Mini-project topics
Object-Oriented Programming

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn

Important note:

- For each topic provided below, it is compulsory for you to **strictly** follow all the **specifications** stated below. The reference links are there to clarify/illustrate the specifications only, in case of conflict between the specifications and the reference, the specifications should be prioritized.
- Some topics may require a bit more field-specific knowledge than others (topics related to physics, electronics, …). In which case, we have provided summarizations and formulae for you to quickly and aptly grasp the knowledge needed. This way, you can focus on the object-oriented aspect of the project, which is the most important goal.

Caution: Since we will only assess the use of object-oriented programming in this project, you should not spend too much time on **interface design**, which **won't be graded.**

How to conduct mini-projects:

- You will conduct mini-projects with your teammates. Each team must have from 3 to 4 members. The monitor may play a random game for allocating the topic to teams if there is any conflict.
    - Each team will select one of the following topics and each topic must be chosen by **4 teams**
    - In case more than 4 teams choose a specific topic, the teacher's assistant will request any group to randomly select a topic that hasn't yet reached the required number of groups.
    - Your mini-project will be 50% for the midterm score, and 50% for labs (5 labs)
- Your midterm score will contribute 50% to your grade in the course.
- You will be given an **individual score** for the mini-project although you do it with your teammate. Please remember to clarify **the responsibility of each member of your team**, otherwise, **your teamwork score will be very low**.

1. Visualization of operations on tree data structures

Overview: Tree is a useful data structure with lots of applications in computer science. In this project, you will design a program to display and explain some basic operations of four types of tree structure.

Basic knowledge:

- Generic tree (no special properties): A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges and contains no cycles.
- A Binary Search Tree (BST) is a binary tree in which each vertex has only up to 2 children that satisfy the BST property: All vertices in the left subtree of a vertex must hold a value smaller than its own and all vertices in the right subtree of a vertex must hold a value larger than its own (we have an assumption that all values are distinct integers in this visualization and small tweak is needed to cater for duplicates/non-integer).
- An Adelson-Velskii Landis (AVL) tree is a self-balancing BST that maintains its height to be O(log N) when having N vertices in the AVL tree. The heights of two subtrees from its root nodes are limited to a "certain distance". The child nodes are rebalanced when the height difference exceeds a certain distance.
- Balanced binary tree: A balanced binary tree has the properties of both a BST and an AVL tree

Operations on the above tree structures: create, insert, delete, update, traverse.

Specifications:

- GUI: You can refer to this source for some ideas: https://visualgo.net/en/bst
- Design:

+ We only consider undirected-weight trees, with integer node values and no duplicated node values allowed.

+ For the balanced tree and balanced binary tree, the maximum difference in distance from the root of the leaf nodes must be chosen by the user

+ On the main menu: title of the application, navigation bar for user to choose between the four types of tree, help menu and quit

- User must select a type of data structure before getting into the visualization
- The help menu shows the basic usage and aim of the project
- Quit button exits the application. Remember to ask for confirmation

+ In the visualization

- Users can choose to visualize one of six operations, by selecting an option on the operations menu, and then providing the necessary parameter.
  The description of the operations is as follows:

Operations	Parameters	Description
Create	None	Create a new empty tree
Insert	Value of parent node, value of new node	Add the new node with a specified value as a child of the specified parent node
Delete	Value of the node	Delete the node from the tree
Update	Current value of the node, new value of the node	Change the node with current value to new value

Traverse	Algorithm (DFS or BFS)	Traverse all nodes in the tree (highlight the current node in each step of traversal)
Search	Search value	Search for the node value in the tree

- When an operation starts to execute, on the code panel, the pseudo-code (or actual code) should be displayed, and the currently executing line is highlighted to help the user keep track of the process. On the bottom bar, the user can see the progress bar of the executing operation and choose to pause, continue, or go backward or forward a step in the execution.
- The user can also undo or redo operations from the bottom bar.
- Always have a Back button for the user to return to the main menu at any time

2. Demonstration of sorting algorithms on an array (1)

Overview: Array is the most basic structure of computer science. Most operations as well as other data structures are built and performed on an array. In this project, you will make an application to explain three sorting algorithms on an array: bubble sort, heap sort, and shell sort.

Basic knowledge: bubble sort, heap sort, and shell sort on an array

Specifications:
- GUI: you can freely design your own GUI. However, since the basic aim of the project is to develop an application based on OOP, focusing on the interface is not required.
You can refer to these sources to have some ideas:
https://visualgo.net/en/sorting
https://www.youtube.com/watch?v=nmhjrI-aW5o
https://www.youtube.com/watch?v=MtQL_ll5KhQ
https://www.youtube.com/watch?v=SHcPqUe2GZM
- Design:
+ On the main menu: title of the application, 3 types of sort algorithms for the user to choose, help menu, quit

- Users must select a sort type to start the demonstration.
- Help menu shows the basic usage and aim of the program.
- Quit option exits the program. Remember to ask for confirmation.

