Cairo University
Faculty of Engineering
Computer Engineering Department
Programming Techniques
Project

Programming Techniques Project

Game

Introduction

In this project we are going to build a simple game application that is a mixture between the Snakes & Ladders and Monopoly games. There are **two modes** in the game: **Design mode** is where the user is able to build the playing grid and **Play mode** is where he plays the game itself. The general idea of the game is four players moving in a grid having snakes, ladders, and Cards. The player goal is to get to the last cell before anyone else. He also takes advantage of the special cells to boost his game and obstruct his opponents. The game ends when a player reaches the last cell.

Your Task: You are required to write a **C++ code** for this Game. Delivering a working project is NOT enough. You must use ***object-oriented programming*** to implement this application and respect the **responsibilities** of each class as specified in the document.

NOTE: The application should be designed so that the types of items and types of operations can be easily extended (*using inheritance*).

Project Schedule

Project Phase	***Deliverables***
Phase 1 [25%]	**Input-Output Classes**
Phase 2 [75%]	**Final Project Delivery**

The project code must be totally yours. The penalty of cheating from any other source is not ONLY taking ZERO in the project grade but also deducting grades from the other class work grades.

Game Description

In the Design mode, the player can add, copy, cut, paste, and delete any game object. He can also save a grid and load a previously saved grid.

In the Play mode, the player is moving in a grid of 99 cells divided into rows and columns. Each cell has a number from 1 to 99. They are divided to 9 rows and 11 columns.

Grid cells may contain:

1. **Ladder**
2. **Snake**
3. **Card** making the cell a **special cell**

Players:

- The Game has four players.
- Each player has:
  - Steps' count (initially step 0). This holds the value of the current cell the player is in.
  - Wallet (initially contains 100 coins).
    - The player can only move if there is at least 1 coin in his wallet. If he has zero or less coins, he must wait till his wallet is recharged with coins.
- Players take turn rolling the dice, one after another.
- After a player rolls the dice, he moves from his n cell to n + t, where t is the result of rolling the dice. For example, if the player gets 4 after rolling the dice and he is in cell 25, he moves to cell 29.
- Every 3 turns, a player dice roll is used to add money to his wallet instead of moving. A player gets 10*t coins (where t is the result of rolling the dice). For example:
  - The game starts and player 0 has 100 coins. He rolls the dice to get 5. He moves to cell 5.
  - In his next turn he gets 2 and moves to cell 7.
  - In his third turn, he rolls the dice and gets 4. Then, 40 coins are added to his wallet, **and he will <u>not</u> move to cell 11.** This happens every three turns, i.e., turn 3, 6, 9, … etc.
- The player's target in the game is to reach the last cell before any opponent reaches it.
- **Winning Player** → Is the first player to reach cell #100.

A Game cell contains players' objects and game objects. It can hold **multiple** player objects, and at most **one** game object.

A Game object is the fixed objects placed in the grid's cells.

You cannot place game objects in the first or last cell.

There are 3 types of Game Objects:

1. Ladders:

- Ladders occupies exactly two cells, the start and end cells.
- If a player moves to the start cell, it will transfer him to the end cell.
- The start cell should be **before** the end cell in the grid.
- For simplicity, the start and end cell should be in the **same** column.
- The start cell cannot be a special cell (a card cell), nor a cell in the last row.
- Two ladders or more cannot overlap.

- The end cell of a ladder cannot be the start cell of another ladder or snake.

2. **Snakes:**
   - Snakes occupy exactly two cells, the start and end cells.
   - If a player moves to the start cell, it will transfer him to the end cell.
   - The start cell should be **after** the end cell in the grid.
   - For simplicity, the start and end cell should be in the **same** column.
   - The start cell cannot be a special cell (a card cell), nor a cell in the first row.
   - Two snakes or more cannot overlap.
   - The end cell of a snake cannot be the start cell of another ladder or snake.

3. **Cards:**
   - A card placed in a cell makes it a special cell.
   - Each card instructs the player stopping at it to do a specific action.
   - Any cell can't have more than one card.
   - A card can be placed only on an empty cell or at the end of a ladder or a snake. This means that a card cannot be placed at the start cell of a ladder or a snake.

