CS 101 - Algorithms & Programming I

Fall 2025 - Lab 7

Due: Week of November 24, 2025

*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 `lab7`.

In this lab, you are to create Java classes/files (under `labs/lab7` folder) as described below. You are expected to submit 4 files for your initial solution (`TowerDefenseGame, Tower, EnemyWave, and GameManager`). For the revision, you may submit up to 4 additional files, adding the `"Rev"` suffix to each revision file (e.g., `TowerRev`). **Please submit the files without compressing them, and ensure that no other or previous lab solutions are included.**

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. Tower Defense Game

In the ancient lands, a lone Tower stands as the final line of defense. Waves of ruthless enemies march endlessly toward its walls. The rules are simple: defend the Tower at all costs. With each passing wave, the enemies grow stronger and more relentless. The fate of the realm rests on your ability to build, position, and command towers wisely. How long can you withstand the onslaught before the defenses crumble?

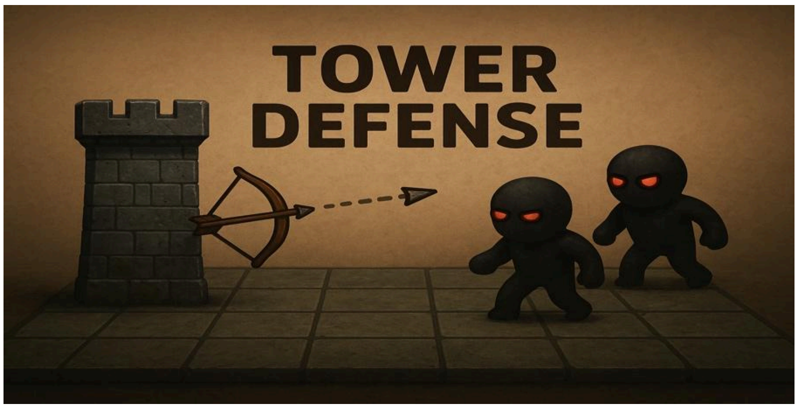


Image Credit: ChatGPT

In this exercise, you will use object-oriented programming concepts to implement the **Tower Defense Game**. All class structures are provided, along with some partial implementations. You are expected to complete the remaining parts of the code, and you may add helper methods if needed. Detailed descriptions of each class are provided below, and to view gameplay videos and images, visit this link.

`TowerDefenseGame` Class:

This class handles the overall game flow and user interactions. It runs the main game loop, renders the game map and tower/enemy information, handles player input for attacking enemies, and displays the end-of-game messages. Also stores static variables shared across the project and focuses on controlling the sequence of the game, not on managing the internal state game state and core logic, which is handled by the `GameManager` class.

Static Variables:
- `ROWS:` Number of horizontal rows in the game grid.
- `COLUMNS:` Number of vertical columns in the game grid.
- `INITIAL_HEALTH:` Starting health value of the tower.
- `ENEMY_WAVES:` Total number of enemy waves before the game ends.
- `MAX_ENEMY_SPAWN_RATE:` Maximum number of enemies in a single wave.
- `VICTORY_MESSAGE:` Message displayed if the tower survives all waves.
- `DEFEAT_MESSAGE:` Message displayed if the tower's health drops to zero.
- `TOWER_SYMBOL:` Symbol used to display the tower on the grid. ("🏰")
- `ENEMY_SYMBOL:` Symbol used to display enemies on the grid. ("👾")
- `EMPTY_SYMBOL:` Symbol used for empty grid positions. ("⬜")
- `gameManager:` Instance of `GameManager` controlling the main game logic.
- `scanner:` Instance of `Scanner` used to read player input.
- `random:` Instance of `Random` used for enemy generation.

