Odd Semester (2023)

BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	**Surname**	**Given Names**	**Student ID Number**
1.	Frans	Christensen Mario	2301963316

Course Code	**:** COMP6502	**Course Name**	**:** Programming Languages
Class	**:** L1AC	**Name of Lecturer(s)**	**:** Mr. Jude Martinez
Major	**:** CS		
Title of Assignment (if any)	: Tetris Game Project Java		
Type of Assignment	**:** Final Project		
Submission Pattern			
Due Date	**:** 20-06-20	**Submission Date**	**:**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.

2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

CHRISTENSEN MARIO FRANS

(Name of Student)
Christensen Mario Frans

Table of Contents

Project Background

Problem

As a teenager, it is common that I found myself in a situation where I wished time would've accelerated, such as waiting for the internet to buffer, attaching large files on emails, or even waiting for my coffee at Starbucks. Likewise, I realized many people often meet similar situations…

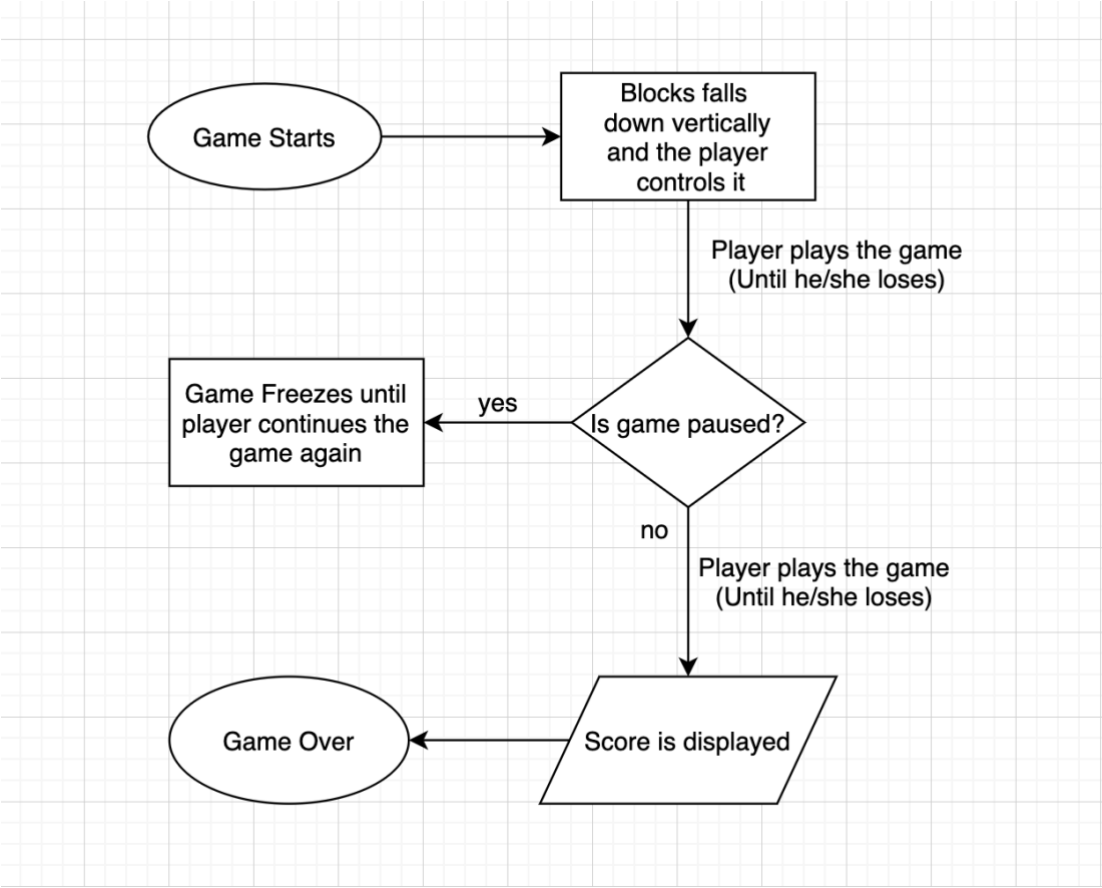
Solution

The solution I proposed for this Java project was to make a very simple time-killing computer game; Tetris. In this game, the player controls a piece of 'block', which is moving down the screen. Players have to make a row full of blocks, then, that row will disappear and the player will be rewarded with a score for each row cleared. The objective of this game is to gain as much score as possible. The player must also avoid stacking the blocks too much and not clearing them, as this makes it harder for newer spawned blocks to move around. If blocks stack up to the brim of the game screen, then the game is over.