There are different types of Cards:

1. **Card1**:
   - Decrements the value of the passing player's wallet by a value specified when creating the grid.
   - Input data in design mode:
     i. Value to decrement
2. **Card2**:
   - Moves the player forward to the start of the next ladder in the grid.
   - If no ladders ahead, do nothing.

3. **Card3 and Card4**:
   - Card3 gives the player an extra dice roll.
   - Card4 prevents the player from rolling the next turn.
   - Both the extra dice roll or the canceled dice roll count normally towards recharging the wallet turn (both increment the count of player turns).

4. **Card5**:
   - Moves the player backward with the same number of steps that he just rolled.
   - If he reaches a ladder, a snake, or a card at the end of moving, take it.

5. **Card6**:
   - Instructs the player to go to a specific cell.
   - If the destination cell contains a ladder, snake, or card, take it.
   - Input data in design mode:
     i. Cell to move to
6. **Card7**:
   - Restarts the game for the first player whose cell is after the current player in the grid.
   - Restarting the game for a player makes him go to cell number 1.

7. **Card8**:
   - This card is a **prison**.

- When a player stops at a Card8 cell, the player should choose either to pay specific amount of coins to go out of the prison, or stay in prison and not playing for 3 turns.
- Input data in design mode:
    i. The amount of coins needed to go out of the prison

8. Cards [from 9 to 11]:

- Gives the player the option to buy this cell and all cells containing a card with the same number. For example, if a player chooses to buy a cell with Card10, he will own all cells having a Card10.
- This cell is considered a **station** for a specific price.
- This specific price is deducted from the player's wallet in case he chooses to buy the cell.
- Whenever another player stops at a cell owned by another player, he has to pay fees to the player who owns the cell.
- The price of the cell and the fees are taken as input during grid design.
- Input data in design mode is only taken if it is the first time to insert a card with the same number. For example, user will insert price and fees of Card10 only once.
- Input data in design mode:
    i. Card price
    ii. Fees to pay by passing players

9. Card12:

- This card moves the ownership of the most expensive station that the current user owns from the current user to the player that has the least amount of coins in his wallet.

Important Note: all the above cards are executed only if the current player stops at it (not passing on it on his way).

Main Toolbar Operations

The application supports 2 modes: **Design Mode** and **Play Mode**. Each mode contains 2 bars: **tool bar** that contains the main operations of the current mode and **status bar** that contains any messages the application will print to the user. The application should support the following operations (actions) in each mode.

[Important Notes]:

- *Each operation in any of the following modes **must** have an **icon** (or more) in the tool bar. The user should click on the operation icon from the tool bar to choose it.*
- **In all the following actions, any dynamically allocated memory MUST be deallocated.**

The main operations supported by the application are the following.

[I] Design Mode:

The purpose of this mode is to design the Grid of the game that you will play later. You specify in this mode the location of the snakes, ladders, and cards. You also specify each cards needed values.
Note: in any of the following actions, if a cell position is needed to be taken from the user, the user **clicks** on it **AFTER** choosing the action (after clicking on the action icon on toolbar not before it).

The operations supported by this mode are:

1- **Add Ladder:** add a ladder to two cells of the user choice in the game grid. This means the user has to click twice: first on the start cell, then on the end cell. The status bar should show the following messages:
   - ☐ "click on the ladder's start cell" (to instruct user to click on the start cell).
   - ☐ "click on the ladder's end cell" (to instruct user to click on the end cell).
   - ☐ If the user chooses an invalid cell, cancel the operation and print an error message, such as:
     - ➢ "end cell cannot be smaller than start cell",
     - ➢ "end cell and start cell are not in the same column",
     - ➢ "two ladders cannot overlap",
     - ➢ "end cell cannot be a start of another ladder or snake", …etc.

2- **Add Snake:** add a snake to two cells of the user choice in the game grid. This means the user has to click twice: first on the start cell, then on the end cell. The status bar should show the following messages:
   - ☐ "click on the snake's start cell" (to instruct user to click on the start cell).
   - ☐ "click on the snake's end cell" (to instruct user to click on the end cell).
   - ☐ If the user clicks an invalid cell, cancel the operation and print an error message.