Methods:
- `main(String[] args):` Entry point of the program. Initializes game variables, runs the main game loop, and handles the end-of-game display.
- `initializeVariables():` Sets up the `Random`, `Scanner`, and `GameManager` instances needed for the game to run.
- `playGame():` Main game loop. Continues running until the game is over. In each iteration, it renders the game graphics, handles the player's attack, applies tower damage, and spawns new enemy waves if any remain.
- `handleGameEnding():` Displays the final state of the game. Prints victory or defeat messages based on whether the tower is still standing and shows the tower's final score. Closes the input scanner.
- `renderGraphics():` Draws the current game state to the console. Calls `renderGameMap()` to display the battlefield and `renderGameInformation()` to display tower stats and remaining waves.
- `renderGameMap():` Prints the grid showing the tower, enemy waves, and empty spaces. Shows row and column indices for reference.
- `renderGameInformation():` Displays tower information, including its symbol, health, current score, and remaining enemy waves.
- `handleHit():` Prompts the player to enter a column and row index to target an enemy. Calls methods to process the attack and updates the tower's score.
- `getValidInput(String prompt, int min, int max):` Reads integer input from the player and ensures it is within the specified range. Repeats until a valid number is entered.

`Tower` Class:

This class models the Tower in the game, which has a health value, a score, and a visual symbol. It provides methods to adjust health and score, check if the Tower is still standing, and represent the Tower's current status as a string. The class ensures the Tower's health never drops below zero and keeps track of points earned by hitting enemies

Instance Variables:

- `health:` Stores the current health of the tower. If health drops to 0, the tower is considered destroyed.
- `score`: Keeps track of the points earned by the tower when enemies are hit.
- `symbol:` The visual representation of the tower on the game grid.

Constructor & Methods:

- `Tower(int health, int score, String symbol):` Initializes a new tower with the given health, starting score, and symbol. Calls `setHealth()` to ensure the health is not negative.
- `isStanding():` Returns `true` if the tower's health is greater than 0, otherwise `false`. Determines if the tower is still active in the game.
- `takeDamage(int damage):` Decreases the tower's health by the given damage value. Health is never allowed to go below 0.
- `incrementScore(int score):` Adds the specified value to the tower's current score. Used when the tower successfully hits enemies.
- `toString():` Returns a string summarizing the tower's current symbol, health, and score. Useful for displaying tower information in the console.
- `getSymbol():` Returns the tower's symbol.
- `getScore():` Returns the current score of the tower.
- `setScore(int score):` Updates the tower's score to a new value.
- `getHealth():` Returns the current health of the tower.
- `setHealth():` Updates the tower's health. Ensures the health value is never negative.

`EnemyWave` Class:

This class represents a single column in the game grid that can contain enemies or empty spaces. It handles the creation of the column, places enemies according to the game rules, and provides methods to update or remove enemies during gameplay.

Instance Variables:

- `enemyWave:` An array of strings representing a single column in the game grid. Each element can either be an enemy symbol (`ENEMY_SYMBOL`) or an empty symbol (`EMPTY_SYMBOL`).

Constructor & Methods:

- `EnemyWave(int numberOfEnemies):` Initializes the column with a specified number of enemies. Enemies are first placed at the top positions in the array and then shuffled randomly if any enemies exist in the wave.
- `initializeWave(int numberOfEnemies):` Fills the `enemyWave` array with enemies in the first positions up to the number of enemies, and fills the rest with empty symbols. Ensures the number of enemies does not exceed the number of rows.
- `shuffleArray():` Randomly shuffles the positions of enemies and empty spaces using the **Fisher–Yates algorithm** described below, making the enemy distribution unpredictable.
- `hitEnemy(int index):` Attempts to hit the enemy at the specified row index in the column. If an enemy is present, it is removed (replaced with an empty symbol) and returns `1`. If the position is empty, return `0`.
- `getNumberOfEnemies():` Counts and returns the total number of enemies currently present in the column.
- `getEnemyWave():` Returns the array representing the column of enemies and empty spaces.

Fisher-Yates Algorithm[1]:

1. Start from the last element of the array.
2. Pick a random index from the range 0 to the current index.
3. Swap the element at the current index with the element at the random index.
4. Move one step backward and repeat until you reach the beginning.

[1] Fisher, R. A., & Yates, F. (1938). *Statistical tables for biological, agricultural and medical research* (3rd ed.). London: Oliver and Boyd.

