweet**
Each tweet is associated with a like counter. For each post, a user can increase/decrease the counter by clicking the like/dislike bottom.
 4. **Comment a tweet**
A user can leave a comment below a tweet. The comment is available to every when reading this tweet.
 5. **Retweet a tweet**
A user can retweet a tweet with the original user’s information.
 6. **Post a tweet**

A user can post a tweet with images.

7. Show other users' tweets

For a user, the application should show all the tweets that are posted by users he/she follows as the main page.

Advanced Functionality Suggestions

You can consider implementing the following functionalities to your application after you finish all the basic requirements:

- Pretty UI: You can design some cool UI animations.
- Privacy Control: Users can set the visibility of their tweets among their followers.
- User Recommendation: The application can be extended to recommend users of interest for a user to follow.
- Tweet Recommendation: The application can be extended to recommend popular tweets of interest for a user.
- Private Chat: The application can allow users to send private messages to each other.
- Video tweets: You can allow users to post video tweets.
- ...

1.2. Course Selection System

You are required to implement a course selection system for the university. Users can search for courses offered, select and drop courses on the system.

Basic Requirements

- **Client-server architecture**
The application should follow a client-server architecture. The server should hold all the information of courses and valid users. Users interact with the application through the client. Note that the techniques of implementing the client-server architecture are not limited.
- **Global Database**
Either SQL database (e.g., MySQL, or SQLite) or NoSQL database (e.g., MongoDB, or Redis) must be employed by your application for storing data.
- **User Interface**
 1. Your application should at least have a clear graphical UI design. The UI should be consistent and easy to understand. Users should be able to use the application without the help of developers.
 2. You are required to implement two main pages for users, that are, course browsing page and profile page. In the course browsing page, users can search and select courses. In the profile page, users can view all the courses they have selected and can drop selected courses.
- **User Management**
Your application should also have basic user management functionalities. Specifically, you should let users sign up and login/logout.
- **Admin User**
Your application should have an admin user that can view all course/user information and add or delete courses/users if needed.
- **User Operations**
Your application should allow users to conduct the following operations:
 1. Search for courses.
 - a. A course entry should contain course ID, course name, time, place, department, instructor, and capacity.
 - b. A user can search for a course based on course ID and course name.
 - c. A user can search by conditions such as time and department. The system will list all eligible courses.
 2. Select courses.
 - a. A user can select a course by clicking the “Select” button on the course entry.
 - b. If the course capacity is full, i.e., the number of students enrolled in the course has reached the upper limit, the system will prompt that the course cannot be selected.
 - c. When a user successfully selects a course, the course will be displayed on his/her profile page. This information needs to be updated to the database immediately.
 3. Show selected courses.
A user can go to his/her personal page to view all the selected courses.
 4. Drop courses.
 - a. In the profile page, a user can drop a selected course by clicking the “Drop” button

on the course entry.

- b. When a user successfully drops a course, the course will disappear from his/her profile page. This information needs to be updated to the database immediately.

Advanced Functionality Suggestions

You can consider implementing the following functionalities to your application after you finish all the basic requirements:

- Pretty UI: You can design some cool UI animations.
- Concurrency control: When many users access the system at the same time, you need to handle large-scale user requests.
- Course outline upload (admin side) and view (user side): An admin user can upload the course outline for a course, then users can view the outline.
- Schedule display: You can implement a schedule on the profile page that shows the arrangement of all selected courses.
- ...