3- **Add Card:** add a card to the grid. The status bar should show the following messages:
   - ☐ "type card number" (to instruct user to specify card number).
   - ☐ "click on card cell" (to instruct user to click on the required cell).
     - ➢ If the cell has another card or has a start of a ladder or a snake, cancel the operation and print an appropriate error message.
   - ☐ If the card requires values to be inserted as input, ask the user for them and collect the data.

4- **Copy Card:** copy a card with all its data to the clipboard. The status bar should show the following:
   - ☐ "click on the source cell". (to instruct user to click on the source cell).
     - ➢ If the cell does not contain a card, cancel the operation.

5- **Cut Card:** cuts a card with all its data to the clipboard. This is similar to copy card but removes the card from the source cell. The status bar should show the following messages:
   - ☐ "click on the source cell". (to instruct user to click on the source cell).
     - ➢ If the cell does not contain a card, cancel the operation.

6- **Paste Card:** Paste the card in the clip board to the destination cell. You can paste a card from the clipboard multiple times. The status bar should show the following messages:
   - ☐ "click on the destination cell". (to instruct user to click on the destination cell).
     - ➢ If the cell has another card or has a start of a ladder or a snake, do nothing.

7- **Delete Game Object:** delete the game object in the cell that the user chooses.

8- **Save Grid**: saving the information of the designed grid (all the game objects of it) to a file (see "File Format" in phase 2 document). The application must ask the user about the filename to create and save the grid to (overwrite if the file already exists).

9- **Open Grid:** open a saved grid from a file and re-draw it (see "File Format" in phase 2 document).
   - ☐ This operation re-creates the saved game objects and re-draws them.
   - ☐ The application must ask the user about the filename to open and load grid from.
   - ☐ After loading, the user can edit the loaded grid and continue the application normally.
   - ☐ If there is a grid already drawn and the open grid operation is chosen, the application should clear the area and make any needed cleanups then loads the new one.

10- **Switch to Play Mode:** The user can switch to play mode any time even before saving.

11- **Exit:** exiting from the application.
   - ☐ Perform any necessary cleanup (termination housekeeping) before exiting.

[II] Play Mode:

In this mode, four players play the game in the following order:
player 0, player 1, player 2, then player 3.

Current player turn should be displayed in the **top right corner** during his turn. Moreover, for each player, display his current wallet value, and number of turns remaining till he recharges his wallet.

The operations supported by this mode are:

1. **Roll Dice:** rolls the dice and performs necessary actions:
   - Moves the player to the correct cell
     i. If he reaches a start of a ladder,
        a) Display "you have reached a ladder. Click to continue"
        b) Wait for a user click
        c) Move to the top of the ladder
     ii. If he reaches a start of snake,
        a) Display "you have reached a snake. Click to continue"
        b) Wait for a user click
        c) Move to the bottom of the snake
     iii. If he reaches a card,
        a) Display a message on status bar describing the job of the card
        b) Wait for a user click
        c) Ask the user to enter any inputs needed for executing this card. For example, for cards from 9 to 11, if this station is not bought, ask him if he want to buy it or not.

2. **Input Dice Value:** Instead of generating a random value for the dice, ask the user to insert a value and then do the necessary actions similar to rolling a dice.
   - display "please enter a dice value between 1-6"

3. **New Game:** Restarts players' positions, wallets, and turn returns to player 0.

4. **Switch to Design Mode**: At any time, user can switch back to the other mode.
   - The current grid is opened in the design mode.
   - Don't forget to make any needed cleanups.

5. **Exit:** exiting from the application.
   - Perform any necessary cleanup (termination housekeeping) before exiting.

Main Classes of phase 1

You are given a ***code framework*** where we have ***partially*** written code of some of the project classes. For the graphical user interface (GUI), we have integrated an open-source ***graphics library*** that you will use to easily handle GUI (e.g. drawing grid on the screen and reading the coordinates of mouse clicks …etc.).

You should **stick to** the **given design** (i.e. hierarchy of classes and the specified job of each class) and complete the given framework as described in phase 1 deliverables section below.

Input Class:
ALL user inputs must come through this class. If any other class needs to read any input, it must call a member function of the input class. You should add suitable member functions for different types of inputs.