`GameManager` Class:

This class manages the internal game state and core logic. It keeps track of the tower, enemy waves, and remaining wave information, handles creating and updating enemy waves, processes attacks on enemies, and calculates damage to the tower. Also provides methods to query the game state, focuses on game rules and state management, independent of rendering or player input.

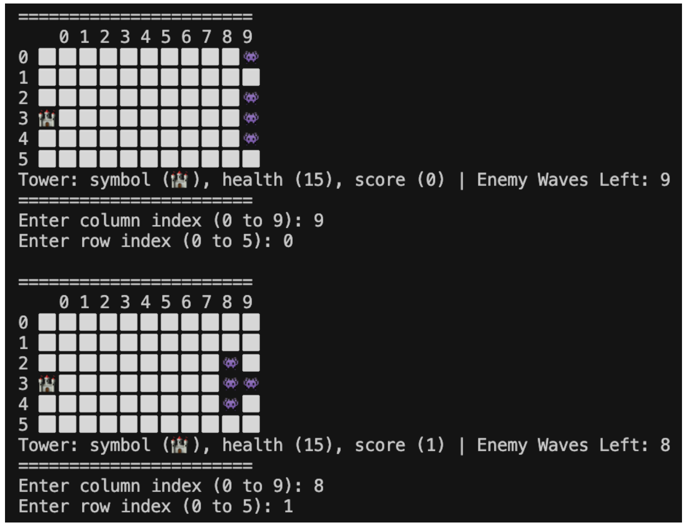
Instance Variables:

- `tower:` Holds the `Tower` object for the game.
- `enemyWaves:` An `ArrayList` that stores all the enemy waves (columns) on the game grid. Each `EnemyWave` represents a column of enemies and empty spaces.
- `enemyWavesLeft:` Tracks how many enemy waves are still left to spawn in the game.

Constructor & Methods:

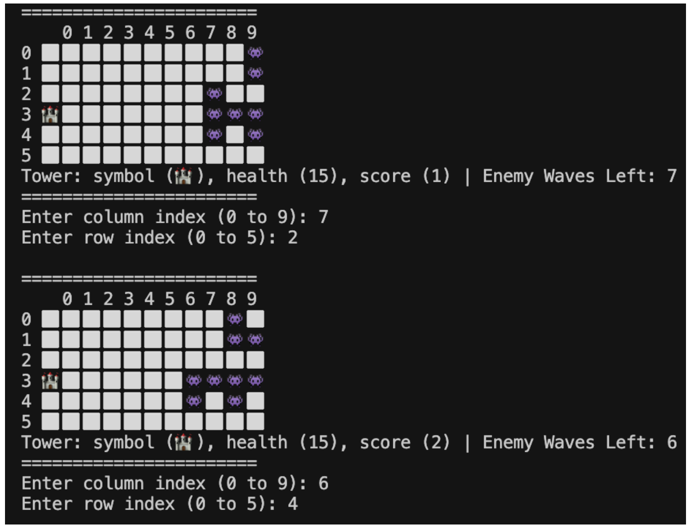
- `GameManager():` Constructor. Initializes the tower with initial health and score, creates an empty grid of enemy waves, and adds the first enemy wave.
- `generateEmptyMap():` Fills the grid with empty enemy waves (no enemies). Ensures the map has `COLUMNS` number of columns initially.
- `generateNextEnemyWave(boolean hasEnemies):` Creates a new `EnemyWave`. If `hasEnemies` is true, spawns a random number of enemies up to `MAX_ENEMY_SPAWN_RATE.`
- `getNumberOfEnemies(int index):` Returns the number of enemies still alive in the enemy wave at the specified column index. Returns `0` if the index is invalid.
- `addNextEnemyWave():` Removes the oldest enemy wave (front column) and adds a new enemy wave at the end. Decreases `enemyWavesLeft` by 1.
- `hitEnemy(int columnIndex, int rowIndex):` Attempts to hit an enemy at the specified column and row. Returns 1 if an enemy is hit and removed, otherwise returns 0.
- `getTower():` Returns the `Tower` object used in the game.
- `hasEnemyWavesLeft():` Returns true if there are still enemy waves left to spawn, false otherwise.
- `isGameOver():` Checks if the game has ended. The game ends if the tower is destroyed or all enemy waves have been spawned.
- `handleTowerDamage():` Reduces the tower's health based on the number of enemies in the front column (column 0). Each enemy in that column decreases the tower's health by 1.
- `getEnemyWaves():` Returns the `enemyWaves` variable.
- `getEnemyWavesLeft():` Returns the number of remaining enemy waves that can still be spawned. Always returns a non-negative number.