+ In the demonstration

- A button for creating the array: The user can choose to randomly create an array or input an array for the program.
- A button for starting the algorithm with the created array. Remember to show clearly each step of the sorting.
- A back button for the user to return to the main menu at any time.

Note: You MUST use a pure array for this project. If you use any Java implementations, you must design your wrapper to show your OOP design in the project
Hints: To design this program with OOP methods, treat each sort type as a class – not an array. In this way, you will find there are some similarities between the sortings (input, functions, attributes, etc.)

3. **Demonstration of sorting algorithms on an array (2)**

Overview: Array is the most basic structure of computer science. Most operations as well as other data structures are built and performed on an array. In this project, you will make an application to explain three sorting algorithms on an array: merge sort, counting sort, and radix sort.

Basic knowledge: merge sort, counting sort, and radix sort on an array

Specifications:
- GUI: you can freely design your own GUI. However, since the basic aim of the project is to develop an application based on OOP, focusing on the interface is not required.
You can refer to these sources to have some idea:
https://visualgo.net/en/sorting
https://www.youtube.com/watch?v=JSceec-wEyw&t
https://www.youtube.com/watch?v=7zuGmKfUt7s
https://www.youtube.com/watch?v=nu4gDuFabIM
- Design:
+ On the main menu: title of the application, 3 types of sort algorithms for the user to choose, help menu, quit

- Users must select a sort type to start the demonstration.
- Help menu shows the basic usage and aim of the program.
- Quit option exits the program. Remember to ask for confirmation.

+ In the demonstration

- A button for creating the array: The user can choose to randomly create an array or input an array for the program.
- A button for starting the algorithm with the created array. Remember to show clearly each step of the sorting.
- A back button for the user to return to the main menu at any time.

Note: You MUST use a pure array for this project. If you use any Java implementations, you must design your wrapper to show your OOP design in the project
Hints: To design this program with OOP methods, treat each sort type as a class – not an array. In this way, you will find there are some similarities between the sortings (input, functions, attributes, etc.)

4. **Design a Genetic Algorithm to solve TSP**

Overview: Traveling salesman problem (TSP) is a famous problem in Computer Science and may be familiar to you. The problem statement is simple, given a graph $G = (V, E)$, where V is the set of nodes, E is the set of edges, there is additional information about distances between nodes. The goal of this problem is to output the shortest route starting from V_{start} and ending at V_{start}, such that each node is visited once. In this project, you are asked to implement a Genetic Algorithm (GA) to determine the (near-)optimal route.

Basic Knowledge: Algorithm, Data structure (Note: you don't need advanced knowledge about the evolutionary algorithm, you can deploy this project without any prior knowledge of GA)

Specifications:
- Algorithm overview: In this project, you must implement a GA algorithm to solve TSP, your proposed method may be simple and you should not focus too much on advanced GA techniques. Here is a simple guide for deploying this algorithm to solve the TSP:

- Step 1: Design an encode-decode mechanism for each route. The simplest way is to use permutational encoding, for example, to represent a route $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$, we can use $(4, 5, 1, 2, 3)$ or $(0, 4, 5, 1, 2, 3)$, … We call this representation as 'chromosome'
- Step 2: Create a population with N individuals, each individual is a chromosome of a route. For example, individual $A = (4, 5, 1, 2, 3)$
- Step 3: Design a crossover mechanism: Given input as two individuals, output two offspring. You can refer to many mechanisms, one simple approach is one-point crossover, you can refer to it here: https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm), and also many other sources.
- Step 4: Design a mutation mechanism: There are many methods, one simplest way is to swap two genes in a chromosome
- Step 5: Design a parent's selection and mutation implementation.
- Step 6: Repeat selecting parents and mutating to evolve the population.

You can refer to a full-detail algorithm at: https://www.youtube.com/watch?v=3GAfjE_ChRI
- GUI: you can freely design your own GUI to demonstrate the found route.
You can refer to these sources to have some ideas:
https://www.youtube.com/watch?v=94p5NUogClM
https://www.youtube.com/watch?v=u_WHKDYHltE
- Design:
+ On the main menu: title of the application, 3 buttons for the sort algorithms for the user to help, solve the TSP, and quit

- Help menu shows the basic usage and aim of the program
- Quit option exits the program. Remember to ask for confirmation

+ In the demonstration

- A button for starting the simulation, namely, "Start"
- A visualization for the individual that currently holds the best fitness, in array-structure
- A visualization for the individual that currently holds the best fitness, in graph-structure

Note: If you use any Java implementations, you must design your wrapper to show your OOP design in the project.

