CSCI 3100 Software Engineering
Project Requirement Specification

Last Update: 27th Feb 2024

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. We will arrange a demo day at the term end, where you will present your product on the day. Your complete code should be submitted after the demo.

2. Project Grouping

Each project group is composed of **four or five** 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 (**19 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.

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. Please refer to **Appendix 3** for more details.

- 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.

- 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.

- 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 content-oriented application, a *online shopping system*, and two games -- *snack.io* and *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 2** containing the grading details of basic requirements.

The advanced requirements account for the remaining 30% of the project. In Appendix 1, you'll find helpful advanced suggestions for each application. Feel free to try out these features in your projects. You're also encouraged to add as many cool and unique features as you like. For example, you could make your application stand out by adding more specific features that users will love. 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 attribute any reused code clearly. This requires them to clearly attribute any code not authored by themselves at the beginning of each submitted file. 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. Feedback 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 3 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 24 February (Saturday)**. You 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 4**.

(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 23 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 **11 April**. Each group have 15 minutes to present the application to the course instructors. Detailed project demo arrangements will be announced later. Please pay attention to the announcement on the course website and Piazza.

(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 4 May (Saturday)**.

5. Grading Criteria:

The followings are the project schedule of different phases:

Phase Deliverables	Weightings	Durations	Due Date
0. Team Formation	--	--	19 Jan. (23:59:59)
1. High-Level Design Document	5%	2 weeks	3 Feb. (23:59:59)
2. DFD Specification Document and GitHub Repository Creation	10% (8%: DFD Specification Document, 2%: GitHub Repository Creation)	3 weeks	24 Feb. (23:59:59)

3. UML Specification and UI Design Document	15%	4 weeks	23 Mar. (23:59:59)
4. Project Demo	60%	2.5 weeks	Demo Day: 11 Apr.
5. Testing Document and Final Commented Code	10% (8%: Testing Document, 2%: Final Commented Code)	3 weeks	4 May (23:59:59)
Total	100%	15 weeks	

5.1. High-Level Design Document

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.

5.2. 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.

5.3. 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.

5.4. 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 2**. 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.

5.5. 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 3** for more details. 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. Free-rider cases will be investigated on the project demo day, where group members can complaint about free-rider(s), please raise the case *right after* your project demo. The course instructors 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, 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 4** on code submission. Further inf**orma**tion will be provided in the related emails or information on the website.

6.3. Submission Policies

Like homework, you shall submit your project document **with signed VeriGuide receipts**. The policy late submission (including VeriGuide receipt) is the same as homework. You can find the policy on the course [website](). You are allowed to use AI tools, but you have to explicitly cite or acknowledge the use of these tools.

Appendix 1: Application Requirements

1.1. Content-Oriented Application

You are required to implement your own content-oriented application like Twitter, RedBook, and Tik-Tok, which mainly display text, pictures and videos. Users can post and interact with different type of messages. The following are the basic requirements and advanced suggestions.

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:

  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 message

     Each message 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 message

     A user can leave a comment below a message. The comment is available to every when reading this message.

  5. Retweet a message

     A user can retweet a message with the original user's information.

  6. Post a message

A user can post a message (text is ok, image and video are preferred).

7. Show other users' tweets

For a user, the application should show all the messages 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. And consider adapt UI to your specific content type.

- Message Tag: You can mark messages with tag, which can be later used for search.

- 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.

- Content Recommendation: The application can be extended to recommend popular content of interest for a user.

- Private Chat: The application can allow users to send private messages to each other.

- Higher level Content type: Of the three types of content, we prefer video the most, followed by images, and finally text.

- Any other features that may make your design impressive.

