ISYS 1117/1118 Software Engineering Fundamentals

Assignment Draft   **2018 Semester 2**

Objective

This SEF assignment has two milestones designed to introduce students to various processes including the software engineering lifecycle, UML-design, refactoring and testing. The first milestone nurtures an agile development approach placing more emphasis on working software and test driven development (TDD). Students will become familiar with collaboration, version-control and testing tools that facilitate a continuous integration (CI) approach. The second milestone requires students to refactor the code and document the design using UML notation, thus making the application maintainable and extensible. In the second milestone students will be required to present UML diagrams in addition to demonstrating the final working software. In addition to these milestones, marks will also be awarded for class progress as specified in the Progress Marks section below. Students are expected to work individually (50 hours) and as a team (30 hours) for this assignment in addition to the time spent during contact hours. **Please note this assignment carries 40% of the overall marks for SEF**.

Progress Marks

Wk	Suggested Activities (these may vary between teams but students are expected to show progress)	Progress	Milestone
3	Select the project and form the team – consider the strengths, interests and experiences of team members	1 mark	
4	Identify User stories for Product/ Release Backlog	1 mark	
5	Identify main domain classes and add responsibilities such that objects can interact to get the first release done. Distribute classes/responsibilities to members.	1 mark	
6	Write Test Cases and start writing code to make them pass (individual)	1 mark	
7	**Initial Milestone**: Present partially completed product and get feedback from tutor and class. Demonstrate working Test cases (individual) and continue integrating code and testing.	1 mark	5 marks (see table)
8	Plan second release incorporating feedback. Add classes, extend class responsibilities and distribute work.	1 mark	
9	Develop detailed Group UML Diagrams (class, use-case).	1 mark	
10	Develop Individual UML Diagrams (sequence, state, object) Demonstrate individual Test cases	1 mark	
11	Develop Activity Diagrams. Combine code and run integrations tests. Resolve conflicts. Refactor design and code before final submission.	1 mark	
12	**Second Milestone**: Present final product to tutor and class.	1 mark	5 marks (see table)
13	**Face to Face Marking of design and code by Class Tutor**		20 marks (see table)

Milestone 1 (Total 5 marks)

This first demo/presentation will award marks for both group and individual work. Each individual member is expected to undertake at least two major tasks or responsibilities. Some of these responsibilities can be shared. For example, in the chess-like game "show all valid move destinations for a piece" can be a user-story/responsibility that must be shared by the Piece and the Board classes. The Piece derived classes such as Rook and Knight may know valid moves but only the Board classes knows where the vacant and opponent squares are located. In such a situation, those members writing the Rook and Knight classes may override the method getAllValidMoves():Squares[], while the person developing the Board class may write a method to check which of these squares are free to move using IsValidDestination(square). These two methods from different classes may thus be combined to determine all valid destinations. Students can defer non-model (GUI) aspects until the "core-logic" is fully tested (by using a simple textual interface).

Activity	Marks
Group Part	
Presenting the Initial Release Backlog (design documents: Class and Use Case diagrams) and demonstrating a partially completed product	2 marks (4 mins)
Individual Part	
1. Individual member is expected explain his or her role/obligation towards carrying out the main user-stories/responsibilities, and any challenges faced. 2. Individual member is expected to demonstrate how these obligations are discharged, by running some test cases. If code implementing these tests is not complete, present the test cases designed. If little or no progress has been made briefly explain the reasons.	3 marks (4 x 1.5 mins)

Milestone 2 (Total 5 marks)

The second demo/presentation will award marks for both group and individual work though more marks is awarded for group work.

Activity	Marks
Group Part	
Presenting the Final Release Backlog and demonstrating the final product	2 marks (3 mins)
Use a class diagram to present the overall design; use two sequence-diagrams to present the core functionality. Explain the design rationale highlighting usability, maintainability, extensibility etc.	2 marks (3 mins)
Individual Part	
Individual members explaining their role/ contribution and any challenges	1 marks (4 x 1 min)

Face to Face Tutor Marking (Total 20 marks)

This milestone will also award marks for both group and individual work though the emphasis is mostly on individual contribution to design and coding. The group effort will be measured through overall class and use-case diagrams, which should explain the overall requirements and design decisions. Each individual member is expected to demonstrate their individual contribution by detailing their part of the use-case (a use case textual description) and class-diagrams and by coming up with two other UML diagrams (a sequence diagram and **one** of state/activity/object diagrams). **Tutors will check your diagrams are aligned with your code**.

You must include a readme file identifying each group members percentage contribution. Group members that contribute either more or less will have their group mark adjusted accordingly.

Activity	Marks
Group Part	**Total 6 marks**
Overall Class Diagram (group) The class diagram must show all the methods, attributes, associations and multiplicities. Name the associations wherever possible	3 marks
Overall Use Case Diagram (group) This diagram should show all the relationships including extends/includes.	3 marks
Individual Part	**Total 14 marks**
Show detailed methods/attributes/associations relating to individual contribution in the class diagram	2 marks
Show detailed explanation relating to individual contribution in use case diagrams – present one use case textual description for a significant use case.	2 marks
Sequence Diagram (individual)	2 marks
One of state/activity/object diagram	2 marks
Code explanation relating to individual contribution. The code is expected to reflect the design. Code quality and explanation (code must be refactored)	6 marks

Submission Details: 12th Oct 2018 4.30 pm. No late submission allowed.

All source code and diagrams must be printed and bound together and be submitted to the School of engineering office (10.09.03) by 4.30 pm on the last day of the semester (12th Oct 2018).

You must also upload a zip file called FinalSubmission.zip to your groups Trello board.

Face to face marking sessions will commence on 15th Oct.

Bonus Features should not to be attempted until the completion of Standard Features

(Up to 5 marks)

Notes:

Bonus marks are applicable only to those individuals contributing to these features – who may be asked to explain the design rationale. Total mark for Assignment cannot exceed 40/40 (including progress and bonus marks)

FAQ

1. Can I use other OO languages (e.g. Groovy and Ruby) and technologies (e.g. Angular –JS and Node-JS) for implementation? You must seek permission with your tutor. While we are happy to encourage you to learn new technologies, there may be no additional support from the teaching team. Moreover, the UML diagrams and code required for the second milestone assumes an OO design/implementation is followed.