5. **Interactive simulation of the composition of forces**

Overview: Newton's laws of motion are the foundation of dynamics. These laws provide an example of the breadth and simplicity of principles under which nature functions. In this project, you will create a simple interactive simulation for demonstrating Newton's laws of motion.

Basic knowledge: Gravitation force, normal force, friction, Newton's laws of motion, Newton's law for rotation

Specifications:

- GUI: You can refer to this source: http://phet.colorado.edu/sims/html/forces-and-motion-basics/latest/forces-and-motion-basics_en.html ("Acceleration" module)
- Design: The design of this simulation is closely similar to the one in the reference above

+ In the simulation, the user controls a physical system. The system includes three components: one main object, the surface (which is always horizontal), and an actor who can apply a horizontal force on the object. The user can control all the components of the physical system and observe the motion of the main object, specifically:

- The main object: The user has two options for the main object, either a cube-shaped object or a cylinder-shaped object. For each type of object, the user can also specify the desired parameter as follows:

Object type	User-control parameters
Cube-shaped object	- Side-length (cannot exceed a maximum threshold) - Mass
Cylinder-shaped object	- Radius (cannot exceed a maximum threshold) - Mass

  To set up the main object in the system, the user can drag an option from the object menu on the bottom left onto the surface, then click on the object, and provide the parameters in the context menu that pops up.
- The actor: The actor always applies force on the center of mass of the main object. Unlike the reference, you don't need to represent the actor with an actual figure, instead, the actor is represented by the force it applies. This force can be represented with a horizontal arrow. To control the strength and direction of the applied force, the user can use the bottom center panel by using the sliding bar or specify the number of Newtons in the textbox.
- The surface: The user can control the friction coefficients of the surface in the bottom right panel. There are two friction coefficients: static friction coefficient and kinetic friction coefficient. For each coefficient, the user can control its value through a sliding bar and a text box. Note that the value of the static coefficients must be higher than the value of the kinetic coefficient.

+ To simulate motion, we recalculate the position of the main object after each time interval Δt.
+ Throughout the motion simulation process, the user can

- Change the applied force as well as the friction coefficients of the surface.
- Pause, continue and reset the simulation.

- Choose to show or hide detailed information such as the forces, the sum of forces, the values of forces, the mass, speed, and acceleration of the main object through the corresponding tick-boxes on the panel on the upper right:
    - The forces and sum of forces are displayed as arrows like in the reference.
    - The masses are displayed as text on the main object like in the reference.
    - The speed and acceleration are displayed as text in the upper left corner.

Knowledge summary:

The solution and formulae to calculate the motion of the main object will be presented below:

The main object is under the effect of four forces:

Forces	Applied point	Direction	Value
Gravitational force	Center of mass	Vertical, downward	Mass x 10
Normal force	Point of contact with the surface	Vertical, upward	Equal to gravitational force
Actor's applied force	Center of mass	Horizontal	
Friction	Point of contact with the surface	Horizontal, opposite direction of motion	Depending on the applied force and the shape of the main object

The value of friction is dependent on the value of the applied force as follows:

Cases of applied force	Object shape	Value of friction
Applied force <= (Normal force x Surface's static friction coefficient)	Cube	Equal to the applied force
Applied force > (Normal force x Surface's static friction coefficient)	Cube	(Normal force) x (Surface's kinetic friction coefficient)
Applied force <= 3 x (Normal force x Surface's static friction coefficient)	Cylinder	Equal to applied force / 3
Applied force > 3 x (Normal force x Surface's static friction coefficient)	Cylinder	(Normal force) x (Surface's kinetic friction coefficient)

The position of the main object after each time interval Δt is calculated as follows:

- The translational motion of the main object is calculated as follows (assuming x, v, x', v' are the previous position, the previous velocity, the new position and the new velocity of the main object, respectively):

$$a = \frac{|Applied\ force - friction|}{mass}$$

$$v' = v + a * \Delta t$$

$$x' = x + v * \Delta t$$

- If the main object is cylinder-shaped, it also has an additional rotation motion. The rotation motion is calculated as follows: (assuming θ, ω, θ', ω' are the previous angular position, the previous angular velocity, the new angular position and the new angular velocity of the main object, respectively):

$$on\gamma = \frac{Friction}{\frac{1}{2} mass * radius^2}$$

$$w' = w + \gamma * \Delta t$$

$$q' = q + w * \Delta t$$

6. Demonstration of types of cell division

Overview & gameplay: Chu kì tế bào và các hình thức phân bào - Ôn Tập Sinh Học 10 - Để học tốt (dehoctot.vn)

Basic knowledge: Cell cycle, Direct cell division or amitosis, Mitosis, and Meiosis

Specifications:
- GUI: You can freely design the GUI in your favor. However, this project focus on structuring the application with OOP design; therefore, too much focus on the interface is not necessary
- Design: The application must have these functions:

+ On the main screen: Title of the application, options to choose the type of cell (prokaryotic or eukaryotic), help menu, and quit.