1.2. Online Shopping Mall System

You are required to implement an online shopping application for a retail company. Users can search for products, add them to their cart, and make purchases.
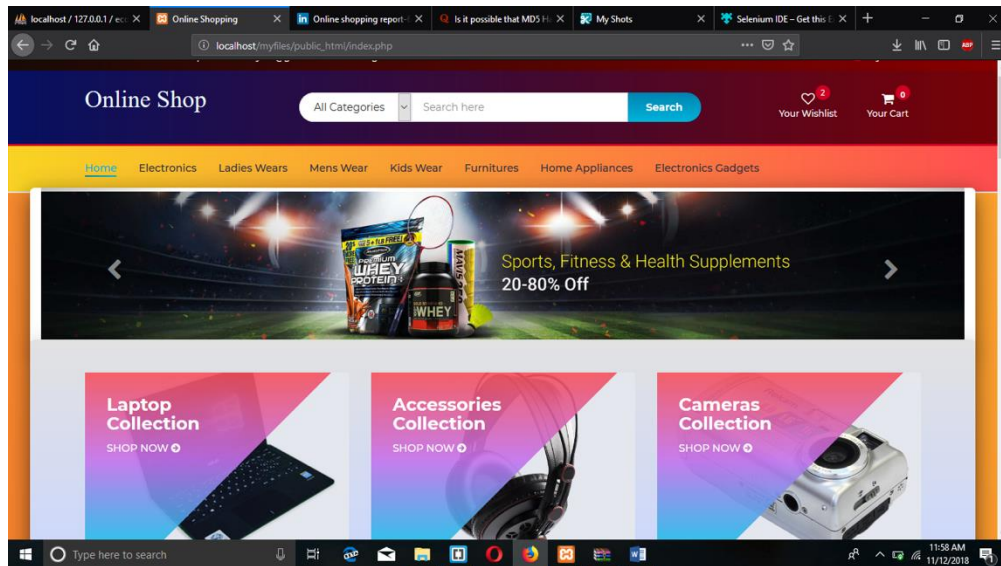
Basic Requirements

▪ Client-server architecture

The application should follow a client-server architecture. The server should hold all the information of products 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

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.

▪ You are required to implement two main pages for users, that are, product browsing page and cart page. In the product browsing page, users can search and add products to their cart. In the cart page, users can view all the products they have added and can remove selected products.

▪ 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 product/user information and add or delete products/users if needed.

▪ User Operations

Your application should allow users to conduct the following operations

    1. Search for products.

        ▪ A product entry should contain product ID, product name, price, category,

description, and stock.

- A user can search for a product based on product ID and product name.
- A user can search by conditions such as category and price range. The system will list all eligible products.

2. Add products to cart.
   - A user can add a product to the cart by clicking the "Add to Cart" button on the product entry.
   - If the product stock is zero, the system will prompt that the product cannot be added to the cart.
   - When a user successfully adds a product to the cart, the product will be displayed on his/her cart page. This information needs to be updated to the database immediately.

3. Show selected products.
   - A user can go to his/her cart page to view all the selected products.

4. Remove products from cart.
   - In the cart page, a user can remove a selected product by clicking the "Remove" button on the product entry.
   - When a user successfully removes a product, the product will disappear from his/her cart 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.
- Simulated Payment System: Implement a simulated payment system that allows users to go through the process of making a purchase, including entering payment information and completing the transaction
- Product review and rating: Users can leave reviews and ratings for the products they have purchased.
- Recommendation: Implement a recommendation system that suggests related or similar products to users based on (1) product rating or (2) their browsing and purchase history.
- Order tracking: Users can track the status of their orders.
- Any other features that may make your design impressive.

1.3. Snake.io

Snake.io is an online io game where you play as a snake fighting to survive on a battlefield of snakes. Eat colorful bits of food to grow bigger and take down other snakes to become an absolute unit to be reckoned with.

