CS 101 - Algorithms & Programming I

Fall 2024 - Lab 8

Due: Week of December 2, 2024

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to define and use custom classes. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice. Specifically for this lab, you are to organize your data and methods, working on them as classes.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab8`.

In this lab, you are to create Java classes/files (under `labs/lab8` folder) as described below. You are expected to submit 4 files for your initial solution (`SkyBattleGame, Knight, BattleColumn, and BattleManager`). For the revision, you may submit up to 4 additional files, adding the `"Rev"` suffix to each revision file (e.g., `KnightRev`). **Please submit the files without compressing them, and ensure that no other or previous lab solutions are included.**

Output/gameplay is shared as **animated gifs through Google Drive.** When possible, outputs of sample runs are shown in **brown** whereas the user inputs are shown in **blue**.

Your code must match the names and types of variables and interface of the methods specified.

1. Sky Battle Game

In the infinite sky above, the **Knight** is tasked with one simple mission: survive. The once calm skies are now full with **Dark Knights**, dangerous souls that endlessly pursue the Knight. The rules are simple: the Knight must avoid these dark souls as long as possible.

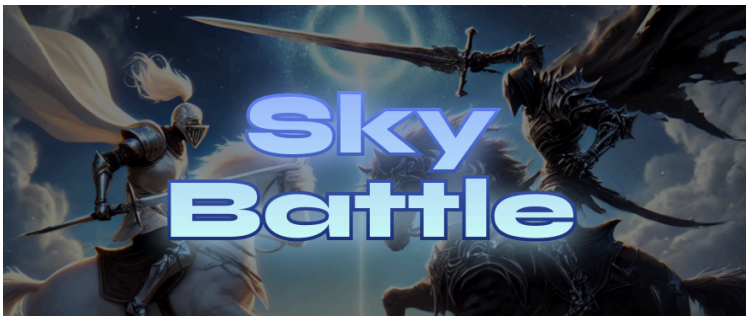


Image Credit: GPT Vision

In this exercise, you will use object-oriented programming concepts to implement the **Sky Battle Game**. All class structures are provided, along with some partial implementations. You are expected to complete the remaining parts of the code **(marked by three dots)**, and you may add **helper methods if needed**. Detailed descriptions of each class are provided below.

`SkyBattleGame` Class:

This class implements the game logic including the main game loop for the sky battle where the player controls a knight to avoid dark knights. The game initializes the environment, processes user inputs, and keeps track of the score as the player survives.

Static Data Members:

- `score`: Tracks the player's score based on how long the Knight survives.
- `battleManager`: Manages the gameplay mechanics, including the Knight's movements and interactions with Dark Knights. (will be implemented by you)
- `scanner`: Handles user input for controlling the Knight's movements.
- `COLS`: The number of columns of the game grid
- `ROWS`: The number of rows of the game grid
- `INITIAL_HEALTH`: Initial health of the Knight
- `MAX_DARK_KNIGHT_SIZE`: Maximum Dark Knight size
- `MOVE_UP`: The user input character to move the Knight up
- `MOVE_DOWN`: The user input character to move the Knight down
- `STAY_STILL`: The user input character to make the Knight stay still
- `MOVE_UP_DIST`: The distance traveled by the Knight on moving up
- `MOVE_DOWN_DIST`: The distance traveled by the Knight on moving down
- `STAY_STILL_DIST`: The distance traveled by the Knight in staying still
- `EMPTY_COL_SYMBOL`: The symbol of the empty column (use this: ' ')
- `DARK_KNIGHT_SYMBOL`: The symbol of Dark Knights (use this: '♘'[1])
- `KNIGHT_SYMBOL`: The symbol of the Knight (use this: '♞')

Methods:

- Main method: Starts the game by calling `initializeGame()`, then enters the game loop through `startGame()`, and concludes with `endGame()` when the Knight's health reaches zero.
- Service Methods:
  - `initializeGame()`: Initializes the `battleManager` to set up the game's mechanics and starts the `scanner` to read user input.
  - `playGame()`: Runs the main game loop, which continues until the Knight's health is depleted. It repeatedly renders the game and handles user input.
  - `endGame()`: Prints the final score and closes the `scanner` when the game is over.
  - `renderGame()`: Displays the current state of the game by rendering the battleground and game information (Knight's stats and points).
  - `displayBattleground(Knight knight, ArrayList<BattleColumn> columns)`: Renders the grid of battle columns and displays the Knight's position within the grid.
  - `displayGameInformation(Knight knight)`: Prints the current game points and the Knight's health.
  - `handleUserInformation()`: Reads the player's input for knight movements (up, down, or stay still) and directs it to `battleManager` to execute.
  - `isKnightPositionedAt(Knight knight, int x, int y)`: Checks whether or not the Knight is located at the given coordinates (x, y) in the grid.

---

[1] Notice that since Consoles, where output is displayed, normally have black background, colors will be reversed :) If on white background, simply reverse them.

`Knight` Class:

This class represents the Knight in the game that has a position, health points, and a symbol. The Knight's health is used to determine whether they are still "alive" in the game. The class provides methods for managing the Knight's state and rendering its current information.

Instance Data Members:

- `y`: The y-coordinate of the Knight, representing its vertical position on the grid.
- `health`: The current health points of the Knight. The Knight is considered "alive" as long as health is greater than zero.
- `symbol`: A character representing the Knight visually.

Methods:

- Constructor:
  - `Knight(int y, int health, char symbol)`: Initializes the Knight with a specified y position, health, and a visual symbol. It sets the health using the `setHealth()` method to ensure the value is non-negative.
- Service Methods:
  - `isAlive()`: Returns true if the Knight's health is greater than zero, meaning the Knight is still alive; otherwise, returns false.
  - `toString()`: Returns a string representation of the Knight, including its symbol and a visual representation of its health using the '*' symbol. The number of asterisks corresponds to the Knight's health.
- Accessor Methods:
  - `getHealth()`: Returns the current health of the Knight.
  - `getY()`: Returns the current y-coordinate of the Knight.
  - `getSymbol()`: Returns the symbol representing the Knight.
- Mutator Methods:
  - `setY()`: Updates the Knight's y-coordinate with the provided value.
  - `setHealth(int health)`: Sets the Knight's health to the provided value, but ensures the health is never set to a value below 0 using `Math.max(health, 0)`.

`BattleColumn` Class:

This class represents a single column of the game grid, which may contain empty spaces or Dark Knights. This class manages the creation of the column and the placement of Dark Knights based on the game configuration.

Instance Data Members:

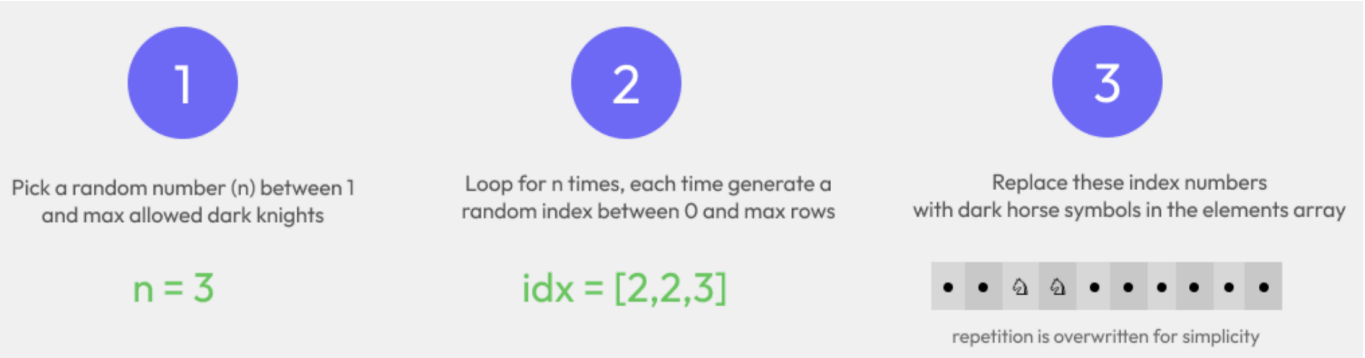
- `elements`: An array of char representing the contents of the column. Each element in the array corresponds to a column in the grid and can either be an empty space or a Dark Knight.