- For each type of cell, the user can choose to investigate one of the cell processes (Amitosis, Mitosis, and Meiosis) for demonstration.
- The help menu shows the basic usage and aim of the application.
- The quit button exits the application. Be sure to ask for confirmation

+ In the demonstration:

- Display the cell components. Note that each component has different functions, you should display and explain them.
- One button to start demonstrating the progress of cell division through separate phases.
- On the bottom bar, the user can see the progress bar of the executing phase and choose to pause, continue, or go backward or forward a step in the execution.
- The user can also replay the process.
- Always have a Back button for the user to return to the main menu at any time.

7. Demonstration of types of COVID-19 virus and its mechanism

Overview: COVID-19 has been spreading all over the world and there is a need of understanding the different types of the virus, as well as the way they infect to have the basic knowledge to prevent them.

Basic knowledge:
- Basic structure of virus:
+ Every virus has 2 basic elements: acid nucleic and capsid.
+ Based on their structure, viruses are divided into 2 categories: with and without lipid envelope.
    - Virus without the envelope will dissolve its capsid when reaching the target cell
    - Viruses with envelopes usually have anchors, called glycoprotein. The mechanism for infecting, in this case, is by lock–key: when reaching the host cell with the suitable outer structure, it uses its glycoproteins to attach, then injects its acid nucleic into the cell

You can refer to these materials for more information:
https://en.wikipedia.org/wiki/Virus#Structure
https://www.vinmec.com/vi/tin-tuc/thong-tin-suc-khoe/suc-khoe-tong-quat/dac-diem-cau-tao-cua-virus-gay-benh/
https://opentextbc.ca/microbiologyopenstax/chapter/the-viral-life-cycle/

Specifications:
- GUI: You can freely design the GUI in your favor. However, this project focus on structuring the application with OOP design; therefore, too much focus on the interface is not necessary
- Design: the application must have these functions:
+ On the main screen: Title of the application, options to choose between virus with lipid envelope and virus without lipid envelope, help menu and quit
    - User can choose to investigate one of the two types of viruses in the main menu to start the application
    - After choosing the desired type, the application will show a variety of viruses for the user to select (for example, after choosing a virus with lipid envelope, the application displays 2 viruses: HIV, COVID, and Rotaviruses for the user to choose. The choice of viruses to demonstrate depends on you)
    - The help menu shows basic usage and aim of the application
    - The quit button exits the application. Be sure to ask for confirmation

+ In the demonstration:
    - Display the structure of the virus. Note that each virus has a different structure, you should display and explain them.
    - One button to start demonstrating the progress of the virus infecting the host cell. Different viruses have the same basic mechanism of spreading with a minor difference - remember to show that
    - There is always a return button for the user to get back to the main menu at any time.

8. Traditional game: Ô ăn quan

Overview & gameplay: https://hocvienboardgame.vn/huong-dan-tro-choi-o-an-quan/

Demo game: http://choinhanh.vn/game-tri-tue/o-an-quan

Specifications:
In this project, you will make an application for 2 users to play the traditional game Ô ăn quan.
- GUI: You can freely design your own UI; however, since this project aims to design a program using OOP, there is no need to pay too much focus on the interface. You can also use some image references of the web game in the link provided, or look for any materials you like.
- Design: The application must have these functions:
+ On the main screen:
    - Start: start the game. For convenience, you do not have to create different difficulties
    - Exit: exit the program. Be sure to ask users if they want to quit the game
    - Help: Show guide for playing the game

+ In the game:
    - Gameboard: The game board consists of 10 squares, divided into 2 rows, and 2 half-circles on the 2 ends of the board. Initially, each square has 5 small gems, and each half-circle has 1 big gem. Each small gem equals 1 point, and each big gem equals 5 points.
    - For each turn, the application must show clearly whose turn it is. A player will select a square and a direction to spread the gems. He got points when after finishing spreading, there is one empty square followed by a square with gems. The score they got for that turn is equal to the number of gems in that followed square (see the gameplay for more details about streaks)
    - The game ends when there is no gem in both half-circles. The application must notify who is the winner and the score of each player.
    - For simplicity, you do not have to build a bot to play with human