https://snake.io/
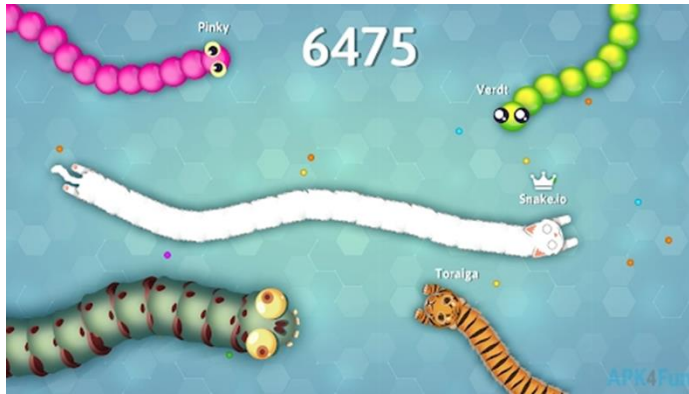
Basic Requirements

▪ **User Interface (Basic Game Components)**

1. Menu Items: Include "Play", "Leaderboard", "Customize", "Settings", and "Exit". These allow players to start the game, view high scores, customize their snake, adjust game settings, and exit the game.
2. Title Screen: Displays the game's title and a brief instruction or tagline. May also include quick access to play or customization options.
3. Characters: Players control a snake that grows as it consumes pellets. Different color and pattern options for customization. Another moving snake controlled by the computer.
4. Game Arena: The playfield is a bounded area where snakes move around to collect pellets and interact with other players' snakes.
5. Messages: Display the player's current length and rank during gameplay, along with other relevant in-game messages.

▪ **User Management**

1. Players can play as guests or create an account for additional features like customization and leaderboard tracking.

▪ **Database**

1. A global database to store player data, customization options, leaderboard information, and game settings.

▪ **Functional Requirements**

1. Basic Gameplay: players control their snake within the arena, aiming to grow as long as

possible by eating pellets and defeating other players.

2. The game is over when the player's snake collides with another snake or the boundary.

3. Character Behaviors: Snakes move continuously and can change direction based on player input. Snakes grow in length with each pellet consumed. Another snake controlled by the computer shares the same behavior.

4. Scores: Score is based on the length of the snake and the number of other snakes defeated.

5. Game Over: The game ends when the player's snake collides with another snake or boundary, with a display showing the player's final length and rank.

6. Levels: but continuous gameplay with increasing difficulty as the snake grows longer.

Advanced Functionality Suggestions

- Power-Ups: Special items that give temporary advantages, like speed boosts or invincibility.
- Event Modes: Special limited-time game modes with unique themes or rules.
- Multiplayer Features: Options for playing with friends or in specific groups.
- 3D game.
- Mouse mode vs. Keyboard mode.
- Complex arena.
- Any other features that may make your design impressive.

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 colour on an empty intersection. The side that first forms five consecutive pieces of the same colour 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.
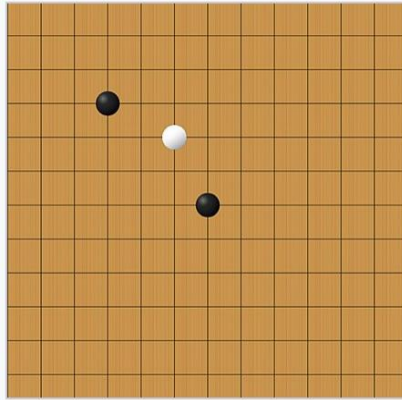
Basic Requirements

- 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)
- User Management
    1. Users are required to sign up / log in before the game. Only user name and password are needed. The score of the user is initialized.
    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:
        o Start time
        o Elapsed time
        o Player Names
        o Winner
        o Final Goboard with stones
- General Game Logics
    1. Users select player types (human vs human, or human vs machine).
    2. Assign users with stones.
    3. Player move: Player can place a stone to a non-occupied position within the goboard. After a player places a stone, the stone is rendered correctly. Then, another player takes the turn

to place another stone.

4. 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.

5. (Only appliable 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.

6. Score system: users win scores upon winning the game, lose scores upon losing the game.

7. Chat system: users are able to chat during the game.

8. During the game, show the following information with clear UI design:
   a. Start time
   b. Elapsed time
   c. Current player (name) and its stone type
   d. Player scores
   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. Add friends and further invite friends to a game
  b. Early Termination of the Game (e.g., "open 4" 活四, double "open 3s" 雙活三)
  c. Time control. For example, each player has a *main time limit*, say 20 minutes to make decisions on *all* their moves. Once a player uses up their main time, they enter into the *elegant time* period. This is often a series of fixed time intervals, like 30 seconds per move.