Gameplay:



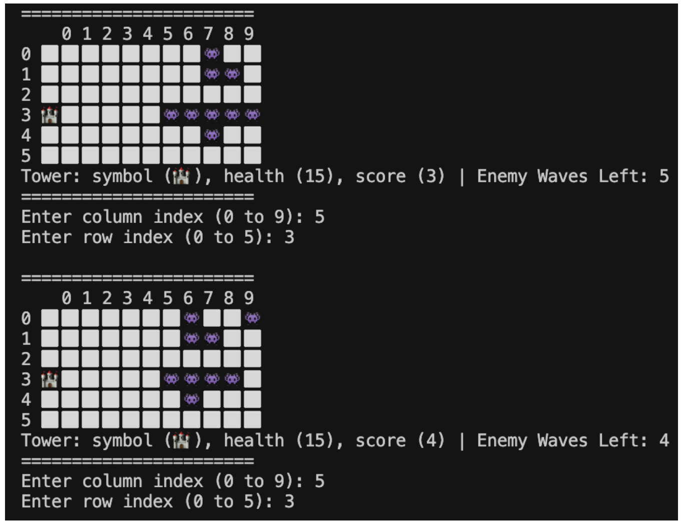
```
=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
1 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
3 🏰⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
4 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (0) | Enemy Waves Left: 9
=======================
Enter column index (0 to 9): 9
Enter row index (0 to 5): 0

=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
1 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾⬜
3 🏰⬜⬜⬜⬜⬜⬜⬜⬜👾👾
4 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (1) | Enemy Waves Left: 8
=======================
Enter column index (0 to 9): 8
Enter row index (0 to 5): 1
```



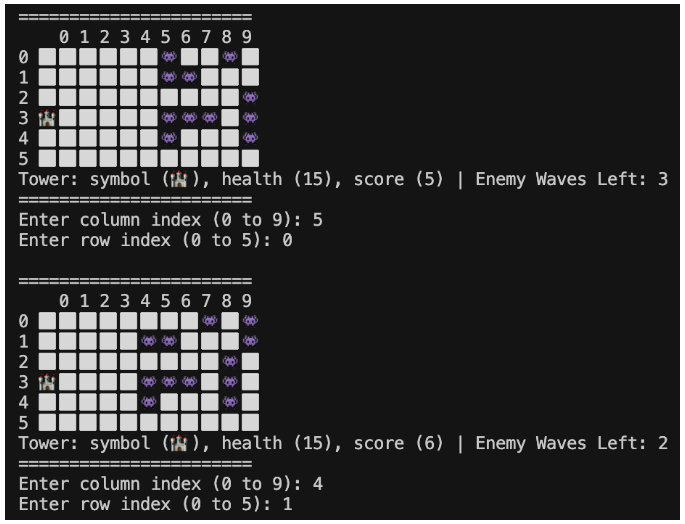
```
=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
1 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
2 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜⬜
3 🏰⬜⬜⬜⬜⬜⬜⬜👾👾👾
4 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜👾
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (1) | Enemy Waves Left: 7
=======================
Enter column index (0 to 9): 7
Enter row index (0 to 5): 2

=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾⬜
1 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾👾
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
3 🏰⬜⬜⬜⬜⬜⬜👾👾👾👾
4 ⬜⬜⬜⬜⬜⬜⬜👾⬜👾⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (2) | Enemy Waves Left: 6
=======================
Enter column index (0 to 9): 6
Enter row index (0 to 5): 4
```