9. Traditional game: Cờ gánh

Overview & gameplay: Hướng dẫn cách chơi Cờ Gánh (Cờ chém) (thuthuatchoi.com)

Demo game: Trò chơi dân gian: "Hướng dẫn chơi Cờ Gánh" - YouTube

Specifications:
In this project, you will make an application for 2 users to play the traditional game Cờ gánh.
- GUI: You can freely design your own UI; however, since this project is to design a program using OOP, there is no need to pay too much focus on the interface. You can also use some image references of the web game in the link provided, or look for any materials you like.
- Design: The application must have these functions:
+ On the main screen:
    - Start: Start the game. For convenience, you do not have to create different difficulties.
    - Exit: Exit the program. Be sure to ask users if they really want to quit the game.
    - Help: Show guide for playing the game.

+ In the game:
    - Gameboard: The chessboard is a flat surface divided into 16 squares with diagonal, horizontal, and vertical lines representing the allowed paths of the pieces. The chessboard has 25 intersection points from the above lines, which are also 25 places to place pieces when playing.
    - For each turn, the application must show clearly whose turn it is. A player will select one of his pieces and a valid point in the chessboard to place the piece. The pieces will change the color if eaten by the opponent.
    - The game ends when there is only one color remaining on the chessboard.
    - For simplicity, you do not have to build a bot to play with human

10. Electronic piano

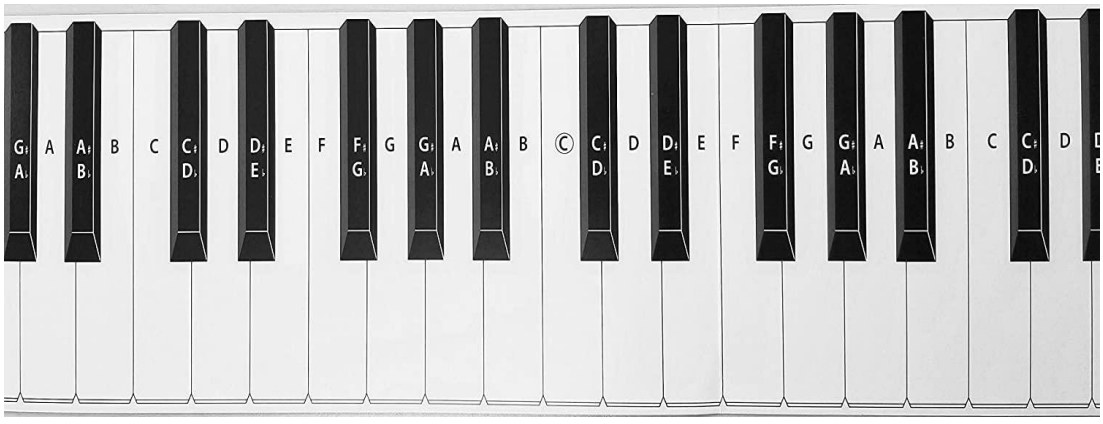
Overview: Piano is a popular musical instrument. In this project, you are asked to implement an application that provides GUI for the user to virtually play an electronic piano.

Basic Knowledge: Data structure, Algorithms, Project Settings, and Music

Specifications:
- GUI: you can freely design your own GUI. However, since the basic aim of the project is to develop an application based on OOP, focusing on the interface is not required
You can refer to these sources to have some idea:
https://www.youtube.com/watch?v=DFnUDyzF1Uk
https://www.youtube.com/watch?v=kvVdeARztt0
- To implement this project, you just need some basic music knowledge. *You don't have to implement a mechanism to play sound, you can directly import libraries from any source*.
You can refer to some libraries:
http://www.jfugue.org/
https://github.com/TitasNandi/Chord-JAVA

- Design:
+ On the main menu: title of the application, piano GUI, help menu, quit
    - User can play the piano by interacting with GUI
    - Help menu shows the basic usage and aim of the program
    - Quit exits the program. Remember to ask for confirmation

+ In the demonstration
    - Keyboard: C (Do), D (Re), E (Mi), F (Fa), G (Sol), A (La), B (Xi), … You can design a keyboard, it doesn't need to be a standard one. However, the more details the keyboard has, the more assessments you can get. Here is an example:



    - A button for increasing/decreasing volume (optional)
    - A record button to record the play (optional)
    - A button for changing music style (optional)

Note: You should use libraries and learn how to add libraries to your project. Again, you are **NOT** asked to implement how the sound will be played.