1.3. Pac-Man

Pac-Man is an interactive computer game developed in the early 1980s. It was one of the most popular games at that time. It is still being played by many people. The first World Championship was held in 2007 in New York City. You can play the game online on several websites:

- www.pacmangame.net/
- www.learn4good.com/games/pacman.htm
- www.webpacman.com/



Basic Requirements

- User Interface (Basic Game Components)

The Pac-Man game has a menu and one component on its main window -- the playfield, and everything should be drawn into the playfield. The necessary contents of the menu and playfield are listed as follows:

 1. Menu items: Menu items named "Reset", "Clear High Score", and "Exit" can respectively restart the game, reset high score to zero and exit the program at any time.
 2. Title screen: A title screen is displayed before the game is started. You can design the title screen freely, but you must include instructions such as "Press any key to start". You may also want to put the how-to-play information within the title screen.
 3. Characters: There should be 3 moving characters in the game, the Pac-Man and at least two ghosts (in arbitrary colors, but should be at least visible). When the characters are moving, they are rendered as animations (changing pictures like .gif) instead of still images. If the Pac-Man stops moving, its image becomes still. If Pac-Man bumps into a ghost, an animation is played.
 4. Maze and pac-dots: The layout of the maze is open to your own design, but it should contain at least four horizontal and four vertical passages. All passages must be one-character wide, which basically means that the characters can only move along the passage. The maze is filled with small dots known as pac-dots except for the initial locations of the characters. For simplicity, power pellets (which enable Pac-man to eat ghosts) are not required in this assignment.
 5. Messages: On the top of the playfield, there are two labels indicating the current score and

the high score. The high score should be kept in a persistent storage (i.e. a file). On the bottom of the playfield, the number of rest lives is displayed as a corresponding number of Pac-Mans on the left. Fruit is not necessary in this assignment.

- **User Management**

Your application should also have basic user management functionalities. Specifically, you should let users sign up and login/logout.

- **Database**

A global database must be employed by your application for storing data.

- **Functional Requirements**

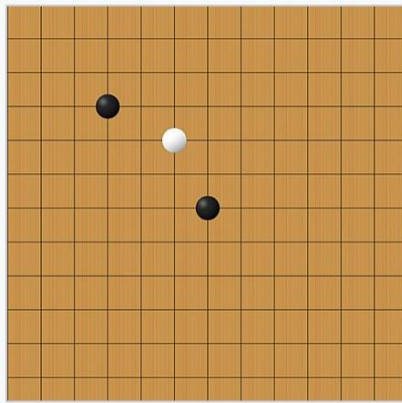
1. Basic gameplay: The player controls Pac-Man through a maze, eating pac-dots. (At least Two ghosts roam the maze, trying to catch Pac-Man (however, no intelligent strategy is needed). If a ghost touches Pac-Man, a life is lost. The initial number of lives is three.
2. Character behaviors: The ghosts never turn back halfway, nor do they turn back at crossroads. Pac-Man moves at a constant speed. Player can use keyboard to control the Pac-Man's moving direction, but not speed. If no key is pressed, the Pac-Man continues going forward until it meets a wall and then stops. If two ghosts meet, they keep on moving forward and cross each other.
3. Scores: Each pac-dot is worth 10 pts.
4. Completing a level: When all pac-dots are eaten, the level is completed and a congratulation screen is displayed within the playfield. After the player hits any key, the game is reset to the title screen.
5. Game over: When all lives have been lost, the game ends and a game over screen is shown in the playfield. After the player hits any key, the game is reset to the title screen.
6. Levels: At least three levels (from easy to difficult).

Advanced Functionality Suggestions

- More complicated game logic: power pellets, ghost box, teleport tunnel...
- More complicated designs of levels.
- Machine-controlled mode.
- 3D game.
- Use of sound in the game.
- Prettier UI.
- ...

1.4. Gobang

Gobang, or *Gomoku*, *Five in a row* is a classic strategy board game on a Go board. Players alternate turns placing a stone of their color on an empty intersection. The side that first forms five consecutive pieces of the same color on the horizontal, vertical, and diagonal directions of the board is the winner. For more details, please refer to <https://en.wikipedia.org/wiki/Gomoku>.



Basic Requirements

- Components
 1. Either Command-Line Interface (CLI) or Graphical User Interface (GUI)
 2. A 19×19 Goboard
 3. Two Players
 4. Different Stones for different Players
- Player Type
 1. Game of two human players
 - a. Players are connected via a server (either local or remote), and each player has a separate game window.
 2. Game of one human player and one random player (a machine agent that places a stone randomly)

Design proper UI to select the player type.