Methods:

- Constructor:
  - `BattleColumn(boolean hasDarkKnights)`: Initializes a new column with a size corresponding to the number of rows in the grid (`SkyBattleGame.ROWS`). It calls the `generate()` method to populate the column with either empty spaces or Dark Knights depending on the `hasDarkKnights` parameter.
- Service Methods:
  - `generate(boolean hasDarkKnights)`: Populates the column by first initializing all positions as empty (`initEmptyColumn()`), and then randomly placing Dark Knights if `hasDarkKnights` is true. It calls `addDarkKnights()` to handle the placement of Dark Knights.
  - `initEmptyColumn(boolean hasEnemies)`: Fills the column's elements array with the empty column symbol (`SkyBattleGame.EMPTY_COL_SYMBOL`), ensuring that all positions start as empty.
  - `addDarkKnights(Random random)`: Adds a random number of Dark Knights (**between 1 and the configured maximum**, `SkyBattleGame.MAX_DARK_KNIGHT_SIZE`) to random positions in the column using the Random object. Each selected position is replaced by the Dark Knight symbol (`SkyBattleGame.DARK_KNIGHT_SYMBOL`). Notice that if the same location is chosen multiple times, fewer dark nights will be created.
- Accessor Methods:
  - `getElements()`: Returns the array of char representing the contents of the column. Each element in the array corresponds to a row and can either be empty or contain a Dark Knight.

Dark Knight Generation Logic

While generating Dark Knight's, the following logic should be implemented for `addDarkKnights` method: (Please note that, with any duplicate random indexes simply overwrite the previous value for simplicity)



`BattleManager` Class:

This class is responsible for managing the Knight's movements, grid interactions, and the game state related to the columns. It handles player movement, checks for collisions with Dark Knights, and updates the grid by shifting columns.

Instance Data Members:

- `knight`: An instance of the Knight class representing the player-controlled Knight.
- `columns`: An ArrayList of BattleColumn objects representing the columns in the game grid, where each column may contain empty spaces or Dark Knights.

Methods:

- Constructor:
  - `BattleManager()`: Initializes the columns list with a number of BattleColumn instances based on `SkyBattleGame.COLS`. The Knight is initialized with `y=0` position, initial health (`SkyBattleGame.INITIAL_HEALTH`), and a knight symbol (`SkyBattleGame.KNIGHT_SYMBOL`).
- Service Methods:
  - `handleMovement(String direction)`: Takes the player's input (direction) and adjusts the Knight's y position accordingly. It checks for valid movement directions (up, down, stay still), ensures the knight does not go out of bounds. If the move is valid, the `shiftGrid` method is called first to shift the grid to the left, followed by the `move` method, which updates the knight's position and checks for collisions. Shifting the grid before moving ensures accurate collision detection based on the knight's new position. The method returns `true` if the move is valid and successful, or `false` if the move is invalid or out of bounds.
  - `move(int yDist)`: Updates the Knight's y position by the provided `yDist` and checks the position for Dark Knights. If the Knight collides with a Dark Knight, its health is reduced by 1, and an attack message is displayed using `displayAttackMessage()`. Otherwise, the player's score (`SkyBattleGame.score`) increases by 1.
  - `shiftGrid()`: Shifts the grid by moving each column to the left and generating a new column at the rightmost position.
  - `displayAttackMessage()`: Displays a message to the player when the Knight is attacked by a Dark Knight.
- Accessor Methods:
  - `getKnight()`: Returns the current instance of the knight.
  - `getColumns()`: Returns the ArrayList of columns representing the current game grid.

Gameplay (Animated GIF):

You can view the gameplay here: Gameplay.gif
Additionally, you can access sample output as text file here: sample_output.txt