A red square with white text

Description automatically generated

COS20007

**Object Oriented Programming**

Learning Summary Report

VU PHAN

104222099

104222099@student.swin.edu.au

1. **Introduction**

TetrisV is an ode to the classic Tetris game, aiming to recreate the nostalgic gameplay experienced on the Gameboy console and the original 1984 version. It encompasses various components, including a display manager, frame organizer, user controls, scoring system, and Tetromino manager. The game aims to engage users in an interactive Tetris experience, manipulating falling Tetrominos to complete rows and achieve high scores.

1. **Submission Details**
2. **Code Files: <https://drive.google.com/drive/folders/1jBq7hSPkSS75KIpX91-iuQLalk5MDklU?usp=sharing>**
3. **UML Class Diagram:**

A screenshot of a computer

Description automatically generated

A diagram of a game

Description automatically generated

1. **Design Overview**
2. **Key Feature.**

The custom program demonstrates the following features:

1. Tetromino Manipulation: Players can rotate and move falling tetromino shapes to fit into available spaces, aiming to create solid rows without gaps.
2. Scoring System: The game keeps track of completed rows, awarding points for each row filled. Clearing multiple rows simultaneously earns bonus points.
3. Increasing Difficulty: As players progress, the game speeds up, challenging them to react faster and make decisions more quickly.
4. Display Manager: Manages the game screen, presenting the falling tetromino and the game board. It showcases essential information, such as the next tetromino.
5. Controls: Allows players to interact with the game using keyboard controls, enabling movement, rotation, and pause functionality.
6. Leveling System: As players achieve higher scores or complete rows, they advance to higher levels, increasing the game's speed and difficulty.
7. Game Over Condition: When tetrominoes reach the top of the game board, the game ends. It displays the final score and allows players to restart.
8. **Classe segmentation.**

| **Class** | **Description** | **Responsibilities** |
| --- | --- | --- |
| Display | Manages screen output | - Display frames - Set frame elements - Manage game screen dimensions - Handle active state and position |
| Frame | Organizes game frames | - Manages walls in the frame - Sets introduction text - Lists all walls in the frame |
| Wall | Handles wall elements within frames | - Manages array of wall values - Handles wall lists - Controls introduction state |
| Key | Manages user controls | - Maps key-color pairs for controls - Handles key press events |
| Move | Controls movement direction | - Manages movement directions |
| Check | Manages the checking mechanism | - Indicates safe condition |
| Preview | Handles the next tetromino info | - Generates the next tetromino |
| Reset | Handles the reset event | - Resets the game |
| Rotate | Manages the checking mechanism for rotation | - Handles rotation events - Checks wall boundaries for rotation - Controls rotation process - Inverts rotation for calculations |
| Score | Manages rows and scoring | - Manages completed rows - Calculates scores for rows - Updates the scoreboard |
| Row | Handles completed and cleared rows | - Manages completed row indices - Counts completed rows - Manages additional score bonuses |
| ScoreBoard | Handles scoring information | - Manages completed rows, current level, and overall score |
| Speed | Manages speed settings | - Checks speed conditions - Handles pause functionality |
| Set | Manages game settings | - Manages time delay for movements - Controls pause and tetromino dropping |
| Tetrominos | Handles block arrangements in the game | - Generates new tetrominos - Sets tetrominos on the board |
| Block | Manages tetromino arrangements | - Manages current, next, and placed tetromino arrangements - Controls arrangement setting |

1. **Methods**

Main method:

1. **SetFrame():**
   * **Class:** **Frame**
   * **Description:** Sets up the game frame elements including walls and visual components essential for gameplay.

**A screen shot of a computer

Description automatically generated**

1. **Press():**
   * **Class:** Key
   * **Description:** Handles key press events, allowing control over game elements like movement or rotation of tetrominos.

**A screen shot of a computer program

Description automatically generated**

1. **RowCheck():**
   * **Class:** Row
   * **Description:** Checks the game grid for completed rows, triggering actions upon completion.

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. **Now():**
   * **Classes:** Reset, Rotate, Set, and potentially others
   * **Description:** Executes specific actions like game resets, rotations, and settings management.

A screen shot of a computer program

Description automatically generated

1. **New():**
   * **Class:** Tetrominos
   * **Description:** Generates new tetrominos, crucial for gameplay advancement.

A screen shot of a computer program

Description automatically generated

1. **Check():**
   * **Class:** Speed
   * **Description:** Verifies speed conditions, potentially controlling game pace or tetromino falling speed.

A screen shot of a computer program

Description automatically generated

1. **Design features**
2. **Frame Management:**
   * **Classes:** Frame, Display
   * **Description:** Organizes and manages the game display, responsible for rendering and updating visual components.

A screenshot of a computer

Description automatically generated

1. **Control Handling:**
   * **Class:** Key
   * **Description:** Allows user interaction via keyboard inputs, controlling various game aspects such as movement and rotation.
2. **Scoring Mechanism:**
   * **Classes:** Score, Game
   * **Description:** Manages and updates the player's score based on completed rows or other game achievements.

A screenshot of a computer

Description automatically generated

1. **Tetromino Manipulation:**
   * **Class:** Tetrominos
   * **Description:** Governs tetromino behavior including movement, rotation, and placement within the game grid.

A screenshot of a computer

Description automatically generated

1. **Game Speed Control**
   * **Class:** Speed
   * **Description:** Controls the pace or difficulty of the game, potentially affecting tetromino falling speed or other game dynamics.

A screenshot of a computer program

Description automatically generated

1. **Wall and Boundary Management**
   * **Classes:** Wall, Frame
   * **Description:** Ensures tetrominos adhere to game boundaries, preventing invalid movements or placements outside the game grid.

A screenshot of a computer

Description automatically generated

1. **Concept of object-oriented programming**
2. **Abstraction**

Abstraction is the concept of hiding the implementation details and showing only the necessary features of an object. In my code, the use of abstract classes signifies abstraction, particularly in Key.cs and Game.cs.

A screen shot of a computer program

Description automatically generated

Here, **Key** is an abstract class with an abstract method **Press()**. The **Key** class is intended to be a base class where specific key presses are handled in derived classes. The **Press()** method is defined as abstract, which means any class inheriting from **Key** must implement its own version of **Press()**.

1. **Inheritance**

Inheritance is the mechanism where one class acquires the properties and behaviors of another class. You've used inheritance in several places, particularly in classes inheriting from the base class **Key**.

A screen shot of a computer program

Description automatically generated

In **ConcreteKey.cs**, the class **ConcreteKey** inherits from the **Key** class using the **: Key** syntax. It then overrides the abstract **Press()** method, providing its own implementation for handling key presses.

A screen shot of a computer program

Description automatically generated

The **Game** class is an abstract class containing abstract and virtual methods. This allows subclasses to override these methods to provide their own specific functionality while inheriting the common features defined in the **Game** class. In this case:

A screen shot of a computer program

Description automatically generated

1. **Conclusion**

The custom program aligns with the objectives of showcasing a comprehensive understanding and application of object-oriented programming concepts. It exemplifies adherence to coding standards, effective use of UML diagrams, and successful program implementation that demonstrates the desired complexity and functionality.