- User Management
 1. Users are required to sign up / log in before the game. Only user name and password is needed.
 2. Upon login, the game shows a main page, including a “start new game” panel and “view game record” panel. Users can view the game record with the following attributes:
 - Start time
 - Elapsed time
 - Player Names
 - Winner
 - Final Goboard with stones
 3. Your application should have an admin user that can view all user information and delete user if needed.
- General Game Logics
 1. Player move: Player can place a stone to a non-occupied position. After a player places a stone, the stone is rendered correctly. Then, another player takes the turn to place another

stone.

2. Gameover: When a player gets “five in a row”, the game ends, and the player wins the game. The game returns to the main page.
3. (Only applicable to a game of two human players) Allow retracting a false move: Player can retract a false move after seeking agreement from another human player. Show a “Retract move” button.
4. During the game, show the following information with clear UI design:
 - a. Start time
 - b. Elapsed time
 - c. All player names
 - d. Current player and its stone type
 - e. Current Goboard with stones

Advanced Functionality Suggestions

In the advanced requirement part, you are welcome to further enrich the game with the following details:

- Support (some of) the functionalities in more complicated game control:
 - a. Score system
 - b. Chit-chat during game
 - c. Add friends and further invite friends to a game
 - d. Early Termination of the Game (e.g., “open 4” 活四, double "open 3s" 雙活三)
- Gobang is not a balanced game for two players. Hence, in modern competitions, there are forbidden moves that Black cannot play during the game. Hence, you can detect and disallow (some of) the forbidden moves. Here are some useful resources:
 - (In Chinese) 中國五子棋競賽規則
<http://ku10.com/resources/PDF/%E4%B8%AD%E5%9B%BD%E4%BA%94%E5%AD%90%E6%A3%8B%E7%AB%9E%E8%B5%9B%E8%A7%84%E5%88%99-2013.pdf>
 - “Variants” Part in Wikipedia: <https://en.wikipedia.org/wiki/Gomoku>
- Implement Game AI with (some of) the following techniques:
 - Rule-based AI
 - Advanced Techniques, e.g. alpha-beta pruning. You may also propose other methods.
- Sound effects of the game
- ...retty UI Design
- ...

Appendix 2: Guidance for Documentation

A total of four documents (i.e., a High-Level Design Document, a DFD Specification Document, a UML Specification and UI Design Document, and a Testing Document) will be submitted by each project team. The reports should be submitted by the whole team (one report per team) and all team members will have the same score for a report.

Each document should contain three parts: a cover page, a table of contents, and detailed contents. The Cover page must contain the following information

- Name of Document
- Project Title (You can create your own project name under the assigned topic)
- Document Version Number
- Printing Date
- Group ID
- Group member names and SIDs
- Department & University

2.1. High-Level Design Document Outline (Main body *no more than 5 pages*)

The high-level design document is mainly focused on the purpose of the application you are designing and its high-level descriptions. You should describe the objectives, features, and architectural designs of your project. The recommended outline is listed as follows.

1 INTRODUCTION

1.1 Project Overview

1.2 System Features

2 SYSTEM ARCHITECTURE

2.1 Technologies

Database, UI, Programming Language, ...

2.2 Architecture Diagram

What components do you have in your applications? How they interact with each other?

2.3 System Components

Describe your architecture diagram.

1.2 DFD Specification Document (Main body *no more than 10 pages*)

In this document, you should describe the system specification of your project with data flow diagrams (DFDs). You should use DFDs to introduce the operational specification of your applications. Specifically, you can first describe the high-level context diagram of the entire system. Then you can refine the high-level DFD by describing more features of your selected application. The following is the recommended outline:

- 1 High-Level Context Diagram
- 2 Feature Diagrams
 - 2.n Feature-n
 - 2.n.1 Description (A brief description of the feature.)
 - 2.n.2 DFD

1.3 UML Specification and UI Design Document (Main body *no more than 20 pages*)

In this document, you should specify your application components with UML diagrams and design your graphic user interfaces with a UI design document.