Tetris has been a very popular game worldwide for decades since its first release in 1984. It's simple yet addictive behavior catches the eyes of many individuals, as players do not have to think much. Not only it is simple to understand for beginners, it is also quick. These are a few reasons why I suggest making a Tetris game as a solution is a perfect one. Since it is common for people to impatiently wait for time to pass by, they might as well play a enjoy a game of Tetris while waiting!

Solution Design

Flowchart of Program



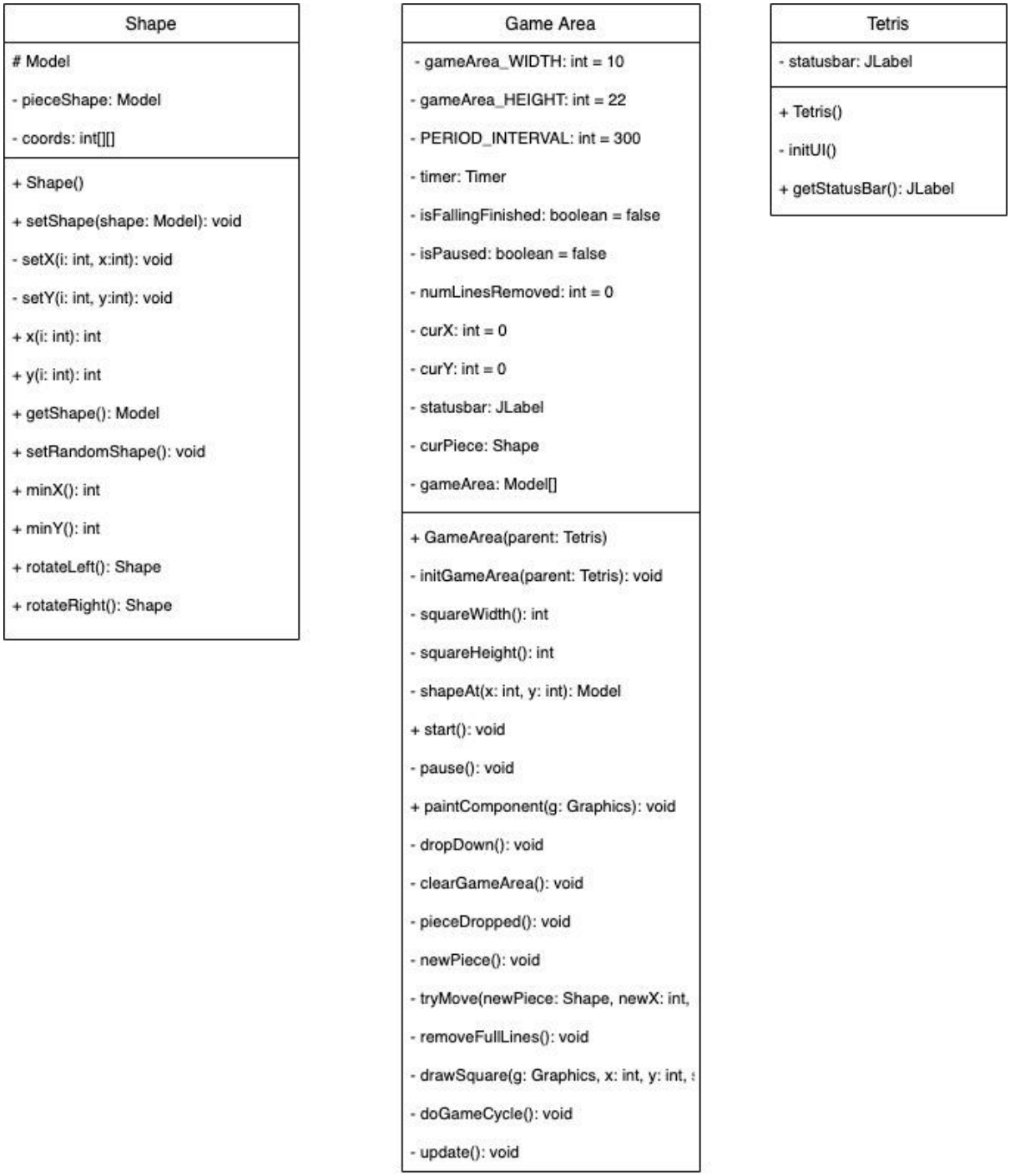
As seen in the flowchart above, the game will start immediately as soon as the player launched the program. The Tetris blocks will spawn on the top of the game area, and the players can control the blocks by moving it around the game area. The controls of the blocks are:

1. “UP-button” – rotates the block clockwise
2. “DOWN-button” – moves the block down manually
3. “LEFT-button” – move the block one step to the left
4. “RIGHT-button” – move the block one step to the right
5. “SPACE-button”- speeds up the game by instantly dropping the block to the bottom; x-coordinate stays the same

Players can also pause the game by pressing the “P” key on their keyboard. This is useful as players can pause the game in case they have to leave the game immediately and temporarily, without closing the game and losing their current progress. To continue the game, simply press “P” again.