2. Do I need to write the test code in JUnit? Yes, unless you are using another OO language such as Groovy, Ruby or Android which have their own API. JUnit is a standard API that comes as part of Eclipse. JUnit Information available on: http://junit.org/faq.html. JUnit will also be covered in the lecture/tutes.

3. Can I use one of the free Cloud based platforms to host my system? We are happy for you to explore this option but we cannot provide any additional support.

4. Will there be a client for my software? Yes, your tutor will play that role, who will also assess both demos/presentations and final design/code. In addition there will be a discussion board for assignments. One of the team members can also take the role of product owner.

5. What additional help can I get? We will arrange additional student consultation times if necessary. The lecturer will announce specific times on the blackboard.

6. Do I need to show exceptions in class diagram? No they will clutter the diagrams.

7. What additional Java features (not taught in introductory programming courses) will be needed or useful? You may want to use Swing, JCF, Abstract classes, interfaces, exceptions etc., which can be found in any Introductory Java Programming books (such as that by Daniel Liang). Those who are developing distributed or web-based applications may consider using sockets, threads, servlets and Java server Faces/ Java server pages which can also be found in most introductory programming books (such as the book by Daniel Liang).

8. Why are we encouraged to use a MVC-based-architecture? It allows you to focus on the model initially (data structures and algorithms) without having to worry about the view aspects. Loose coupling in MVC also makes the product more maintainable.

9. What else can I do to make my product more extensible and maintainable? Consider using common design principles and patterns.

10. Why does this course require programming? Students find it difficult to know whether their UML design is good or even correct, without implementing the design. Moreover, implementing the design and measuring the tangible outcomes promotes an iterative approach resulting in improved designs. Implementing your design and testing whether the product meets the original requirements allows you to go through the whole software engineering lifecycle, thus preparing you better for your final year projects and your working career in the software industry. Note, student-mentoring support is provided for SEF programming related issues from week 3.

11. Why insist on teamwork? Teamwork and communication are the two main attributes employers are looking for in IT/SE graduates – and this project provides a great opportunity. Please be honest and open about your strengths and weaknesses, and seek help from other team members, teaching staff and mentors as early as possible.

Projects: Summary, Rationale and Common Aspects

Students with differing backgrounds and programming experience currently take SEF. SEF is a core course for different levels (UG, PG) and programs (CS, SE, IT). To cater for such diversity, students are allowed to choose one of the projects from the given list, which are designed to facilitate an agile or test driven approach for the first milestone, and a more rigorous design and implementation for the second milestone using UML notation and design/code refactoring. We have tried hard to come up with interesting and varied projects to match your interests. Your team should consider the strengths, background and interests of your members when selecting the project. You may also want to do some reading and testing before finalizing your assignment selection in week 3 (similar to the inception phase in RUP or the feasibility studies in the waterfall process). The first milestone is designed to enable students to develop some working-software or a prototype as soon as possible and get some feedback during the class presentation in week 7. The second milestone will focus more on the design aspects including extensibility and maintainability of the software.

In all of these assignments you are required to show weekly progress during your labs and make use of tools used in the industry such as Trello (Collaboration), JUnit (Testing), GitHub (Version Control) and Lucid Chart (UML Design). Guided lab exercises will help you familiarize with these tools. All students are expected to have some Java Programming experience through previous or current course offerings. The book Introduction to Java by Daniel Liang will cover all the Java knowledge required for this course. In the initial part, students with limited programming skills may encounter some steep learning curve; but our experience shows project based experience in SEF help improve student grades in other programming courses.

All assignments include some suggestions for initial test cases in addition to core requirements; please feel free to replace these test cases with others and come up with additional requirements that will enhance the functionality of the system. In the initial weeks each team is expected to spend some time coming up with specific classes and methods, which can collaborate to meet all the requirements. The features for bonus marks are to be attempted only after completing all the required tasks.

Assignment	Description
Course Management System	Presents a familiar domain dealing with student enrolment and courses. GUI is optional.
Budding Share Market Investor	Ideal for those who have some background in finance and interest in distributed computing.
Simple Strategic Game	Pacman like game combined with strategic collaborative monsters should make for an interesting and enjoyable game. Requires some knowledge of GUI and events. Some start up code may be provided/discussed in the lectures.
Supermarket Support System	Presents students with a familiar domain model involving domain classes Product, Sale, Customer etc. GUI is optional.
Traffic Light Simulator	This project will be an ideal one for those interested in flexible design patterns and graphical simulation. Requires some knowledge of GUI, patterns and use of events.
Chess like Game	This strategic game based project is much simpler than the traditional chess but allows students to develop their OO design and programming skills. The second milestone requires the use of Swing/FX classes.
Vacation House Exchange Club	This project is suitable for those interested in using a web-based framework or a client-server architecture (RMI). Common frameworks include Groovy on Grails, Play, Rails, Stripes, JEE or Spring framework; however there will be no technical help in these frameworks that may involve some steep learning curves. You may also consider using multiple technologies. Suitable for independent learners or those with some prior working experience.
Monster Game	This fun multiplayer project is suitable for those interested in working with distributed applications. It will require the use of sockets, threads and synchronization or RMI. This is probably one of the most technically challenging assignments. If selecting this project you need very good programming and debugging and design skills.
Task and Staff Scheduler	This project will be great for those considering a project management career; it will give you a head start with scheduling and resource allocation, a common problem in project management. This projects requires some use of GUI and events.

Note: Each group member must be from the same tutelab session.

Project 1: Course Management System

You are required to develop a course management system for a small community college that started offering a degree in Information Technology. Your system should allow various personnel to interact with the system.

- Program coordinator to add new courses which includes course details such as prerequisites, main topics etc.
- Admin to add new course Offerings for specific semester/year specifying the maximum numbers allowed.
- Admin to assign a lecturer to the course offering in current semester.
- Admin can advance the system to the next week. After week 12 the system advances to the next offering semester/year. If there are any outstanding (unentered) results, the system must warn admin that they will be automatically marked as RNF (Result Not Finalised) and ask for confirmation.
- Students to enrol in a course offering subject to meeting prerequisites, maximum load (normally one) and not exceeding the number of electives (normally one).
- Program coordinator to grant exemptions based on prior studies.
- Program coordinator to grant special permissions that allows students to exceed the maximum load.
- Admin and lecturers to view the past performance of any student.
- Lecturers to upload the results for the course offering they are assigned at the end of the semester.
- Students to view the course offering they can enrol.
- All student, admin and lecturer access to be password protected (these are created by an external system).