- Detailed description of components by UMLs
In this section, you should use UMLs to provide detailed descriptions of at least the key components. Contents that you should concern in this section: UML diagrams (including use-case diagrams, class diagrams, and sequence diagrams) of the major class, functionality of the component, list of major functions, etc.
- User Interface Design
In this section, you should present the UI of your project. This section could be like a user manual of your project. You should teach and guild the users by walking through your UI operations. The use of screenshots is needed to indicate your UI overview and the result after certain user actions.

The recommended outline is described as follows:

- 1 UML DESIGN
 - 1.n Component-n
 - 1.n.1 Structural Diagram
 - 1.n.2 UMLs
 - 1.n.3 Functionality
 - 1.n.4 Procedures and Functions
- 2 UI DESIGN
 - 2.n View-n
 - 2.n.1 Description of the view
 - 2.n.2 Screen Images
 - 2.n.3 Objects and Actions

1.4 Testing Document (Main body *no more than 15 pages*)

In this document, you should present the test plan, test case design, and test result of your project. Your document should contain a *test plan* section and a *test cases* section. In the *test plan* subsection, you should contain your test approaches (*e.g.*, black or/and white box testing), features to be tested (*e.g.*, the rules of Gobang), testing tools (*e.g.*, gtest for C++), if any, and the testing environment (the hardware and software requirements). In the *test cases* section, you should concern the objective of each test case, input, expected outputs, Pass/Fail criteria, and so on. The following is the recommended outline:

1 TEST PLAN

2 TEST CASES

2.n Case-n

2.n.1 Purpose

2.n.2 Inputs

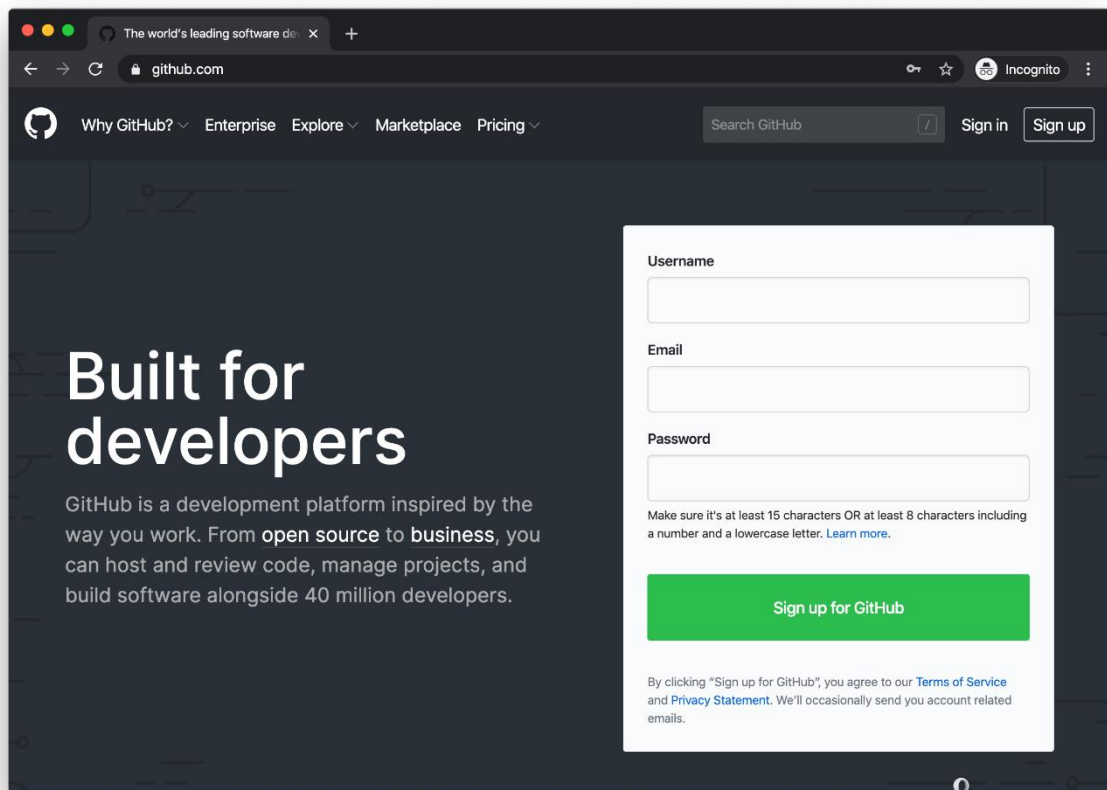
2.n.3 Expected Outputs & Pass/Fail Criteria