For each row is filled with blocks, it will be cleared and the score count will add by 1. The player plays the Tetris game until he/she loses; that is when the piled blocks reach the brim of the game screen. Once the player loses the game, their total score will be shown in the label on the bottom of the game window.

UML Diagram of Program



As seen in the UML diagram above, there are 3 different classes in this program. The UML seems very complex since the classes have so many variables & methods in it, but in reality, it's pretty simple to understand.

The 'Shape' class designs & draws the shapes, uses "Java Swing Painting API" to draw the Tetris tiles/sprites, spawns a random Tetris block shape, and sets their coordinates as well movements/rotations that is controlled by the player.

The 'Game Area' class sets & displays the game window sizes and labels using both JFrame and JLabel libraries, and uses "Java Utility Timer" to create the game cycle. It also includes the start game function, pause game function, and update function (determines if the game continues or is game over).

Finally, the 'Tetris' class is responsible for starting the program, and creating the initial user interface when the game starts; such as a new game area (window) object, and setting up its size as well as its title.

Project Specifications & Libraries

For the development of this project, I'm using Java programming language for the coding, and Eclipse as the IDE (Integrated Development Environment). One benefit of using Java as the programming language is that since it is a compiler, the program scans all the codes involved in the running of the game, which is able to determine which codes run first and foremost, unlike an interpreter, which runs the codes chronologically from line to line.

For this game, I will be using some of the built-in libraries in Java such as "Java Utilities", "Java Swing Painting API" to draw the Tetris tiles/sprites, "Java Utility Timer" to create the game cycle, "Java Utility Random" to randomly select the shapes to spawn. I will be discussing more about the program in the next section, including UML diagrams and flowcharts.

What was implemented and how it works

In this section, I will discuss on the algorithms implemented on this project which I learned in class.

Looping:

```
// scans and clears any full row
private void removeFullLines() {

    int numFullLines = 0;

    // scans the entire window
    for (int i = gameArea_HEIGHT - 1; i >= 0; i--) {

        boolean lineIsFull = true;

        for (int j = 0; j < gameArea_WIDTH; j++) {

            if (shapeAt(j, i) == Model.NoShape) {

                lineIsFull = false;
                break;
            }
        }

        // if row is full, increase the counter by one, and clear the row
        if (lineIsFull) {

            numFullLines++;

            for (int k = i; k < gameArea_HEIGHT - 1; k++) {
                for (int j = 0; j < gameArea_WIDTH; j++) {
                    gameArea[(k * gameArea_WIDTH) + j] = shapeAt(j, k + 1);
                }
            }
        }
    }
```

In this program, I implemented several looping statements for many different purposes. One such example is the 'removeFullLines()' method. This method contains nested for loops which functions to scan the whole game window; from the top(gameArea_HEIGHT) to bottom(0), for any full rows and clears it. This is also how the scoring system is calculated. For each row cleared, the counter will add by 1. This value of the local counter in the method will then be added to a global counter which holds the player's scores.

Arrays:

```
// array to store the game area's blocks, so the blocks will not overlap each other
private Model[] gameArea;
```

I also used some array in my program. This array stores the blocks that are already spawned in the game, and that they already fall to the bottom and cannot be moved anymore. This is done so that newly-spawned blocks that fall will not overlap the blocks that are already at the bottom, instead, it will fall on top of it.

Methods:

```
// this method assigns all the different coordinates to and create the shape of each model
void setShape(Model shape) {

    // each model is assigned to their specific coordinates
    int[][][] coordsTable = new int[][][]{
            {{0, 0}, {0, 0}, {0, 0}, {0, 0}},
            {{0, -1}, {0, 0}, {-1, 0}, {-1, 1}},
            {{0, -1}, {0, 0}, {1, 0}, {1, 1}},
            {{0, -1}, {0, 0}, {0, 1}, {0, 2}},
            {{-1, 0}, {0, 0}, {1, 0}, {0, 1}},
            {{0, 0}, {1, 0}, {0, 1}, {1, 1}},
            {{-1, -1}, {0, -1}, {0, 0}, {0, 1}},
            {{1, -1}, {0, -1}, {0, 0}, {0, 1}}
    };

    // assign one of the coordinates into only one of the Tetris models
    // this will be done more later, for other models when setShape() method is called
    for (int i = 0; i < 4; i++) {

        System.arraycopy(coordsTable[shape.ordinal()], 0, coords, 0, 4);
    }

    pieceShape = shape;
}
```

There are also several methods which I used for running the game. For example, this method assigns the coordinates for each different model (as Tetris blocks have a few different shapes), and stores it in the 'coordsTable' variable. This means, each model has their own specific layout.