- Gobang is not a balanced game for two players. 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:
  o (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
  o "Variants" Part in Wikipedia: https://en.wikipedia.org/wiki/Gomoku

- Implement Game AI with (some of) the following techniques:
  o Rule-based AI
  o Advanced Techniques, e.g. alpha-beta pruning. You may also propose other methods.
  o Invoking API of Large Language Models (LLMs), such as GPT-3.5-turbo from OpenAI (https://platform.openai.com/docs/api-reference). Encode the game information as a prompt, and ask a LLM to return the location to place the stone.

- Implement admin users that can view user records and delete users.

- Sound effects of the game

- Good UI Design

- Any other features that may make your design impressive.

Appendix 2: Application Requirement Checklists

1. Content-Oriented Application

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 messages	Each message is associated with a like counter. For each message, a user can increase/decrease the like counter by 1 by clicking the like/dislike bottom. Each user can either like or dislike a message.	6'
Comment messages	A user can leave comments below messages. The comment is available to everyone when reading this message.	5'
Post messages	A user can post a message with images.	5'
Retweet messages	A user can retweet a message with the original author's information	5'
Show following user messages	For a user, the application should show messages of users he/she follows on the main	5'

	page.	

2. Online Shopping Mall 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	Product browsing product page and cart 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.	4'
List all products/users	The admin user should be able to access the information of all products/users.	4'
Delete products/users	The admin user should be able to delete products/users.	4'
Application Requirements		
Search for product	1) A product entry should contain product ID, product name, price, category, description, and stock. 2) A user can search for a product based on product ID and product name. 3) A user can search by conditions such as category and price range.	4) 3' 5) 4' 6) 4'
Select products	1) A user can select a course by clicking the "Select" button. 2) If the product stock is zero, the system will prompt that the product cannot be added to the cart. 3) The selected product should be displayed on the cart page. This information needs to be updated to the database immediately.	4) 3' 5) 3' 6) 4'

Show selected products	All the selected products should be shown on the cart page.	4'
Remove products	1) A user can remove a selected product by clicking the "Remove" button on the product entry. 2) The dropped product will disappear from the cart page. This information needs to be updated to the database immediately.	3) 3' 4) 4'

3. Snake.io

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	An arena, snakes and pallets.	6'
Text	The current player's score and highest scores are displayed at the top of the playfield. In the lower-left corner the number of snakes is drawn, displaying the number of lives left.	3'
Gameplay Mechanics	Control of the snake, movement mechanics, pellet consumption, enemy snake behaviors, and game status updates.	15'

Animations	When the characters are moving, they are rendered as animations (changing pictures like .gif) instead of still images. If the snake stops moving, its image becomes still.	4'
Behavior of enemies	The enemy snakes keep moving in the arena and will avoid the boundaries of the arena or the player. It will also consume pallets.	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 enemies die, 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'
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	4'

	and password are needed.	
User Login & Logout	Let users login and logout in your game. Users should be able to conduct more operations after login.	4'
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)
UI Design		
Upon Login	Upon login, show the main page with two panels: "start new game", "view the game records".	4 (2' * 2)
During the game	During the game, display a 19x19 Goboard, current player and its stone type, all player scores, start time, elapsed time. The information should be presented clearly and correctly.	10' (2' * 5)
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 *five consecutive pieces* 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'
Chat System	Players are able to send and receive chat messages from each other during the game.	6'

Appendix 3: 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 freely name your own project)
- Document Version Number
- Printing Date
- Group ID
- Group member names and SIDs
- Department & University

3.1. 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.