Appendix 3: Guidance for Code Submission

You MUST submit your project via **GitHub.com**, and faithfully record your coding activities.

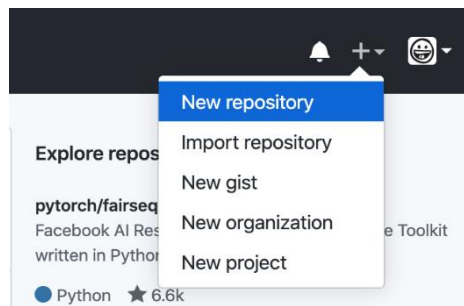
1. Create a GitHub account (if you don't have one)

GitHub is the leading code management platform worldwide. If you do not have an account, you can navigate to <https://github.com> and Sign up.



2. Create a new repository

To facilitate submission, you are required to set up a git repository on GitHub.com. Click the “+” on the upper right corner of GitHub, and click “New repository”




Then enter the name of your repository and choose the visibility. You can set up either public or private repository. For public repository, you need to submit **HTTPS URL** of your repository (e.g., <https://github.com/ytty/my-awesome-project.git>). For private repository, you need to follow the guide below to add ssh key and submit **SSH URL** of your repository (e.g.,


[git@github.com:ytty/my-awesome-project.git](https://github.com/yttty/my-awesome-project.git)).

Owner

Repository name *


 yttty ▾

 /


awesome-code 

Great repository names are short and memorable. Need inspiration? How about **supreme-potato**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾


 |

Add a license: **None** ▾ 


Create repository

Owner

Repository name *


 yttty ▾

 /


awesome-code 

Great repository names are short and memorable. Need inspiration? How about **supreme-potato**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.


Add .gitignore: **None** ▾

 |

Add a license: **None** ▾ 


Create repository


Owner **Repository name ***

 ytty / awesome-code ✓

Great repository names are short and memorable. Need inspiration? How about **supreme-potato?**

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.


Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** ⓘ


Create repository


Owner **Repository name ***

 ytty / awesome-code ✓

Great repository names are short and memorable. Need inspiration? How about **supreme-potato?**

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

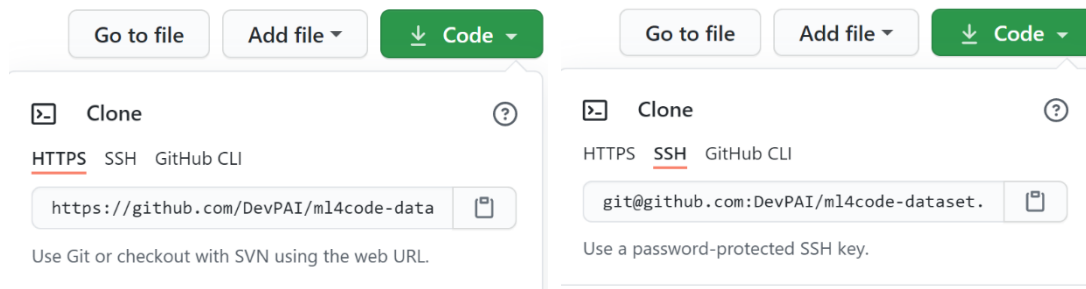
Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

