SWEN30006 Design Analysis Report - Project 1

We redesigned the Tetris system to better adhere to GRASP principles and patterns. Our design model and design sequence diagrams based on our domain model reflect the actual implementation of the Tetris system. The sections below document our modifications and justify our design.

Please note: Tetris piece classes refers to all I, J, L, O, S, T, Z, P, Q and Plus; whereas Piece refers to the abstract Piece superclass

The IGameController interface and GameController superclass

The initial Tetris game system demonstrated poor extendibility with respect to the addition of difficulty levels. From the use cases, it is evident that the easy, medium and madness levels exhibit differences in: playable piece shape, piece speed, and piece movement functionality. The initial design supported the easy mode. Logic dictating playable piece shape, piece speed, and piece movement functionality was contained in the Tetris class. To accommodate medium and madness levels without refactoring the initial design would require conditional statements to be used. This is an example of **high coupling** between the Tetris class and difficulty levels. Additionally, the large number of responsibilities assigned to the Tetris class in the initial design reflects that it has **low cohesion**.

Our team identified difficulty level to be an aspect of variariation within the Tetris system. Hence, we developed the `IGameController` interface which contained methods for controlling playable piece shape, piece speed, and piece movement functionality. In our design, we implemented an easy, medium and madness controller, all of which implemented the IGameController interface. This is an example of **protected variation** and was included in our design because it is likely that our Tetris system will be required to accommodate more levels in the future.

Additionally, we refactored our Tetris class so that it had an attribute `gameController` of type IGameController. Hence, the difficulty level of the Tetris game was a pluggable component into the Tetris class. This is an example of **polymorphism** and supports different levels alongside **low coupling**. Furthermore, the Tetris class can accommodate all levels through a single association with the IGameController interface, unlike the original design which requires associations with each playable level. The IGameController interface allows **delegation** of responsibility from the Tetris class to the GameContoller class. Such a design decision is expected to increase the **cohesion** of the Tetris class by focusing its responsibility.

The abstract Piece superclass

The initial Tetris game system demonstrated poor use of **polymorphism** with respect to Tetris pieces. The Tetris pieces (I, J, L, O, S, T and Z) did not inherit from a common parent class despite their similar functionality and attributes. Hence, the `moveBlock` method in the Tetris class contained **repeated code** that could be omitted through the inclusion of a

superclass for the Tetris classes. Additionally, the lack of a common superclass resulted in associations between all Tetris piece classes and the Tetris class; this is an example of **high class coupling**. Due to the lack of a superclass amongst the Tetris piece classes, significant effort would then be required to collectively change the behaviour of Tetris pieces, and the addition of new Tetris piece classes (such as P, Q and Plus) would exacerbate the problem of repeated code.

The described limitations were addressed through applying **polymorphism**. We collected the common attributes and functionality of Tetris piece classes into an abstract superclass called Piece. All classes representing Tetris pieces inherit from this superclass. This new design reduced instances of repeated code and improved **code maintainability**. For example, functionality could be added to all Tetris pieces by modifying the parent Piece class. Furthermore, this refactored design improved the **extendability** of the Tetris system. For example, pieces P, Q and Plus of the medium and madness levels were implemented by extending the Piece superclass. This mode of extension promoted **code reuse**. Additionally, common Piece superclass for all Tetris classes also **reduced the level of class coupling**. After this superclass was added to the design, the Tetris class could be associated with the Piece class rather than all Tetris piece classes. This improves **code maintainability**.

In the initial design, each Tetris piece class had its own `autoMove` method which contained identical code. This created **coupling** between the Tetris piece classes and the JGameGrid Framework, and arose due to a **lack of polymorphism**, i.e. a common superclass for all Tetris piece classes. Therefore, the Piece superclass was used to avoid this repeated code and excessive coupling. It allowed all Tetris piece classes to retain their `autoMove` behaviour by a **single association** between the superclass and the JGameGrid Framework. Furthermore, we created an association between the Piece class and the classes implementing the IGameController (i.e. GameController classes) to facilitate multiple difficulty behaviours in the `autoMove` method.

The PieceFactory class

In the initial design, Tetris pieces were created in the Tetris class. As a result, the Tetris class was highly coupled with each of the Tetris piece classes. In our design, we created a PieceFactory class which was **delegated** the responsibility of Piece creation from the GameController class which in turn was associated with the Tetris class. This is an example of **pure fabrication**. It **improved the cohesion** of the GameController and Tetris classes and **reduced class coupling** between the Tetris class and all of the Tetris piece classes.

Player Statistics

The Player Statistics class was created to record and output the statistics of a player as required by Tetris Madness, such as the number of pieces and the score for the current round as well as the average score for each round. It uses the principles of **information expert, high cohesion and low coupling**, taking some responsibilities from the Tetris class.

Additionally, it is **extendable** for further additions of Tetris piece classes and difficulty levels, since it makes use of the **polymorphic** properties of these classes.