Class & Inheritance:

```
// this class is a blueprint of all the game logic
public class GameArea extends JPanel {
```

For this program, I also implemented inheritance. For this example, the 'GameArea' class extends JPanel, which is a Swing package that stores many built-in components. As the child class inherits all the components of the parent class, I can take advantage of this as not only my codes are shorter, I also do not need to re-type my own codes to create the same thing.

Boolean:

```
// boolean to determine if the game is paused or not
private boolean isPaused = false;
```

These are some Booleans which I used in my codes. The 'isPaused' boolean determines if the game is paused or not; if it's false, then it's not paused, if it's true, then the game is paused. This Boolean will be used later in the program to determine what happens when the game is paused, and will be used to resume the program too.

If & Else:

```
// updates the instances of the game, keeps updating as the game is still running
private void update() {

    // if game is paused
    if (isPaused) {

        return;
    }

    // if blocks have fallen completely
    if (isFallingFinished) {

        isFallingFinished = false;
        newPiece();
    } else {

        // if blocks have not reached the bottom, move it down one step
        oneLineDown();
    }
}
```

This update method includes some of the if-statements; including nested if-statements which I included in my program. The update method updates each instance of the game for the Tetris block controlled by the player. To do this, it runs a few if statements. If the game is paused, then, nothing happens, thus nothing is returned, the game will freeze until the player resumes. Next, to check if the block has fallen completely (whether it's on top of another block or on the bottom most of the game area), then a new shape will be issued with the 'newPiece()' method. Otherwise, the block will be moved down by another step.

Comments:

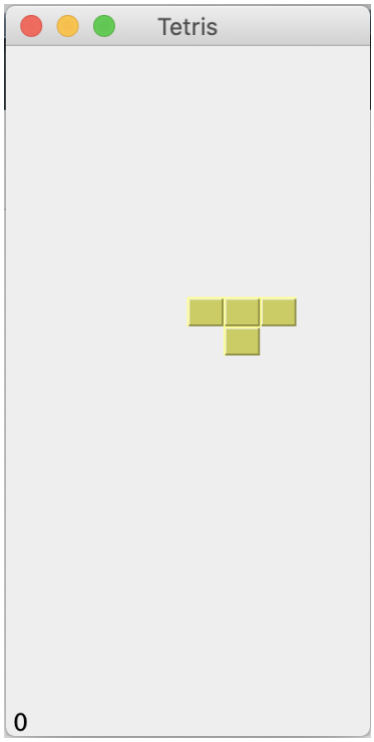
```
// these are 3 constants of the game which will not be changed as the game runs
// gameArea_WIDTH and gameArea_HEIGHT constant determines the size of the board
// PERIOD_INTERVAL constant determines the speed of the game
private final int gameArea_WIDTH = 10;
private final int gameArea_HEIGHT = 22;
private final int PERIOD_INTERVAL = 300;

// create the timer variable for the game using javax.swing.Timer
private Timer timer;
// boolean to check if the block has fallen to the bottom
private boolean isFallingFinished = false;
// boolean to determine if the game is paused or not
private boolean isPaused = false;
// counts the number of lines removed, which will be used to calculate the player's score
private int numLinesRemoved = 0;
// curX and curY determines the exact position of the falling Tetris block that we are controlling
private int curX = 0;
private int curY = 0;
// creates a statusbar using JLabel to show the status; score, paused, or when game is over
private JLabel statusbar;
// curPiece variable is the current Shape variable that we are controlling
private Shape curPiece;
// array to store the game area's blocks, so the blocks will not overlap each other
private Model[] gameArea;
```