Assumptions

- All results for current and previous course offerings are to be accessed via electronic files (or database tables).
- You are required to produce a system that can handle future courses and course-offerings.
- Lecturer can upload and modify the results for the current semester only.
- Only results recorded as RNF can be modified for past semesters and once changed to a valid mark, no further changes are allowed.
- This assignment requires only a stand-alone system but it must be designed to be easily extendable to a web-based system in the near future.
- New courses must be added 4 weeks prior to semester commencement
- New course offerings must be added at least two weeks prior to semester commencement
- No exemptions to be granted after the end of first week
- No new enrolments after the second week
- No withdrawals after the third week of semester

Additional Requirements to Enable Testing (prior to launch)

Create some dummy data for 5 students including their results for all the course offerings in the previous 2 semesters.

Bonus Features:
- Use of relational-database
- Extending the stand-alone system to a web-based system

Suggested Initial Test Cases for Milestone 1

1. Test that a student cannot enrol in a course that has no offering in the current semester
2. Test a student with no special permission cannot enrol in more course offerings than the maximum load permitted
3. Test that a student cannot enrol in a course offering without the necessary prerequisite or exemptions
4. Test a student without the necessary prerequisite can still enrol in a course offering if granted the necessary exemptions
5. Test no new courses can be added after 4 weeks prior to semester commencement
6. Test no new course offerings can be added after 2 weeks prior to semester commencement
7. Test no exemptions can be granted after the first week of the semester
8. Test no enrolments are allowed after the second week of the semester
9. Test no withdrawals are allowed after the third week of the semester
10. Test admin, lecturer and students functionalities can only be accessed with valid username and password (to preserve privacy)
11. Test that admin can advance to next semester (after week 12)
12. Test that advancing to next semester automatically marks outstanding results as RNF (after confirmation)
13. Test that only lecturer can enter results
14. Test that past results cannot be modified (unless marked as RNF)

Project 2: Budding Share Market Investor

You are required to write a stock market game to introduce budding investors to learn the risks and opportunities share market-trading presents. Users should be given $1,000,000 (bogus money) initially and be allowed to buy and sell the shares at current price from the ASX. Assume the brokerage cost is made up of two parts: a fixed charge ($50) and a charge that is a percentage of sale or purchase (0.25% for sale and 1.0% for purchase) that may be lowered when high volumes are traded. **Please limit to 6 ASX shares** of your choice.

www.shutterstock.com · 68191825

Users should be provided all of the following functionality:

- Register as a stock market player
- Login as a user
- Open a Trading Account
- List the average price of shares in possession and current number of shares held
- Queue to buy or sell specified number of shares. The transaction should take place if the current share price is lower than or equal to the buy price queued or if the current price is more than or equal to the sell price queued. You may assume that the specified number of shares can be traded at the queued price (not in real life).
- Reject purchases or sales exceeding $600,000.
- Remove the shares queued for buying or selling (before transaction is done)
- Track the movement of share price (at regular 1 hour intervals) and plot the graph
- Specify cut loss options in case of a stock market crash
- List a summary of transactions within specified dates
- View current balance in dollars and current stock value
- Update the balance of top budding traders at the end of each trading week

Admin should be able to:

- Delist a member.
- List players trading amounts exceeding $100,000 within a week.
- Lower the brokerage cost of specific players
- Modify parameters (e.g. trade limits and frequency of share price updates)

Bonus Features

- Encode specific strategies for automatically adding or removing items to buyer and seller queues. For example, if the overall stock market index drops by 1% within an hour all existing shares may be put for sale at current stock prices. You may come up with a number of strategy classes to suit the credit-risk level of investors.
- Develop a separate server that simulates the stock movement to test the behaviour of your strategies. In this case Share market application will connect to this server instead of SGX.

Some Considerations

This project will be of above average difficulty. You may have to work with (delayed) stock prices (unless you are willing to pay). You must have some interest and knowledge about web services, XML, distributed computing etc. If using Java, you may consider using servlets, RMI, web services or sockets.

Possible Domain Classes

Customer, TradingAccount, Transaction, QueuedItem, Admin etc.

**Suggested Initial Test Cases for Milestone 1**

1. Trading can start with valid (registered) user ID and password
2. Trading cannot start without valid user ID and password
3. Test purchasing specified number of share results in customer stock level for the share being increased by the specified number
4. Test whether the brokerage price is computed correctly for a sale
5. Test whether the brokerage price is computed correctly for a purchase
6. Test whether average share price for existing shared held by the customer is computed correctly based on three or more purchases
7. Test whether an item queued can be removed successfully before transaction is done
8. Test that an item cannot be removed from the queue after the transaction is done
9. Test attempts to queue a purchase or sale for amounts exceeding $600,000 are automatically rejected.

Project 3: Single Player Strategic Game

This assignment uses an interactive single player game as a vehicle to demonstrate your understanding of UML design, testing and object oriented programming.