Diagrams

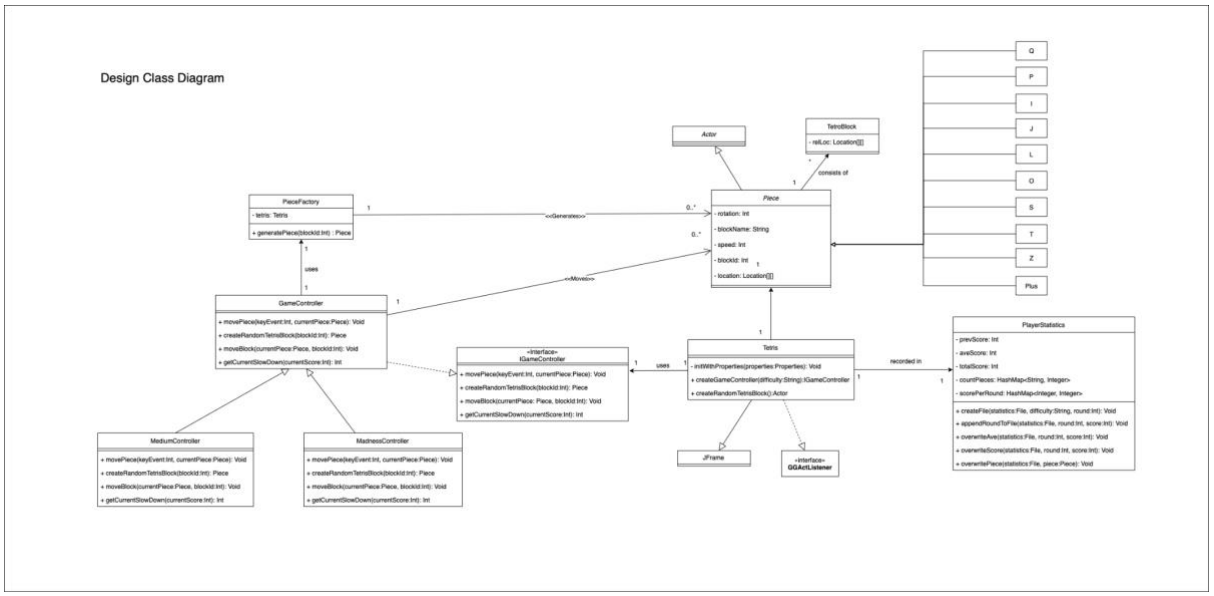

Figure 1: Design Class Diagram

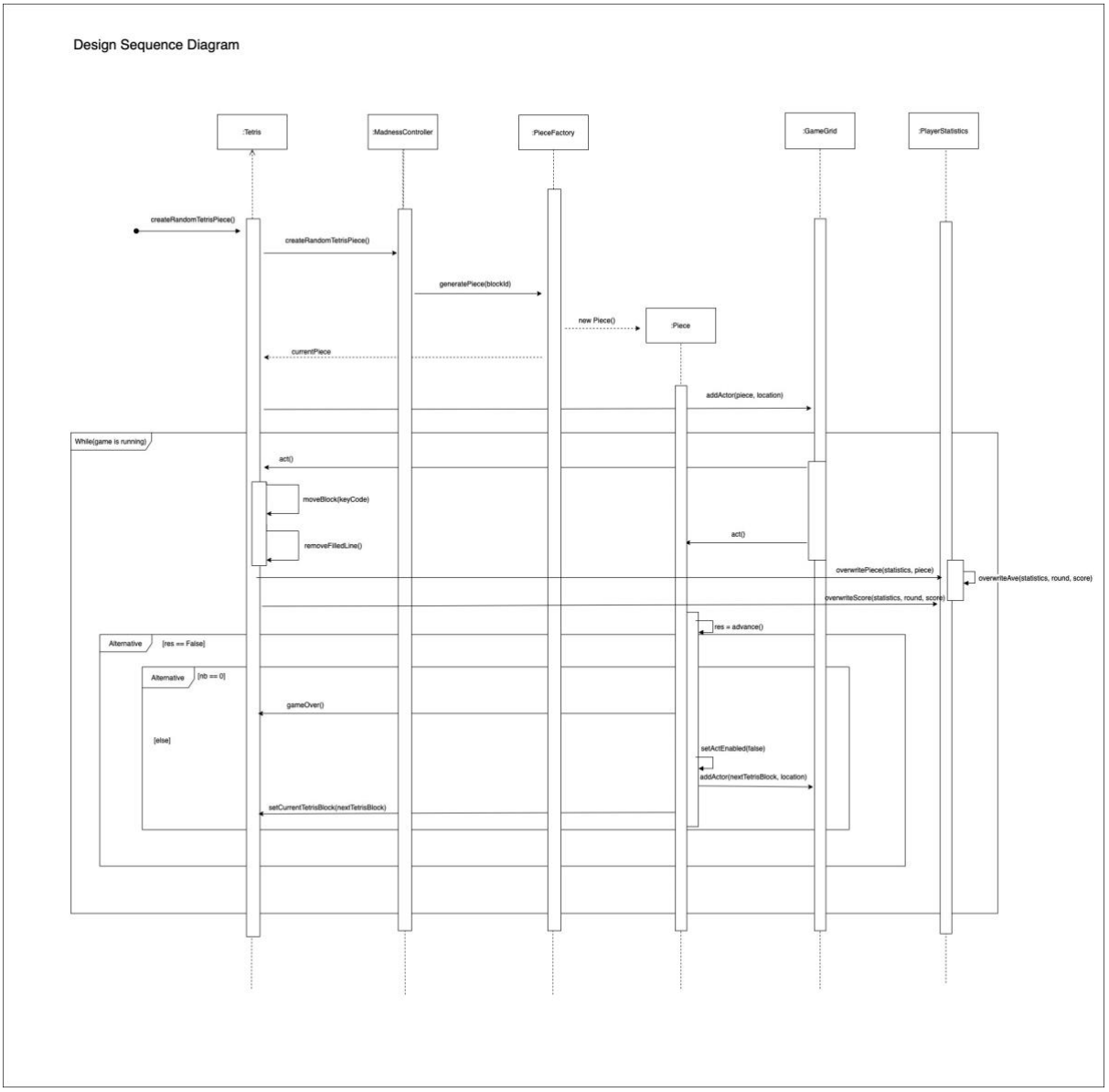