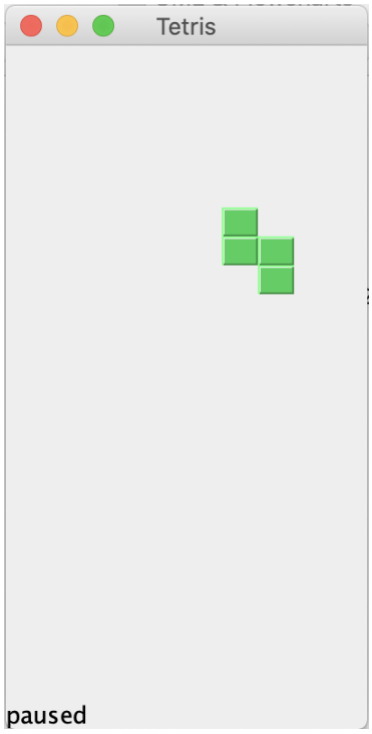
I used a lot of comments in the program as I find it very useful to remind me what the code that I created does. Without them, I might even forget what some the codes are supposed to be. It also makes other programmers understand my code more easily. I did this to all the variables & methods in this program.

Evidence of the Working Program(Including Screenshots):

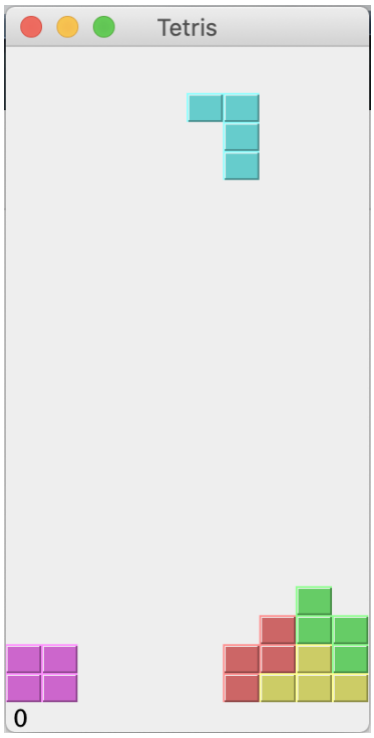
(Disclaimer: since Tetris is a high-speed game, it is not possible to take screenshots of some specific events as things happen too fast. Thus, the screenshots gathered below are just basic ones. To fully test and proof the interface of the program, it is recommended run the code.)



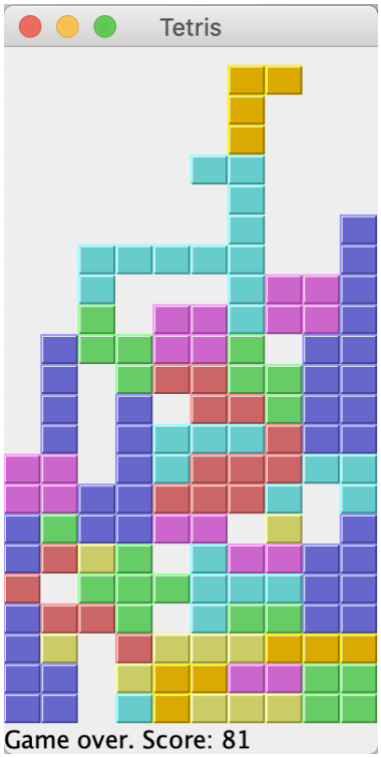
1. This screenshot below shows the interface when the program is launched, the Tetris game starts immediately; with the first piece of block spawned on the top of the game window, and the initial score is zero.



2. The user is able to pause the game by pressing the 'P' button on the keyboard. If this is done, the status bar on the bottom of the window will show "paused" as seen in the picture below, and the Tetris block will freeze. To continue, simply press "P" again.



3. The next screenshot shows the blocks moving down the screen while the player is controlling it. For each row is filled with blocks, it will be cleared and the score count will add by 1. The player plays the Tetris game until he/she loses; that is when the piled blocks reach the brim of the game screen.



4. When the player fails to clear rows and the Tetris blocks piled up reaching the brim, it is game over and the player loses the game.

Once the player loses the game, a “Game over” text followed by the player’s total score will be shown in the label on the bottom of the game window.

(*Note: These screenshots are just a few events of the game, for more and complete details of the running game, please refer to the screen-recorded video of myself playing the game which I will be including in the submission.*)

Resources:
Throughout the program, I used the following site for reference for some of my codes and methods:
http://zetcode.com/tutorials/javagamestutorial/tetris/
Nonetheless, I made several modifications and learn to understand the code line by line, the comments are added so any programmers looking through my program can understand each lines and their implementations in the program.

Link to Project in GitHub:
https://github.com/mariofrans/S2_Project_Java_Tetris

Link to Video in One Drive:
https://1drv.ms/u/s!AjvcnoBWp9nS4RpVOzD8t-9ivo71?e=EwddI5