Overview

You are required to develop a stand-alone Monster-Chase-Player-Game using Java. The main objective of the player is to stay alive by outrunning and escaping two cruel monsters. Fortunately these monsters cannot run as fast as players; but they conspire and connive together to trap the player. A player who remains alive for more than the stipulated time (default 100 time units) is considered a winner. In the second milestone allow the player to place up to 2 poisoned food parcels that disappear after 20 units of time. The player or the monsters will be poisoned (slowed to half their current speed) for 20 units of time, if they step on it while the parcel is active. This project will need a team with at least one good programmer. Adequate time must also be spent on testing the system.
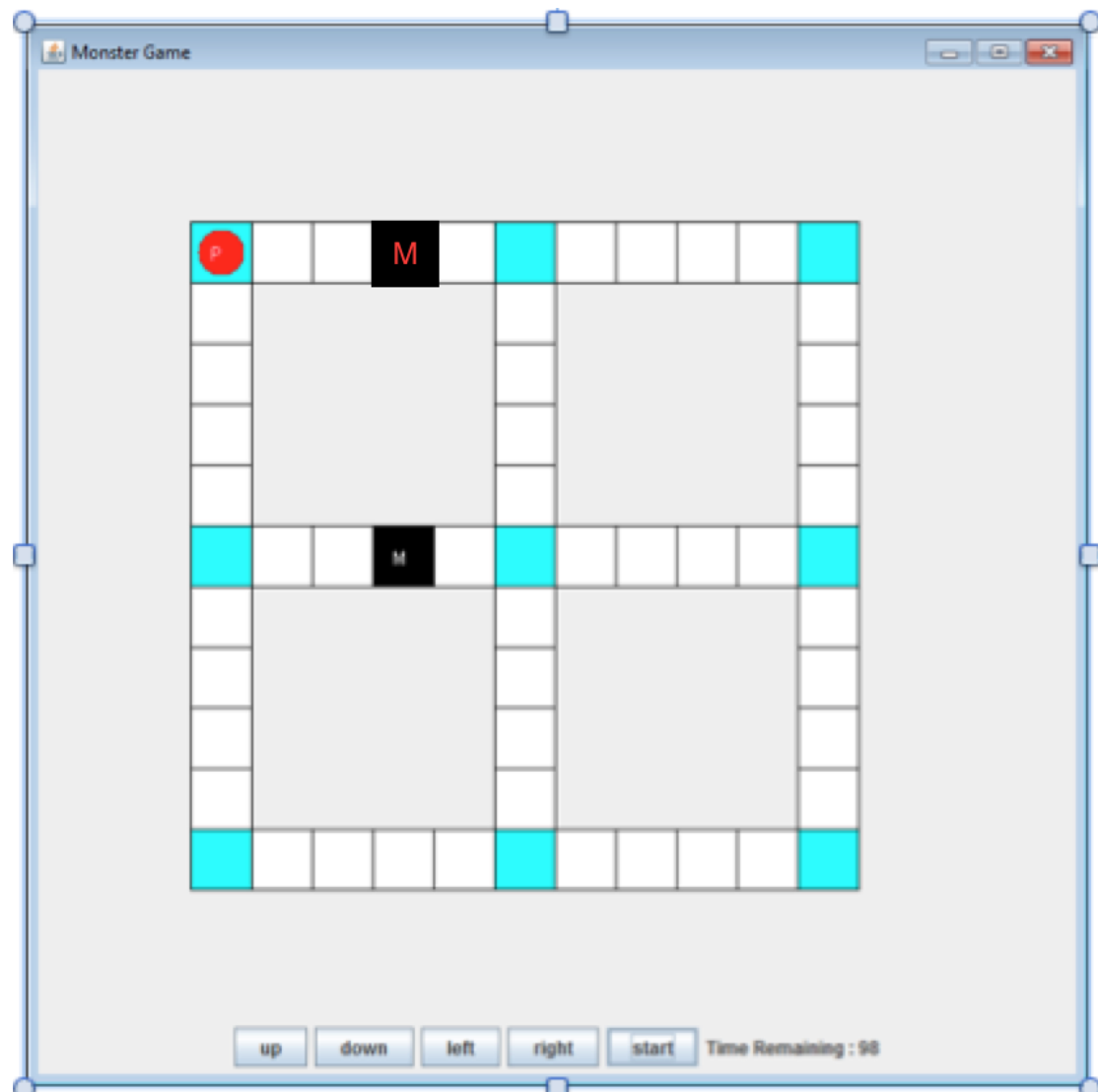
Main Requirements

Client: I want a system that allows

- Users to register – reject repeated user ID
- Users to login
- Players to track scores 1 for winning and 1 for losing
- The current player to commence and play the game
- The current player to pause the game
- A player to place a poisoned food parcel
- A player/monster to consume the poisoned food parcel and halving their current movement speed for 20 units of time
- A player/monster to consume a second parcel while poisoned and halving their current speed (effectively a quarter of the default speed). The duration of each poison is independent (so when the first poison wears off they revert back to a half for the remaining duration)
- System Administrator deregister a player
- System administrator to vary the duration of a game

Domain classes
Player, Monster, Food, Grid, Strategy, Score, Cell

Possible Increments

- Come up with player to move around the board using arrow keys (using text only). Use indexed cells to locate the position of the player.
- Come up with a GUI for Board and a player
- Come up with multiple monsters and strategies for them to collaborate

Bonus Feature
- Allow the System administrator to build a more complex grid (need not be made of rectangles) at run-time and a generic monster strategy able to cope with any grid.

Suggested Initial Test Cases for Milestone 1

1. Verify repeated user IDs are rejected
2. Verify game can start with valid (registered) user ID and password
3. Verify game cannot start without valid user ID and password
4. Test each of the arrow keys to ensure player
   a. moves in the specified direction
   b. does not go outside the grid
   c. does not go inside the grid (hollow region)
5. Test monster(s) does not move out of the grid
6. Test monster move in the expected direction
7. Test the Food laid remains on board for 20 time units
8. Test the player/monster is poisoned (slowed to half current speed) when food parcel is consumed (stepped on) while in active state. Test consumption of second parcel further halves the speed.
9. Test poisoned (slowed down) duration is for 20 time units
10. Test that a food disappears after 20 time units (and does not affect player/monster)

Project 4: Supermarket Support System

A new self-checkout system is introduced in a supermarket, which only allows payments for sales to be made through the company debit card. Currently purchasing and topping up these cards are handled manually using a sales staff. The company has also introduced a scheme to promote loyalty; customers using the card get 1 point, for every $10 spent. Customers are automatically discounted $5 for every 20 points. The system should store the product, customer and employee related details allowing necessary reports to be generated. Data may be stored in relational database or flat files or through serialization. The system should store product, customer and employee related details allowing necessary reports to be generated. GUI is optional for this project.

Current Limitations

A typical checkout terminal consists of a touch screen and a barcode reader but for this assignment all data will be entered through the keyboard (including the customer card number and product ID). Assume the customer is required to specify the customer number thus allowing customer details to be stored.

Project Requirements

A typical customer:

- I want the checkout system with a simple user interface, allowing me to specify ID and quantity or the product name and weight. I want to be able to select the product name from a given list.
- I want to be able check the price of any item by keying in the ID before proceeding with the sale
- I want to check the discounts applicable when I purchase an item in bulk

The sales staff (Kim):

- I want to be able to login to the system the system and override the transaction details (removing item, cancellation) in case a customer has problems.