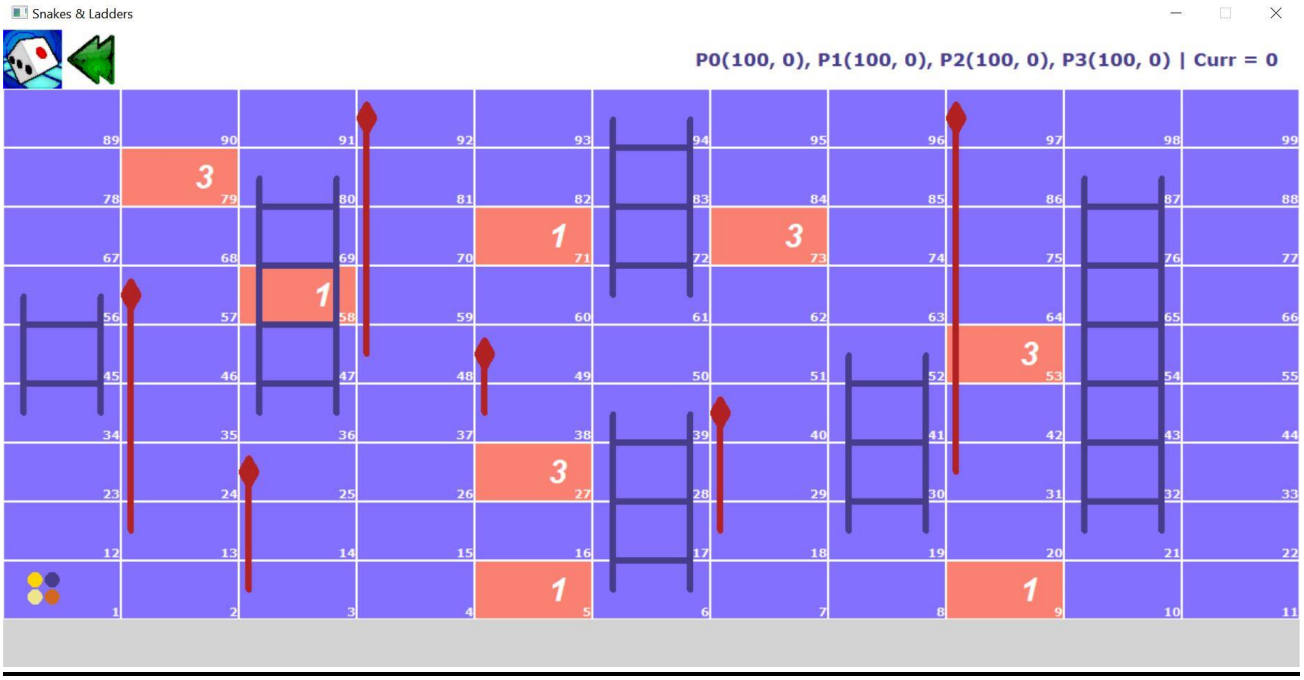
Output Class:
This class is responsible for **ALL** GUI outputs. It is responsible for toolbar and status bar creation, grid and game objects drawing, and for messages printing to the user. If any other class needs to make any output, it must call a member function of the output class. You should add suitable member functions for different types of outputs.

Notes: - No input or output is done through the console. All must be done through the **GUI window**.
- Input and Output classes are the **ONLY** classes that have **access to GUI library**.

Cell position Class:
This class represents the cell position in the grid by having two data members called vCell and hCell. This class does NOT deal with real coordinates, it deals with the vCell, hCell and cellNum instead.

Program Interface

The following is an example of the program interface



Project Phases

A partially-implemented code framework for Phase 1 and for Phase 2 will be given to you.
For **fast navigation** in the given code in **Visual Studio**, you may need the following:

- **F12 (go to definition):** to go to definition of functions (code body), variables, …etc.
- **"Ctrl" then "Minus":** to return to the previous location of the cursor.

1- Phase 1 (Input / Output/ CellPosition Classes)

In this phase, you will implement the **Input**, the **Output** and the **CellPosition** classes. Any expected user interaction (input/output) that will be needed by phase 2, should be implemented in phase 1.

You are given a code for phase 1 (separate from the code of the whole project) that contains phase1 classes partially implemented. Each team should complete the following:

1- *Output Class*:

- **Draw the Full Tool Bars**:
  Output class should create 2 FULL tool bars: one for the **Design mode** and one for the **Play mode**. Each contains icons for every action in this mode.