Figure 2: Design Sequence Diagram

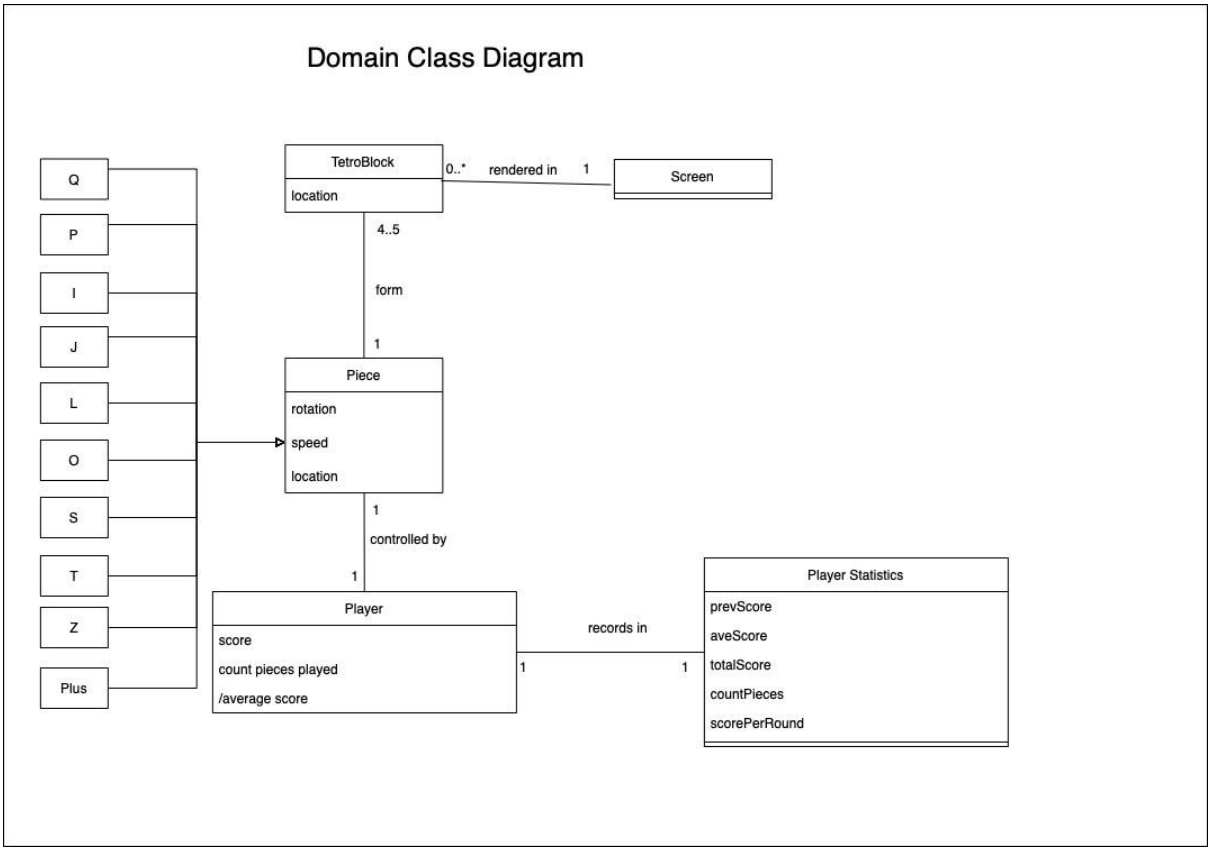

Figure 3: Domain Class Diagram