CS 101 - Algorithms & Programming I

Fall 2025 - Lab 8

Due: Week of December 8, 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 understand and apply the concepts of classes, references, and inheritance through an interactive game. Before starting the implementation, you are encouraged to think about how the game elements interact and organize them on paper as classes with clearly defined responsibilities. In this lab, you will apply the principles of object-oriented programming to structure the **Space Invaders** game using multiple classes that represent different parts of the gameplay.

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 8 files for your initial solution (`Main`, `GameEngine`, `GameField`, `Entity`, `Enemy`, `SpaceShip`, `Bullet`, and `Drawer`). For the revision, you may submit additional files, adding the `"Rev"` suffix to each revision file (e.g., `GameFieldRev`). **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 the interface of the methods specified.

1. Space Invaders Game

In the dark void of space, the player commands a lone spaceship on a mission to defend against waves of invading **enemies.** Each round, monsters descend from the sky, while the player must move and shoot to survive as long as possible. The goal is simple: destroy the enemies before they reach you.

Image Credit: GPT-Vision

In this exercise, you will use object-oriented programming concepts to implement the **Space Invaders 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 and their interactions are provided below.

`Main` Class:

This class serves as the entry point for the **Space Invaders** game. It creates an instance of `GameEngine` and starts the main game loop that manages user interaction and overall gameplay.

Methods:

- `main`: Launches the Space Invaders game by creating a new `GameEngine` object and calling its `run()` method to start the game loop.

`GameEngine` Class:

This class implements the main logic of the **Space Invaders** game. It manages the game loop, player input, entity updates, and interactions between the spaceship, enemies, and bullets. The class also defines key constants such as grid size, entity symbols, and spaceship health.

Static Data Members:

- `ENEMY_SYMBOL`: Character symbol used to represent enemies ('M').
- `SPACE_SHIP_SYMBOL`: Character symbol representing the player's spaceship ('^').
- `BULLET_SYMBOL`: Character symbol used for bullets ('|').
- `ENEMY_PER_ROW`: Number of enemies spawned per turn.
- `SPACE_SHIP_HEALTH`: The initial health of the player's spaceship.
- `WIDTH`: The width (in columns) of the game grid.
- `HEIGHT`: The height (in rows) of the game grid.

Instance Data Members:

- `field`: A `GameField` object managing the interactions between the spaceship, enemies, and bullets.
- `in`: A `Scanner` used to capture user input from the keyboard.
- `spaceShip`: The player-controlled spaceship object.
- `enemies`: A dynamic list holding all active enemies on the screen.
- `bullets`: A dynamic list holding all active bullets fired by the spaceship.

Methods:

- **`Constructor:`**
  - `GameEngine()`:
    Initializes the game by creating a `SpaceShip` object at the bottom of the grid, empty ArrayLists for enemies and bullets, and a GameField object that connects and manages them.
- **`Service Methods:`**
  - `run()`: Starts the main game loop. Displays the controls, repeatedly renders the game, processes user input, and ends the game when the spaceship's health reaches zero or the player quits.
  - `update(char command)`: Executes one complete frame of the game.

        - Processes user commands (A to move left, D to move right, S to shoot).
        - Updates the positions of bullets and enemies.
        - Checks for bullet and spaceship collisions through `GameField`.
        - Spawns new enemies each turn.
        - Removes any bullets or enemies that move outside the game boundaries.

`Drawer` Class:

This class handles all visual output for the **Space Invaders** game. It displays the game board, the player's spaceship, enemies, bullets, and gameplay statistics such as score and health. The class contains only static methods and does not store any game state; it simply receives data from `GameEngine` and `GameField` and prints the current frame each turn.

Static Methods:

- `printControls()`: Prints the control instructions for the game. This method is called once at the beginning of the game to inform the player about movement and shooting commands.
- `render(SpaceShip spaceShip, List enemies, List bullets, int score)`: Draws the current state of the game on the console.
    - Creates a 2D grid using characters to represent the game area.
    - Places all active entities (spaceship, enemies, and bullets) on their corresponding coordinates.
    - Calls `printField()` to display the grid.
    - Displays the player's score, health, and number of active enemies below the grid.
- `place(char[][] grid, int x, int y, char c)`: Helper method that places a single entity on the grid if the specified coordinates are within the game's boundaries.
- `printField(char[][] grid)`: Prints the full grid surrounded by borders to give a framed visual effect.
    - Top and bottom borders use the `'='` character.
    - Side borders use the `'|'` character.
    - Called internally by `render()` to output the game's visual frame.

`GameField` Class:

This class manages the main gameplay environment for the Space Invaders game. It keeps track of all active entities in the game, including the player's spaceship, enemies, and bullets. The class handles spawning enemies, detecting and resolving collisions, and maintaining the player's score.

Instance Data Members:

- `spaceShip`: A reference to the player's spaceship controlled by the user.
- `enemies`: A list containing all active enemy objects currently on the field.
- `bullets`: A list containing all bullets fired by the player.
- `score`: An integer variable tracking the player's score, which increases each time an enemy is destroyed.