The warehouse staff (Frank):

I want to be able to replenish stock levels before placing items received on the shelves.

The manager (Tim): I want a system that:

- Maintains unit-price, stock level, replenish-level and reorder quantity for all items.
- Maintains supplier details for all products.
- Allows me to override the standard price for a specific product when there is a promotion
- Allows me to offer special discounts for bulk sales (for example, 10% discount for 5 items, 20% discount for 10 items etc.) on specific products
- Allows me to modify these percentages and item numbers through the interface.
- That automatically places a purchase order for all items below replenishment level
- Allows me to generate sales report for the specified period
- Generate supply report (Payments for supplies are out of scope)
- List products generating the most revenue.

Bonus marks (one of the following)

Extend into web-based or client server applications allowing concurrent access. This extension will require the use of sockets and/or RMI and/or Servlets access.

Interface to an external barcode reader

Possible Domain Classes

System**,** Sale**,** SalesLineItem**,** Product, Location**,** Customer, Credit- Card, Employee, Employee, Manager, SalesStaff, Supplier

Suggested Initial Test Cases for Milestone 1

1. Verify performing a sales reduces the stock level for all products in the sale
2. Verify replenishing stock increases stock level by the specified amount
3. Verify sale price is computed correctly based on sale line items
4. Verify sale price is computed correctly for discounted items
5. Verify sale price is computed correctly for items offering discounts for bulk sale
6. Verify sale price is not affected for non-discounted items
7. Verify The loyalty points are allocated correctly
8. Verify Maximum discounts are automatically given based on current loyalty points at the end of transaction
9. Verify discounts are computed correctly when loyalty points are combined with bulk discount
10. Verify discounts are computed correctly when loyalty points are combined with some promotional items

Project 5: Traffic Signal Simulator

Design and implement a system, which will allow staff of traffic engineering department to simulate traffic signal timing for up to 10 intersections. Your system should allow creation of different configurations (using a rectangular grid) with equal distances between them. You can also assume the speed limit though variable is constant throughout the domain at any given time. The main purpose of the system is to determine near optimal traffic signal timing (the right amount of time for red, green and yellow) at each intersection at a given time (experimentally) to allow smooth flow of traffic without accidents. (You may make the necessary assumptions). This project requires good use of OO principles as well as simple animation and graphics.
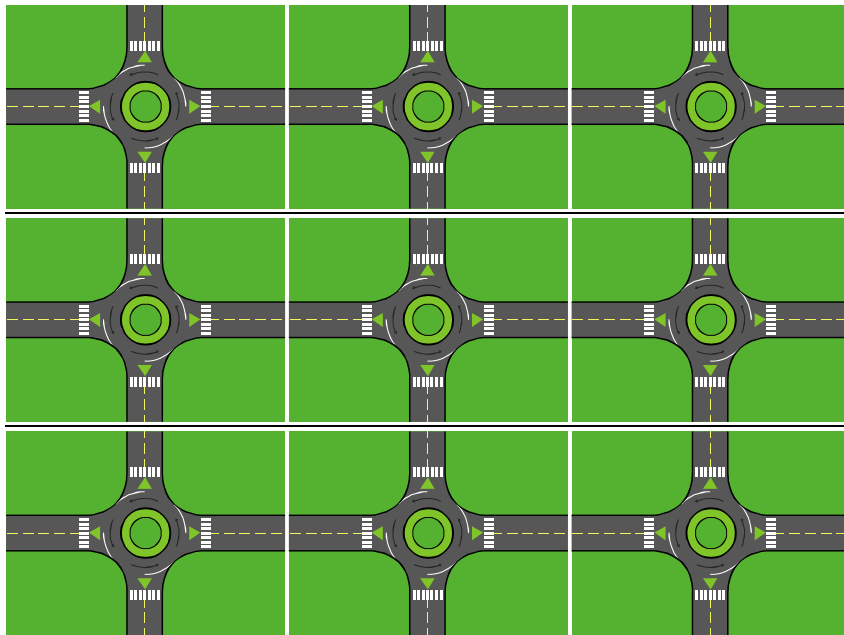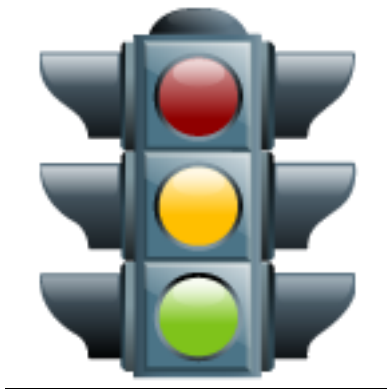
Requirements

- Allow users to build rectangular grids
- Allow user to specify the frequency of cars entering the roads such as 1 car every 3 time units
- Allow users to specify the unit of time for green light in both directions in time units (assume amber is a fixed number of units)
- A car should be allowed to move if the traffic light ahead is red or it is blocked (by a car ahead)

Simplifying Assumptions

- Assume the speed of each car is fixed – such as moving a car by half a car each time unit
- Assume cars are not allowed to turn
- Each junction has 4 interconnected traffic lights (green and amber along one direction corresponds to red along the perpendicular direction.
- Cars need not be modelled once reaching the end of the road

Bonus Marks (one of the following)

- Allow users to specify turning cars – you may assume there are two lanes along each road
- Automatically determine near optimal traffic signal timing for a given configuration.

Possible Classes

Car, CarQueue, TrafficLight, Road, Grid class, Place class (a road may be divided into many places – the amount of space needed to hold a car).

Note: You need not show a GUI version for milestone 1

Suggested Initial Test Cases for Milestone 1

1. Verify the light changes from red to amber to green and from green to amber to red
2. Verify the traffic light for cars coming in the opposite direction in any junction is the same
3. Verify the traffic lights along crisscrossing roads in the same junction are either green and red or amber and red.
4. Verify that no car moves through a red light.
5. Verify that the first car waiting at the junction starts moving into the junction only when the light turns green and junction is free of traffic.
6. Verify that no car moves over another car
7. Verify the number of cars entering at the beginning of the road is the same as the number of cars exiting at the road.