- **Implement the Following Functions:**
  - **GetCellStartX(cellpos) & GetCellStartY(cellpos):**
    Gets the X,Y coordinates of the upper left corner of the passed cellpos**.**
  - **DrawCardNumber(cellpos, cardNum):**
    Draws the passed card "number" in the passed cellpos.
  - **PrintPlayersInfo(info):**
    Prints players' info on the right-side of the toolbar.
  - **DrawCell(cellpos, cardNum):**
    Draws "Cell Rectangle" with "Cell Number" and "Card Number" (if any), cardNum -1 means no Card the cell rectangle's color depends on having a card or not.
  - **DrawPlayer(cellpos, playerNum, playerColor):**
    Draws "Player Circle" filled with the passed color, "playerNum" parameter is used for locating the circle representing the player.
  - **DrawLadder(fromCellPos, toCellPos):**
    Draws Ladder from start cell to end cell (start < end). Ladder is drawn as follows:
    - Two vertical lines
    - Horizontal lines as many as the cell borders between the start and end.
  - **DrawSnake(fromCellPos, toCellPos):**
    Draws Snake from start cell to end cell (start > end). Snake is drawn as follows:
    - Snake's body is drawn as line
    - Snake's head is drawn as Polygon (with Diamond Shape)

2- Input Class:

- **Implement the Following Functions:**
  - **GetUserAction (…):**
    It is where the input class should detect all possible actions of any of the 2 modes according to the coordinates clicked by the user. It is partially implemented.
  - **GetInteger():**
    It takes an integer as input from the user.
  - **GetCellClicked ( ) :**
    It takes a mouse click from the user and returns the cell position on which the user clicks. If the click is NOT on a cell, cell (-1,-1) is returned.

3- CellPosition Class:

- **Implement the Following Functions:**
  - **SetVCell(v), SetHCell(h):**
    Setters for vCell and hCell.
  - **IsValidCell():**
    Checks if the current cell position (vCell and hCell) both are valid then return true. Otherwise, return false
  - **GetCellNumFromPosition(cellPosition):**
    Calculates the cellNum of the passed "cellPosition"
  - **GetCellPositionFromNum(cellNum):**
    Returns the corresponding CellPosition (vCell, hCell) of the passed cellNum
  - **AddCellNum(addedNum**):
    Adds the passed "addedNum" to the "cellNum" of the current cell position and updates the data members (vCell and hCell) accordingly. For example, if cellNum = 50 and the passed num = 6, this will make cellNum = 56 which updates the data members: vCell = 3 and hCell = 0.
    (assuming NumVerticalCells = 9 and NumHorizontalCells = 11)

4- *DEFS.h and UI_Info.h Files*: as you implement the above classes, you will find some parts you need to add in DEFS.h and UI_Info.h files.

5- *Test Code:* this is <u>not</u> part of the above classes; it is just a test code (the **main**).
- Complete the code given in **TestCode.cpp** file to test both Input and Output classes.
- Do NOT re-write the main function from scratch, just complete the required parts.

Notes:
- In Phase 1 code, follow any ***///*TODO** comments to know what should be done in each .h or .cpp file.
- You must follow the example interface mentioned in section “Program Interface” exactly. All drawn objects should look exactly the same as the example.
- Nearly all coordinate calculations are done for you. You just need to use it to draw every object correctly according to its specifications.

Notes on The Project Graphics Library*:*

- The origin of the library’s window **(0, 0)** is at the **upper left** corner of the window.
- The direction of **increasing** the **x coordinate** is to the **right**.
- The direction of **increasing** the **y coordinate** is **down**.
- The only image extension that the library accepts is “**jpg**”.
- You can use any alternative graphics library if you want, but with the condition of keeping the same interface (function prototypes) and any classes of phase 1 or 2 (same interface but the function body will be using your new library). However, we as TAs do not guarantee the support if you have any problem with your new library. It is your responsibility to deal with it.

Phase 1 Deliverables*:*

The online submission of phase contains IDs.txt (team number, member names, IDs, email) and phase 1 code that has Input, Output, and CellPosition classes, DEFS.h, UI_Info.h and test program completed. **No modifications are allowed after the online submission deadline.**