3.2 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

3.3 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

3.4 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* sub-section, 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 4: 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.
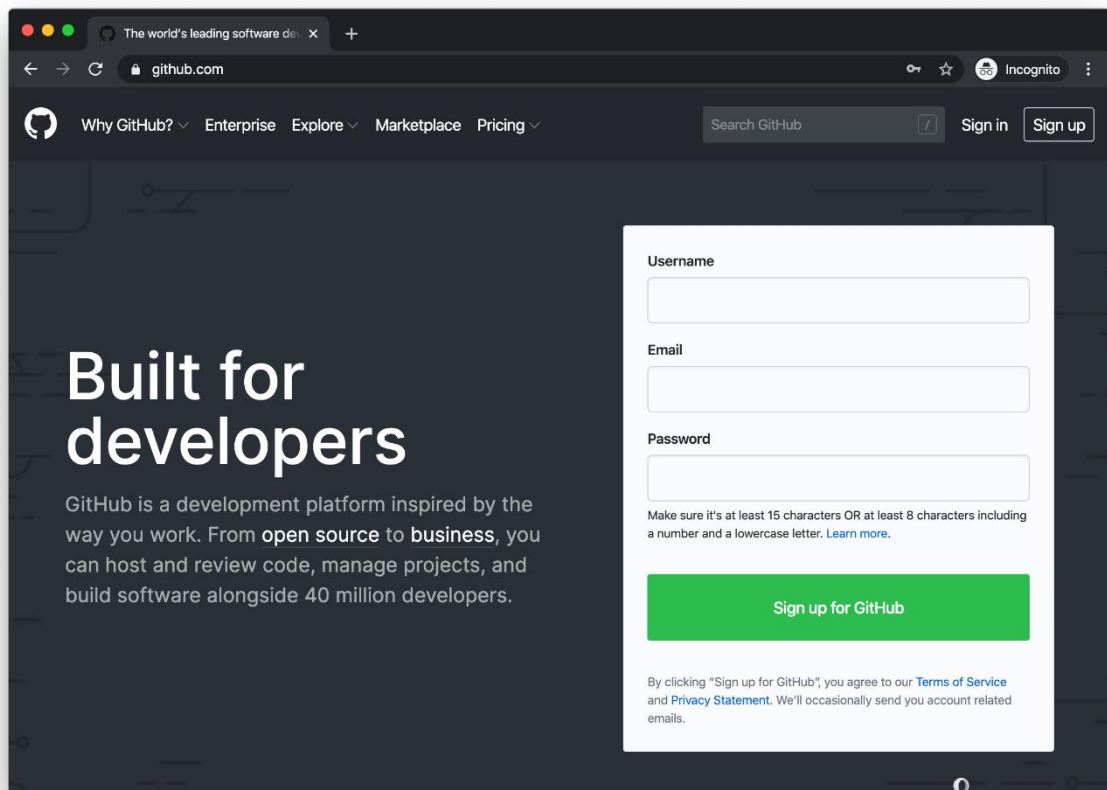
2. Create a new repository

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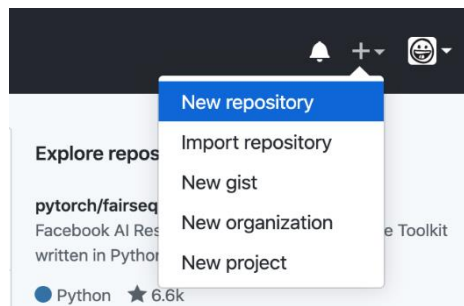
Then enter the name of your repository and choose the visibility. You can set up either public or private repository. **For public repository, you need to submit HTTPS URL of your repository (e.g., https://github.com/yttty/my-awsome-project.git). For private repository, you need to follow the guide below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

git@github.com:yttty/my-awsome-project.git).

You are required to submit the **HTTPS URL** or **SSH URL** of your repository to the Google form https://forms.gle/PPRAzpJkS63JGcDF8 **before the Phase 2 due (i.e., 24 Feb.)**. As specified in the project specification, we will pull the latest code on the due date of **Phase 2 and Phase 5**.
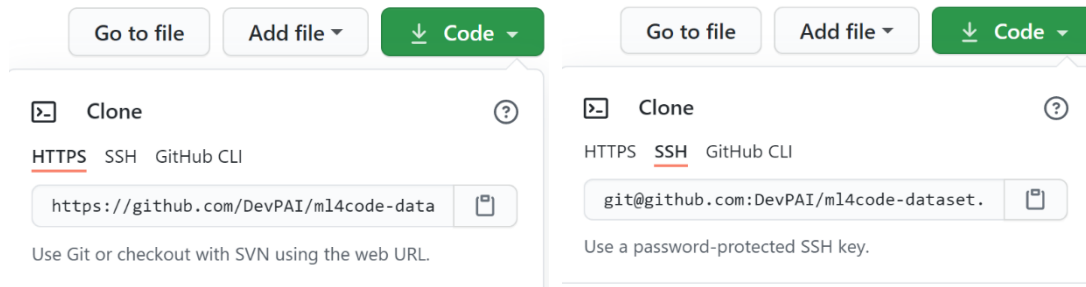
3. Add deploy key to the private repository