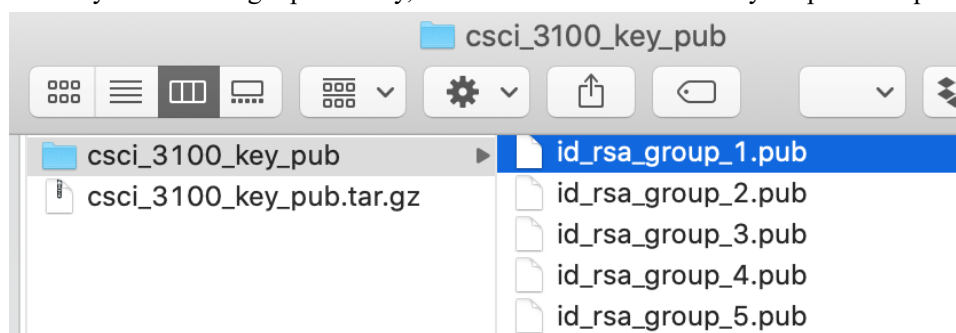
You are required to submit the **HTTPS URL** or **SSH URL** of your repository to the Google form <https://forms.gle/DaE52gpD8q5bC5Zg7> **before the Phase 2 due (i.e., 25 Feb.)**. As specified in the project specification, we will pull the latest code on the due date of **Phase 2 and Phase 5**.



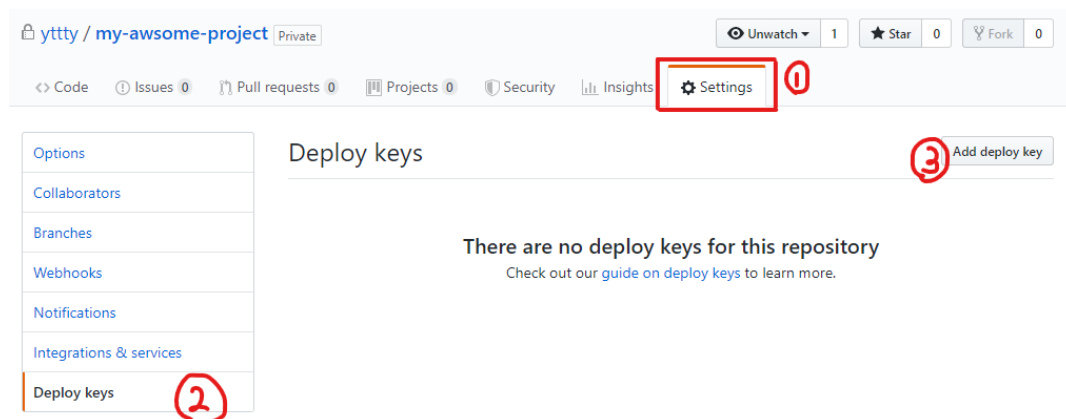
3. Add deploy key to the private repository

If you choose to create a private repository, you need to add deploy key to your project so that TAs can access your code. Please be reminded the member capacity of a private repository of a free account is limited to 3. You should apply for [Student Account](#) to increase the member capacity to enable everyone to contribute code.

First, download the key from the project page of course website http://proj.cse.cuhk.edu.hk/csci3100/assets/project/keys_2023_pub.zip; extract this zip file; open the public key of your group with a text editor and copy the corresponding public key **of your group**. Please ensure you add the right public key, otherwise TAs cannot access your private repository.



Then, go to your GitHub repository, navigate to settings->Deploy keys->Add deploy key



Paste the public key you just copied, then click Add key.