Methods:

- **`Constructor:`**
    - `GameField(SpaceShip spaceShip, List enemies, List bullets)`: Initializes the game field with references to the spaceship, enemy list, and bullet list. Also initializes the score to zero.

- **Service Methods:**
  - `spawnEnemies()`: Spawns a fixed number of enemies `(GameEngine.ENEMY_PER_ROW)` at the top of the grid each turn. Each enemy appears at a random horizontal position without overlapping another enemy in the same column.
  - `checkSpaceShipCollusion()`: Detects collisions between enemies and the player's spaceship. If a collision occurs, the enemy is removed and the spaceship's health decreases by one. Returns true if a collision occurred, otherwise false.
  - `checkBulletCollusion()`: Checks for collisions between bullets and enemies. When a bullet hits an enemy, both are removed from the field, and the player's score increases by one. Returns the number of enemies destroyed during this turn.
- **Accessor Methods:**
  - `getScore()`: Returns the current score of the player.

`Entity` Class:

This class serves as the base (superclass) for all game objects that appear in the **Space Invaders** world. It defines shared attributes and behaviors that are common to all entities such as the spaceship, enemies, and bullets. Subclasses extend Entity to inherit its properties and implement their own specific behaviors.

Instance Data Members:

- `x`: The horizontal position of the entity on the game grid.
- `y`: The vertical position of the entity on the game grid.
- `symbol`: The character symbol used to visually represent the entity on scree

Methods:

- **Constructor:**
  - `Entity(int x, int y, char symbol)`: Initializes the entity's position and symbol. This constructor is called by subclasses (SpaceShip, Enemy, Bullet) to set shared attributes.
- **Service Methods:**
  - `collidesWith(Entity other)`: Checks if this entity occupies the same grid position as another entity. Returns true if both entities share the same (x, y) coordinates, otherwise returns false.
- **Accessor Methods:**
  - `getX()`: Returns the entity's current horizontal position.
  - `getY()`: Returns the entity's current vertical position.
  - `getSymbol()`: Returns the character symbol that represents the entity on the screen.

`SpaceShip` Class:

This class represents the player-controlled spaceship in the **Space Invaders** game. It extends the `Entity` superclass, inheriting position and visual representation while introducing new properties such as health and unique behaviors like movement and shooting.

Instance Data Members:

- `health`: The current health points of the spaceship, which decrease when hit by an enemy.

Methods:

- ➢ **`Constructor:`**
  - `SpaceShip(int x, int y, int health)`: Initializes the spaceship's position and health. Calls the superclass constructor to assign its symbol `(SPACE_SHIP_SYMBOL)` and position.
- ➢ **`Service Methods:`**
  - `shoot()`: Creates and returns a new Bullet object fired from the spaceship's current position.
  - `moveLeft()`: Moves the spaceship one cell to the left, ensuring it does not cross the left boundary of the game area.
  - `moveRight()`: Moves the spaceship one cell to the right, ensuring it does not cross the right boundary of the game area.
- ➢ **`Accessor Methods:`**
  - `getHealth()`: Returns the current health value of the spaceship.
- ➢ **`Mutator Methods:`**
  - `setHealth(int health)`: Updates the spaceship's health value, used when taking damage from enemies.

`Enemy` Class:

This class represents the enemy objects that move downward toward the player's spaceship in the **Space Invaders** game. It inherits position and symbol properties from the `Entity` superclass and adds movement and attack behavior specific to enemies.

Methods:

- ➢ **`Constructor:`**
  - `Enemy(int x, int y)`: Initializes an enemy object at the given (x, y) position using the enemy symbol defined in the `GameEngine` class.
- ➢ **`Service Methods:`**
  - `update()`: Moves the enemy one step downward on the grid by increasing its y coordinate. This simulates the enemy falling toward the player's spaceship each turn.
  - `attack(SpaceShip ship)`: Checks for a collision between the enemy and the player's spaceship. If a collision is detected, the spaceship loses one health point, and the method returns true; otherwise, it returns false.

`Bullet` Class:

This class represents the bullets fired by the player's spaceship in the **Space Invaders** game. It extends the `Entity` superclass, inheriting position and symbol properties common to all game objects. Bullets move upward each turn and can collide with enemies to destroy them.

Methods:

- ➢ **`Constructor:`**
  - `Bullet(int x, int y)`: Initializes a bullet at the given (x, y) position using the bullet symbol defined in the `GameEngine` class. Calls the superclass constructor to set its position and visual representation.
- ➢ **`Service Methods:`**
  - `update()`: Moves the bullet one step upward on the grid by decreasing its y coordinate. This simulates the projectile traveling upward after being fired by the spaceship.

- `collidesWith(Entity other):` Determines whether the bullet has collided with another entity (in this case, an enemy). A collision is detected when both objects share the same x coordinate and their y coordinates differ by at most one unit, allowing accurate detection even when entities move quickly past each other.

3. Gameplay

You can view a sample gameplay here: lab8.mov.
Additionally, you can access the sample output as a text file here.