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 course website

http://proj.cse.cuhk.edu.hk/csci3100/assets/project/keys_2024_pub.zip; extract the 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.
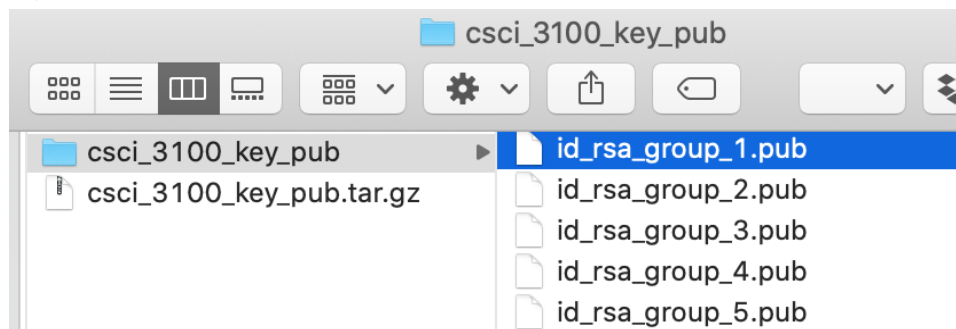
Then, go to your GitHub repository, navigate to settings->Deploy keys->Add deploy key
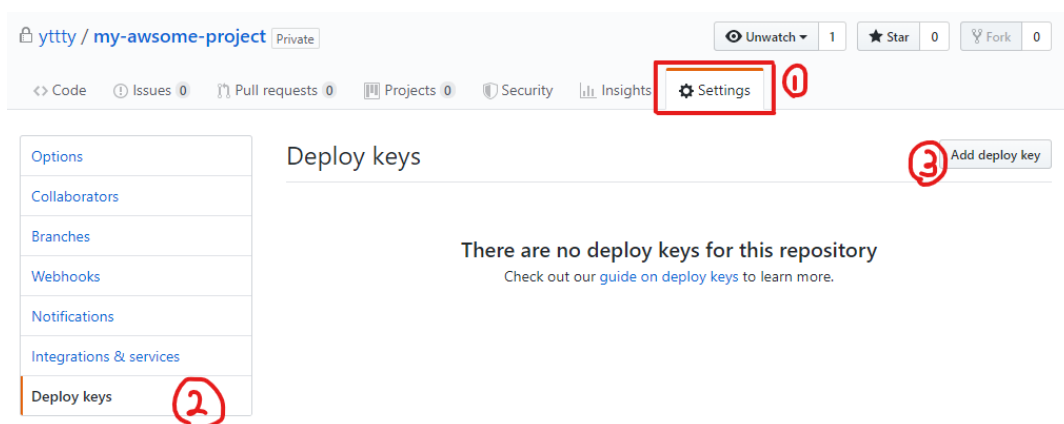
Paste the public key you just copied, then click Add key.

Deploy keys / Add new

Title

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCuedG4zGAdKH27ebrSTc1j1AWK6+fbXiIBWAT98ZTsQSSL65uvEvGoSZx8G
VgHssKoaK4RBnNJ8yhr9DKJdlmOiAzt9kw7ONrz9ctpcx/LrSk5RAnEB2optdRn4EIw1IaejI/cCUvX63/WUhv4uXJ/ThXI4
NnDuuG3+br99p2MXh+K2Hn9RRGdZau/7EIeuFc2uVRtPNSgxmPLmVqz2AIsjpU5wAw8tqiTxIrZmFMUS9iDN4J9/tnr
AxZdvT1Kasp+TyPi+NIQhcAWIPkpKraSdSIn3T6WrGHclY14PEol2Vb3lI/DgabNslThTwuq0+r7AePWBPIyzNrIUOOLDy
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. References

- 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