Deploy keys / Add new

Title

Key

```
ssh-rsa
AAAAAB3NzaC1yc2EAAAADAQABAAQACuedG4zGAdKH27ebrSTc1j1AWK6+fbXilBWAT98ZTsQSSL65uvEvGoSZx8G
VgHssKoaK4RBnNJ8yhr9DKJdlmOiAzt9kw7ONrz9ctpcx/LrSk5RAnEB2optdRn4Elw1laejl/cCUvX63/WUhv4uXJ/ThXI4
NnDuuG3+br99p2MXh+K2Hn9RRGdZau/7EleuFc2uVRtPNSgxmPLmVqz2AlsipU5wAw8tqiTxlrZmFMUS9iDN4J9/tnr
AxZdvT1Kasp+TyPi+NIQhcAWIPkpKraSdSln3T6WrGHclY14PEol2Vb3ll/DgabNslThTwuq0+r7AePW8PlyzNrlUOOLDy
LH ssh key for group 1|
```

☐ Allow write access

Can this key be used to push to this repository? Deploy keys always have pull access.

Add key

4. Read More

- If you are new to git, start with this [simple guide](#).
- The official git documentation: <https://git-scm.com/documentation>.
- Git and GitHub Tutorial <https://www.youtube.com/watch?v=xuB1Id2Wxak>

Appendix 4: Application Requirement Checklists

1. Twitter

Functionality	Requirements	Points (70 Total)
Application Architecture		
Client-Server Architecture	A client-server architecture that can support multiple clients.	4'
User Interface		
Graphic UI Design	A graphical user interface.	5'
Database		
Database Integration	Integrate a global database like MySQL.	4'
User Management		
User signup	Create a new user profile.	4'
User Login & Logout	Let user login and logout your application. Users should be able to conduct more operations after login.	4'
Admin User		
Admin user interface	The admin user should have different interface from normal users.	5'
List all users	The admin user should be able to access the information of all users.	4'
Delete users	The admin user should be able to delete users.	4'
Application Requirements		
Search for users	A user can search for other users with usernames or unique userIDs.	5'
Follow other users	A user can follow another user by clicking a "follow" bottom.	5'
Like/Dislike tweets	Each tweet is associated with a like counter. For each tweet, a user can increase/decrease the like counter by 1 by clicking the like/dislike bottom. Each user can either like or dislike a tweet.	6'
Comment tweets	A user can leave comments below tweets. The comment is available to everyone when reading this tweet.	5'
Post tweets	A user can post a tweet with images.	5'
Retweet tweets	A user can retweet a tweet with the original author's information	5'
Show following user tweets	For a user, the application should show tweets of users he/she follows on the main page.	5'

2. Course Selection System

Functionality	Requirements	Points (70 Total)
Application Architecture		
Client-Server Architecture	A client-server architecture that can support multiple clients.	4'
User Interface		
Graphic UI Design	A graphical user interface.	5'
Two main pages	Course browsing page and profile page.	5'
Database		
Database Integration	Integrate a global database like MySQL.	4'
User Management		
User signup	Create a new user profile.	4'
User Login & Logout	Let users login and logout your application. Users should be able to conduct more operations after login.	4'
Admin User		
Admin user interface	The admin user should have a different interface from normal users.	5'
List all courses/users	The admin user should be able to access the information of all courses/users.	4'
Delete courses/users	The admin user should be able to delete courses/users.	4'
Application Requirements		
Search for courses	1) A course entry should contain course ID, course name, time, place, department, instructor, and capacity. 2) A user can search for a course based on course ID and course name. 3) A user can search for courses by conditions.	1) 3' 2) 4' 3) 4'
Select courses	1) A user can select a course by clicking the "Select" button. 2) If the course capacity is full, the system will prompt the course cannot be selected. 3) The selected course should be displayed on the profile page.	1) 3' 2) 4' 3) 3'
Show selected courses	All the selected courses should be shown on the profile page.	4'
Drop courses	1) A user can drop a selected course by clicking the "Drop" button. 2) The dropped course should disappear from the profile page.	1) 3' 2) 3'