Project 6: Chess Like Game

You are required to develop a new strategic game similar to Chess, being made up of pieces such as rooks, bishops and knights. The bishop can move up to two pieces but only diagonally. The rook can move up to two pieces but only vertically or horizontally. Rook and bishop pieces cannot move over other pieces. A knight moves two squares along horizontal or vertical direction and another square perpendicularly making an "L" shape. A knight can also move over other pieces. Whenever a player removes an opponent's piece 5 points are collected. The game stops when the one play takes out all the pieces of the opponents or after n moves. The player with the most number of points is the winner.
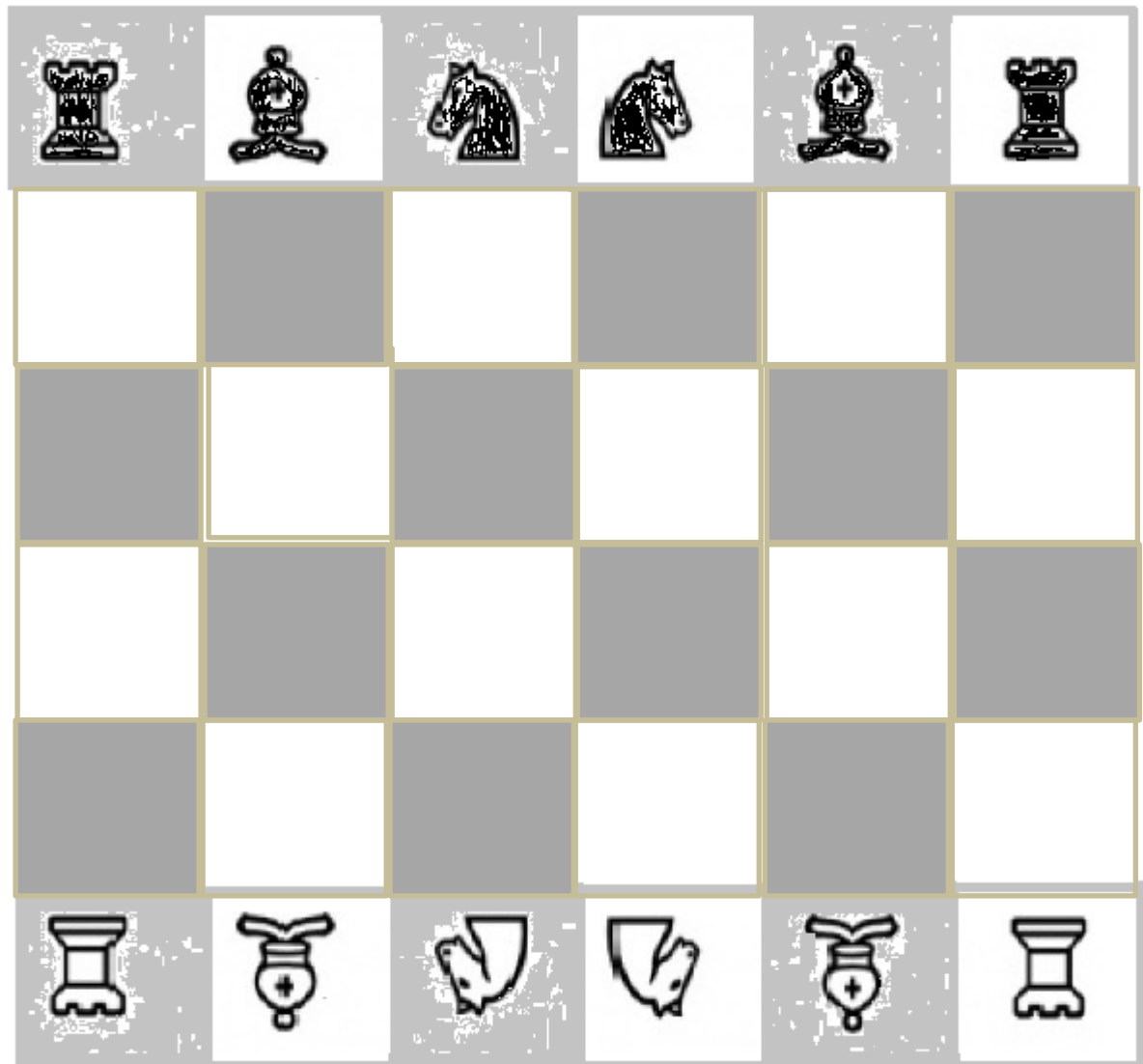
Other Requirements

1. All players must be registered
2. Both players must be logged in before they can commence.
3. The two players must be different.
4. The maximum number of moves must be determined averaging the desired number specified by both players before the game commences.

Enhancements for Milestone 2

- Allow different kinds of pieces of the same player to merge thus combining their moving capabilities, or split when such capabilities are no longer needed. Both merging and splitting can be done only when it is the player's turn and is considered as one move.
- Show all valid moves a piece can take graphically.

Suggested Initial Test Cases for Milestone 1

1. Attempt to register two members with the same login ID fails.
2. Attempt to login with invalid login ID and password fails
3. Attempt to login with valid login ID and password succeeds
4. Attempt to login with the same valid ID and login fails
5. Attempt to move a bishop piece diagonally up to two cells, succeeds when there are no pieces.
6. Attempt to move a bishop piece diagonally bypassing other pieces fails.
7. Attempt to move a bishop piece diagonally more than two cells fails when there are no pieces.
11. Attempt to move a bishop piece in any direction other than diagonally fails
8. Attempt to move a rook piece in horizontal or vertical directions up to two cells succeeds when there are no pieces
9. Attempt to move a rook piece in horizontal or vertical directions bypassing other pieces fails
10. Attempt to move a rook piece in any direction other than horizontally or vertically fails
11. Attempt to move a knight piece along L shape (2 + 1) succeeds
12. Attempt to move a knight piece to any other cell fails
13. Attempt to move a knight piece over another piece succeeds
14. Player points are increased by 5 whenever an opponent piece is removed

Note: You need not show a GUI implementation for milestone 1 (you may display the pieces in a console – use the characters b k r and B K R to represent the white and black Bishop, Knight and Rook respectively).