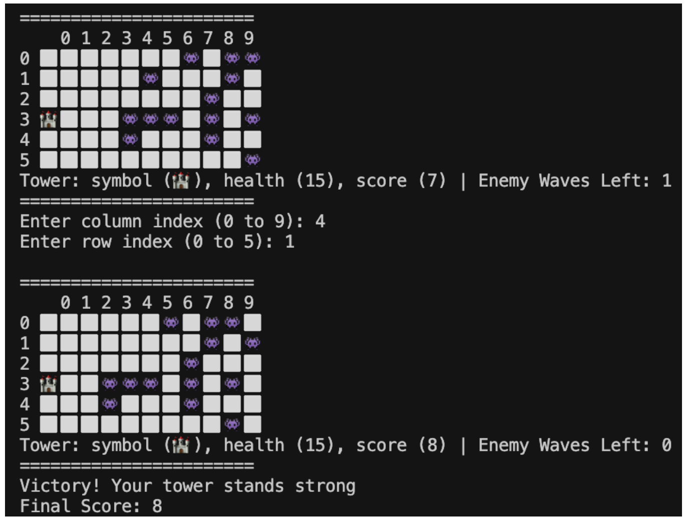
```
=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜⬜
1 ⬜⬜⬜⬜⬜⬜⬜⬜👾👾⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
3 🏰⬜⬜⬜⬜⬜👾👾👾👾👾
4 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (3) | Enemy Waves Left: 5
=======================
Enter column index (0 to 9): 5
Enter row index (0 to 5): 3

=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜👾⬜⬜👾
1 ⬜⬜⬜⬜⬜⬜⬜👾👾⬜⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
3 🏰⬜⬜⬜⬜⬜👾👾👾👾⬜
4 ⬜⬜⬜⬜⬜⬜⬜👾⬜⬜⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (4) | Enemy Waves Left: 4
=======================
Enter column index (0 to 9): 5
Enter row index (0 to 5): 3
```



```
=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜👾⬜⬜👾⬜
1 ⬜⬜⬜⬜⬜⬜👾👾⬜⬜⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
3 🏰⬜⬜⬜⬜⬜👾👾👾⬜👾
4 ⬜⬜⬜⬜⬜⬜👾⬜⬜⬜👾
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (5) | Enemy Waves Left: 3
=======================
Enter column index (0 to 9): 5
Enter row index (0 to 5): 0

=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜👾
1 ⬜⬜⬜⬜⬜👾👾⬜⬜⬜👾
2 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾⬜
3 🏰⬜⬜⬜⬜👾👾👾⬜👾⬜
4 ⬜⬜⬜⬜⬜👾⬜⬜⬜👾⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Tower: symbol (🏰), health (15), score (6) | Enemy Waves Left: 2
=======================
Enter column index (0 to 9): 4
Enter row index (0 to 5): 1
```



```
=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜⬜👾⬜👾👾
1 ⬜⬜⬜⬜⬜👾⬜⬜⬜👾⬜
2 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜⬜
3 🏰⬜⬜⬜👾👾👾⬜👾⬜👾
4 ⬜⬜⬜⬜👾⬜⬜⬜👾⬜⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜👾
Tower: symbol (🏰), health (15), score (7) | Enemy Waves Left: 1
=======================
Enter column index (0 to 9): 4
Enter row index (0 to 5): 1

=======================
    0 1 2 3 4 5 6 7 8 9
0 ⬜⬜⬜⬜⬜⬜👾⬜👾👾⬜
1 ⬜⬜⬜⬜⬜⬜⬜⬜👾⬜👾
2 ⬜⬜⬜⬜⬜⬜⬜👾⬜⬜⬜
3 🏰⬜⬜👾👾👾⬜👾⬜👾⬜
4 ⬜⬜⬜👾⬜⬜⬜👾⬜⬜⬜
5 ⬜⬜⬜⬜⬜⬜⬜⬜⬜👾⬜
Tower: symbol (🏰), health (15), score (8) | Enemy Waves Left: 0
=======================
Victory! Your tower stands strong
Final Score: 8
```