3. Pac-Man

Functionality	Requirements	Points (70 Total)
Database		
Database Integration	Integrate a global database like MySQL.	2'
User Management		
User signup	Create a new user profile.	4'
User Login & Logout	Let user login and logout your application. Users should be able to conduct more operations after login.	4'
Application Requirements		
Main window	A main window appears after the program is launched. The window contains menu items and a playfield.	2'
Menu items	"Reset" menu item restarts the game. "Clear High Score" erases the high score record. "Exit" causes the program to quit. They are functional at any time.	6' (3 * 2', each menu item takes up 2 points)
Title screen	When the main window is shown, a title screen is automatically displayed within the playfield. The title screen should contain the name (preferably the logo) of the game, and an instruction telling player how to start and operate the game.	3'
Basic rendering	A maze whose layout contains at least four horizontal and four vertical passages is rendered inside the playfield when the game is started with a black background. Three characters, Pac-Man and two ghosts in arbitrary colors, are drawn inside the maze at arbitrary initial positions. Other required maze elements include pac-dots and walls.	6'
Text	The current player's score and highest scores are displayed at the top of the playfield. In the lower-left corner a number of Pac-Mans is drawn, displaying the number of lives left.	3'
Control	Pac-Man can be controlled by keyboard. It can move in the direction as the player instructs, but the speed is invariant. Pac-Man keeps on moving without changing its direction if there is no operation and stops if it meets a wall.	6'
Basic movement	All characters move along the lanes and cannot go into unreachable areas which are	9'

	enclosed by the walls. When a Pac-Man passes a pac-dot (including power pellet), that dot disappears.	
Animations	When the characters are moving, they are rendered as animations (changing pictures like .gif) instead of still images. If the Pac-Man stops moving, its image becomes still. If Pac-Man bumps into a ghost, an animation is played.	4'
Behavior of enemies	The ghosts keep moving in the same direction until they reach a cross where they can make a random choice.	3'
Game status	Game score, high score and lives left are updated and rendered correctly. The high score is kept in a persistent way so that it won't be lost as the program exits. Game status is initialized whenever a new game is started.	3'
Gameover	When all pac-dots are eaten, a congratulation screen is displayed. If all lives are lost, a gameover screen is displayed. In either case, the player can dismiss the screen by hitting a key and the game goes back to the title screen.	3'
Levels	At least three levels (from easy to difficult).	12' (3 * 4', each level takes up 4 points)

4. Gobang

Functionality	Requirements	Points (70 Total)
Database Integration		
Database Integration	Integrate a global database, e.g. MySQL, MongoDB	4'
Basic Game UI Design		
Basic Game UI Design	During the game, display a 19x19 Goboard, current player and its stone type, all player names, start time, elapsed time, clearly and correctly.	10' (2' * 5)
Players		
Two human players	Support two-human player game. Players are connected via a server (either local or remote), and each player has a separate game window.	10'
Random Player	Support the game between a human player and a random player	5'
User Management		
User signup	Create a new user profile. Only user name and password is needed.	4'
User Login & Logout	Let users login and logout in your game. Users should be able to conduct more operations after login. Upon login, show the main page with two panels: "start new game", "view the game records".	6' (4' for log in/out, 1' for each panel UI design)
View Game Records	Users can view the game record with the following attributes: start time, elapsed time, player names, winner, and the final Goboard with stones.	5' (1'*5)
Admin User		
List User	The admin user should be able to access the information of all users.	4'
Delete User	The admin user should be able to delete the users when needed	4'
Game Logics		
Player move	After a player places a stone, the stone is rendered correctly. Meanwhile, different players have different stones.	6'
Gameover	When a player forms <i>five consecutive pieces</i> of the same color on the horizontal, vertical, and diagonal directions, the game ends. The game returns to the main page.	6'

Retract a false move	In a game of two human players, one player can retract a false move after seeking agreement from another human player.	6'
----------------------	--	----