Faculty of Computing

Universiti Teknologi Malaysia

SECJ2154, Object Oriented Programming

Semester 2, 2022/2023

Individual Report for the Group Project

Notes: *This report is to be submitted individually.*

Student's Name	Muhammad Adam Fahmi Bin Mohd Taufiq
Group Project Title	Brick Breaker

A. Implementation of Classes

Describe the class (or classes) you implemented for the project. Give the estimate percentage of how much you did on the class. For example, if you did all the code for the class, then you write 100%. Also, give the location of the code you did.

Class Name	**Percentage of contribution**	**Location**		**Remarks**
		File	**Line Numbers**	
MapGenerator	100%	MapGenerator.java	1 - 59	The MapGenerator class is responsible for generating and drawing a game map composed of bricks. It provides a constructor to initialize the map with default values and a draw method to render the bricks on the screen. The setBrickValue method allows for modifying individual bricks in the map, and a getter method getMap retrieves the map array. However, further

				functionality such as collision detection and integration with other game components would need to be implemented to create a fully functional game using this map generator.

B. Implementation of OOP Concepts

Describe the concept (or concepts) you implemented for the project. Give the estimate percentage of how much you did on the concept. Explain why the concept is needed in your project, and explain the general idea of how you implemented it.

The concepts to be described here include ***Association, Inheritance and Polymorphism****. Note that, each member of a group is not necessarily to implement all the concepts.*

OOP Concept	**Percentage of contribution**	**Location**		**Why is this concept needed?**	**General idea of the implementation**
		File	**Line Num.**		
Encapsulation	100%	MapGenerator.java	● 6-8	Encapsulation is needed in the MapGenerator class to ensure the integrity and maintainability of the code. By encapsulating the internal state and behavior of the MapGenerator class, we can control access to its variables and methods. This helps to prevent direct manipulation of the class's internal data by external entities, ensuring that the map is generated and modified in a controlled manner. Encapsulation also allows us to hide the implementation details of the map generation process, providing a clean and consistent interface for other parts of the program to interact with the MapGenerator class. By encapsulating the internal workings of the class, we can easily modify or extend the functionality of the MapGenerator without affecting other parts of the code, promoting code reusability and maintainability.	The MapGenerator class is responsible for generating and drawing a map consisting of regular and special bricks. It uses a 2D array to represent the map, with each element representing a brick. The class provides methods to set the values of individual bricks, allowing for customization of the map. The draw() method utilizes the Graphics2D object to render the bricks on the screen. The class demonstrates encapsulation by hiding the internal details of map generation and providing a clean interface to work with the map. Overall, the MapGenerator class encapsulates the logic of map generation and drawing, making it modular and easy to use in a game or application.

Association	100%	MapGenerator.java	● 27-47	The association between the MapGenerator class and the Graphics2D class is needed to enable the MapGenerator class to draw the map on the screen. By accepting a Graphics2D object as a parameter in the draw() method, the MapGenerator class can utilize the drawing capabilities provided by the Graphics2D class. This association allows the MapGenerator class to leverage the functionality of the Graphics2D class to render the bricks with the specified colors, sizes, and positions on the screen. Without this association, the MapGenerator class would not have access to the necessary tools to visually represent the generated map.	In the MapGenerator class, the association is implemented by accepting a Graphics2D object as a parameter in the draw() method. This allows the MapGenerator class to utilize the functionality provided by the Graphics2D class for drawing on the screen. By establishing this association, the MapGenerator class can set the color and dimensions of the bricks and use the Graphics2D object to draw the rectangles representing the bricks on the screen. This association enables the MapGenerator class to leverage the drawing capabilities of the Graphics2D class to visually render the generated map.
Abstraction	100%	MapGenerator.java	● 27-47	Abstraction is needed in the MapGenerator class to provide a simplified and higher-level interface for generating and drawing maps. By encapsulating the details of map generation and drawing within the MapGenerator class, other parts of the program can interact with it using a more abstract and intuitive interface. The MapGenerator class hides the complexities of the map generation algorithm, the calculations for brick positions and sizes, and the drawing operations, allowing other components to use the class without needing to understand or manage these internal details. Abstraction enables better separation of concerns, promotes code reusability, and enhances maintainability by encapsulating complex logic and providing	The implementation of abstraction in the MapGenerator class involves encapsulating the details of map generation and drawing, providing a higher-level interface for interacting with the map functionality. The class hides the complexity of generating and drawing the map by abstracting the specific algorithms and operations behind methods like draw() and setBrickValue(). This allows other parts of the program to interact with the MapGenerator class without needing to understand or manage the intricacies of the map generation

				a clean and understandable interface for interacting with the map generation functionality.	process. The abstraction in the MapGenerator class promotes modularity, code reusability, and maintainability by providing a clear separation of concerns and a simplified interface for working with maps.

C. Other Implementations (Optional)

This part is only to be filled in should you have other things you did for your project but have not been mentioned in Part A and B.

Things / Code Done	**Percentage of contribution**	**Remarks**
• Polymorphism	10%	I give some help in implementing the Polymorphism concept in the Brick class