Bonus Marks (choose 1)

- Provide a warning to the player if a move is likely to cause a loss of a piece in the next move by the opponent. The player can choose to ignore the warning.
- Extend it into a multiplayer (network) game
- Allow the last two moves to be undone (each player is given one chance to revert to the previous state if they make a careless move).

Project 7: Vacation House Exchange Club

Design and implement a House Exchange club which allows members to use houses of other members when they go on vacation on a weekly basis (Friday to Friday). There is an initial entry fee of $1000 when registering as a member, which earns the new member 1000 credit points. Initial registrations must capture the following information:

Member Details

- Name (personal)
- Address (personal)
- Phone number (personal)

Member House Details

- City and suburb
- Distance to city centre
- Availability of public transport
- Number of rooms
- Air-conditioning/heating facilities
- Swimming pool
- Optional video/images of the house
- Points needed for a week (Friday to Friday)
- Discounts for long stays (over multiple weeks)

Each member earns points when the house is let to another member, which can subsequently be used when the member wants to travel to other cities.

Ratings
The system will track two separate ratings which can vary from -10 to +10. The utility rating will reflect the utility level derived based on the average of responses of other members who have occupied the house in the past. In addition, users may also specify a fair week-value for the property in terms of credit points and write a review. The occupier-rating is derived by averaging the ratings of all owners who had let their houses to be used (how well that member has taken care of the house).

Letting the House
Any member can let his or her house by specifying the period and the minimum occupier (member) rating for potential occupants. The minimum occupier rating can thus be used to prevent the house being rented to poor occupants or those for which no history is available.

Viewing Houses to Let
Any member can list the details of all eligible houses in a specified week and city. The listing should consist of only those houses for which the member has adequate credit points and occupier rating. The information should also include the utility rating and the averaged-fair-value reported by previous occupants. Members should also be given the option to view the reviews on any of the listed properties.

Occupancy Request

Any member can request to occupy any property for which they are eligible. When a request is made, the owner will be able access the rating of the interested occupier and may choose to accept or reject the request. No other member can make a request while a current request is being considered. If no response is sent out within 24 hours it is considered to be rejected. If the property owner responds positively, the contact details will be revealed to the requestor after deducting the points, allowing occupier to make the necessary arrangements.

Non-Members

Non-members can view all house details (but not their availability) to encourage non-members to join.

Suggested Initial Test Cases

1. Verify Member and House details added are captured correctly
2. Verify new members are given 1000 credit points
3. Verify non-members can only view property details but not availability.
4. Verify a member can only view properties for which he or she meets the occupier-rating specified
5. Verify that the occupier rating is the average of ratings specified by all previous owners
6. Verify that utility rating is the average of ratings specified by all occupiers
7. Verify credit points for the occupier member is decreased correctly after a transaction
8. Verify credit points for the owner member is increased correctly after a transaction
9. Verify the status of a property is put on hold when a request for occupation is made
10. Verify the status of a property put on hold reverts to available if owner does not respond within 24 hours
11. Verify the status of the property is changed after a successful transaction
12. Verify that the fair value is based on the average value estimated by previous occupiers

Enhancements for Milestone 2

1. Allow members to specify their vacation requests and allow interested owners to respond
2. Allow all data to be persisted (Database or ORM tools)
3. Incorporate a map for the house

Bonus (one of the following)

- Encrypt all data stored in the database and incorporate options for payment
- Use of design patterns (at least 5)

Specific requirements

Note that while this option is web based, it must use java/groovy on grails for your implementation with additional frameworks (ie. a pure php implementation is not acceptable) and your solution must use Object Orientation.

Project 8: Multiplayer Interactive Monster Game

You are required to write a simple multiplayer game where a monster (controlled by the server) chases and eats up the player pieces moving in the predefined grid shown below. The players and monster can also move along the arrows shown below to end up along the opposite edge. The aim of the game is to escape from the monster and to get all the opponents eaten up by blocking their escape routes. Each player should be allowed to move using appropriate arrow keys. The first player starting the game should be allowed to specify the number of players (between 2 and 4). The game should start when the required number of players joins in. Each player should be allowed to specify one of the four remaining corners as the starting cell (top-left, top-right, bottom-left, bottom-right) in the order they join the game. The monster should initially be placed in the cell at the centre and be made to move towards the nearest player (least number of cells in between). When two or more players are at equal distance, the monster may choose to move towards any one of them randomly. A player can block any other player by being stationed along the escape route (with the aim of getting the other player eaten). A monster must wait for a period of time for the food to digest before starting the chase again. The game should stop when only one player is left, who is considered to be the winner. You may use either sockets and threads (using your own protocol) or Java RMI (easier) to implement the game.

Player 1   Player 2
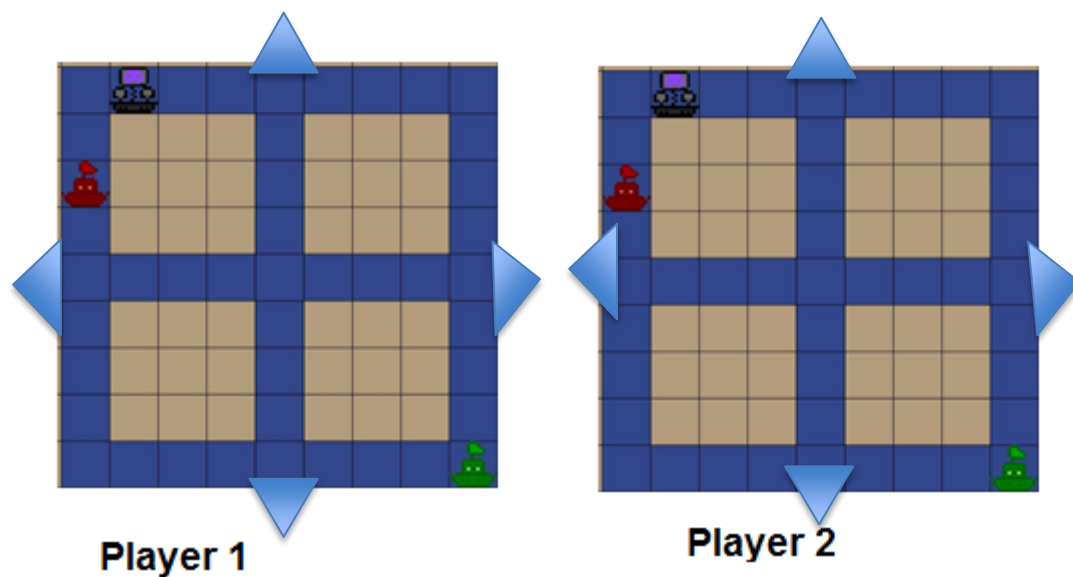
Suggested Initial Test Cases

1. Verify that the four arrow keys moves the payers in the corresponding directions (up, right, left and down).
2. Verify that players cannot move outside the grid
3. Verify that monster does not move outside the grid
4. Verify that player positions in different clients are identical (requires game logic in server to be broadcasted to all clients)
5. Test that players cannot move over each other (server must synchronize move events)
6. Verify that monster always moves towards the nearest player
7. Verify that the game does not start until the specified number of players join
8. Verify that the game stops when only one player is remaining

Enhancement

1. Allow Player Registration and Login
2. Track player scores and allow top 10 players to be displayed based on the percentage of games they win.

Bonus

1. Allow the board grid to be created at run-time.

Project 9: Task and Staff Scheduler

Assume a project manager has approached you to develop a customised project management software over the summer vacation. He does not insist on a GUI based version (but prefers one) as he is more interested in solving his immediate problems before the new year starts. He has a number of projects lined up and he wants to reduce the overall costs. This project is ideal for any student interested in project management – it also has interesting interactions between different project, activity and employee classes and will allow you to develop some heuristic algorithms.
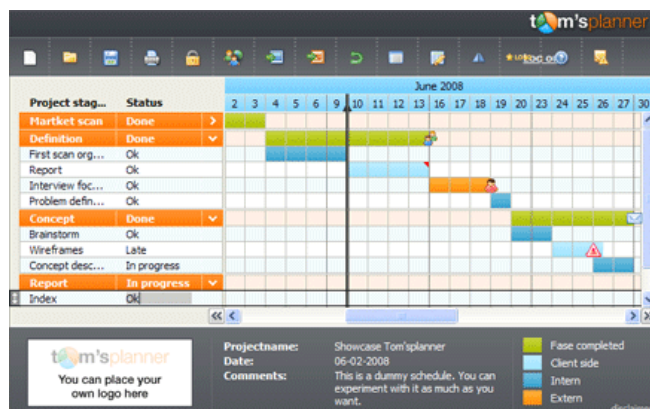
Requirements

Jack (the Project Manager): Currently I am allocating staffs to activities in an ad hoc manner resulting in project delays and cost overruns. I need you to design and develop a Task and Staff Scheduler that assists me to complete the projects I manage within time and budget. The project involves two parts: scheduling activities and allocating staffs to activities. For each activity I can estimate the duration in weeks, the type of skills needed and the number of staff needed. For each such activity I can also specify any dependencies on previous activities (if any). The critical path and the earliest project completion time are computed based on dependencies on previous activities and the duration of activities. My main problem however, is allocating staffs to activities as staffs may be working on a number of projects in parallel; staff in our companies can be working either 20%, 40%, 60% 80% or 100% on a project reflecting the number of days per week they are engaged in the projects. I select the staffs for an activity based on whether they have all the skills needed. Very often full-time staff are underutilized in specific weeks while in some other weeks we had to hire many contract staffs. I would like to minimize the number of contract staffs as they expect premium rates to be paid on an hourly basis. This year, I want a new system that will help me plan all the activities for all the projects at the beginning of the year and maximise the usage of our permanent staff. I want the system to compute the expected project completion date based on the start date for the project (first dummy activity in the project), the duration and dependencies (predecessors) for each activity. For each activity, I also estimate the average number of staffs required (can be fractional) and their required competency levels in needed skills. Secondly, I would like to have the option to move the activities not lying along the critical-path and to experiment assigning different staff with the intention of minimizing the dependencies on external staff.

I would like your system to come up with some initial staff assignment using some simple heuristics, such as assigning the least experienced full-time staffs first to activities (as more experienced staff can fit into many different and complex activities). After assigning all the

full-time staffs, assign inexpensive, less experienced contractors first. I would still like to have an option to change these staffs around manually to do some trial-and-error.

Jim a Full-time developer: I would like a system where I can specify at the beginning of the year the weeks that are blocked (I am not available due to annual leave or other duties). I would also like to specify weeks where I m partially blocked off as I have a fair amount of non-project related duties. In such weeks I would like to specify my availability as 20%, 40%, 60%, 80% and 100%. Though I am multi-skilled my level of competency and experience vary significantly. I would like to specify my competency level explicitly for different skills in a scale of 1 to 10, so that the manager can select me for activities where I can be really productive. For example, I am highly competent in Java (10) but I am weak in Python (4) and domain modelling (4).

Timothy an External Contractor: In the past we had to charge high rates as there were a number of weeks in a row where we got no work at all followed by intensive periods. Most contractors including myself prefer to work on an activity lasting for at least two weeks or more. We tend to be more productive and happy when working for a longer period, and are even willing to charge less. I would therefore like the system to allow me specify different rates for contracts lasting 1 week, 2 weeks and 4 weeks or more.

Possible Domain Classes

Project, Activity, Staff, Contract Staff, Full-time staff, Competency, BlockedWeeks, Skill.

Bonus Requirements

Compute near optimal schedules by minimizing the involvement of contract staffs and maximizing the utilization of full-time staff.

Suggested Initial Test Cases

1. Verify there are no cyclic dependencies in the activities specified, such as activity B cannot start until A, C cannot start until B and A cannot start until C.
2. Verify that no activity in any of the projects starts before its dependent (predecessor) activities are complete.
3. Verify the earliest project completion time is computed correctly.
4. Test the commitment of a staff to an activity is either 20%, 40%, 60%, 80% or 100%.
5. Test that no staff assignment for an activity results in a staff being assigned more than 100% commitment in all the projects combined at any one week.
6. Verify that the staff skill levels are in the range 1 to 10.
7. Verify that in every activity assigned to a staff, the staff competency level is greater than or equal to the required competency level in every skill required for that activity.
8. Verify partial availability of full time staff are taken into account during project assignment.
9. Verify that any attempt to assign a staff to activity in a week that he or she is unavailable is rejected.