Tetris - Final Report

Justin Olson, Travis Roundy, Jake Traut

1. List the features that were implemented (table with ID and title).

User Requirements			
ID	**Requirement**	**Topic Area**	**Priority**
UR-02	Player must be able to login.	Authentication	Critical
UR-03	Player can start a new game.	GUI	Critical
UR-06	Player can manipulate the piece on the board	Input mapping	Critical
UR-08	Player can view current in-game score.	Points/GUI	High
UR-09	Player can view threshold line above which pieces cannot be stacked.	GUI	Critical
UR-11	Player can view the next piece in queue.	GUI	High
UR-12	Player can collapse (destroy) stable pieces on board with a “good” placement of piece.	GUI	Critical
UR-13	Player can end the game	GUI	High

Non-Functional Requirements			
ID	**Requirement**	**Topic Area**	**Priority**
NFR-01	GUI must be able to be used by all levels of expertise	Usability	High
NFR-04	Game must run smoothly	Performance	High
NFR-05	Piece must move immediately following user input	Performance	High
NFR-07	User must be able to easily install game	Packaging	Medium

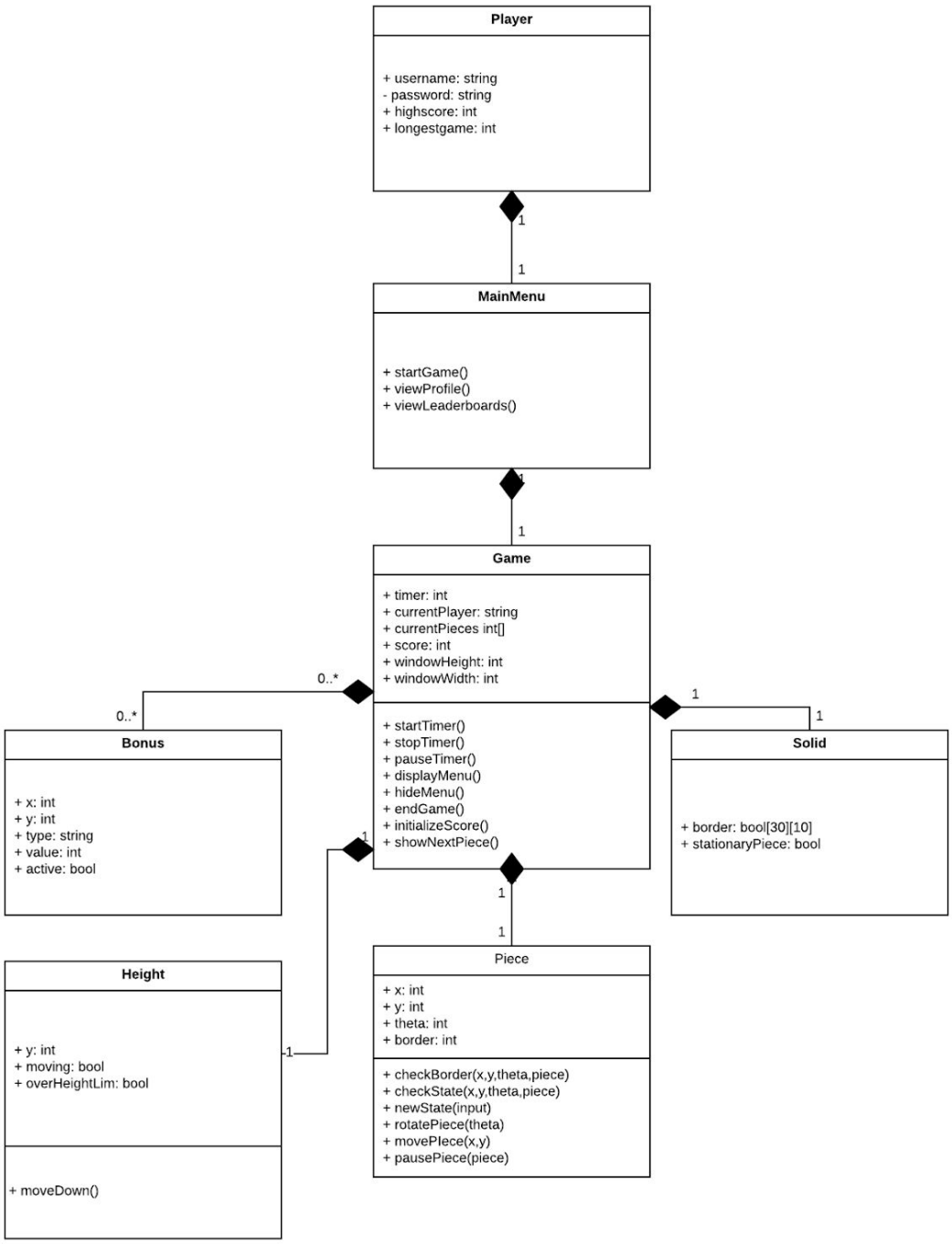
2. List the features were not implemented from Part 2 (table with ID and title).

User Requirements			
ID	**Requirement**	**Topic Area**	**Priority**
UR-01	Player must be able to register username and password.	Authentication	Critical
UR-04	Player can view high score.	Profile	High
UR-05	Player can view position on leaderboards.	Leaderboard (DB)	Medium
UR-07	Player can collect randomly spawned bonuses.	Points	Medium
UR-10	Player can view the time-duration of current game.	GUI	Low
UR-14	Player can view keyboard controls	GUI	High

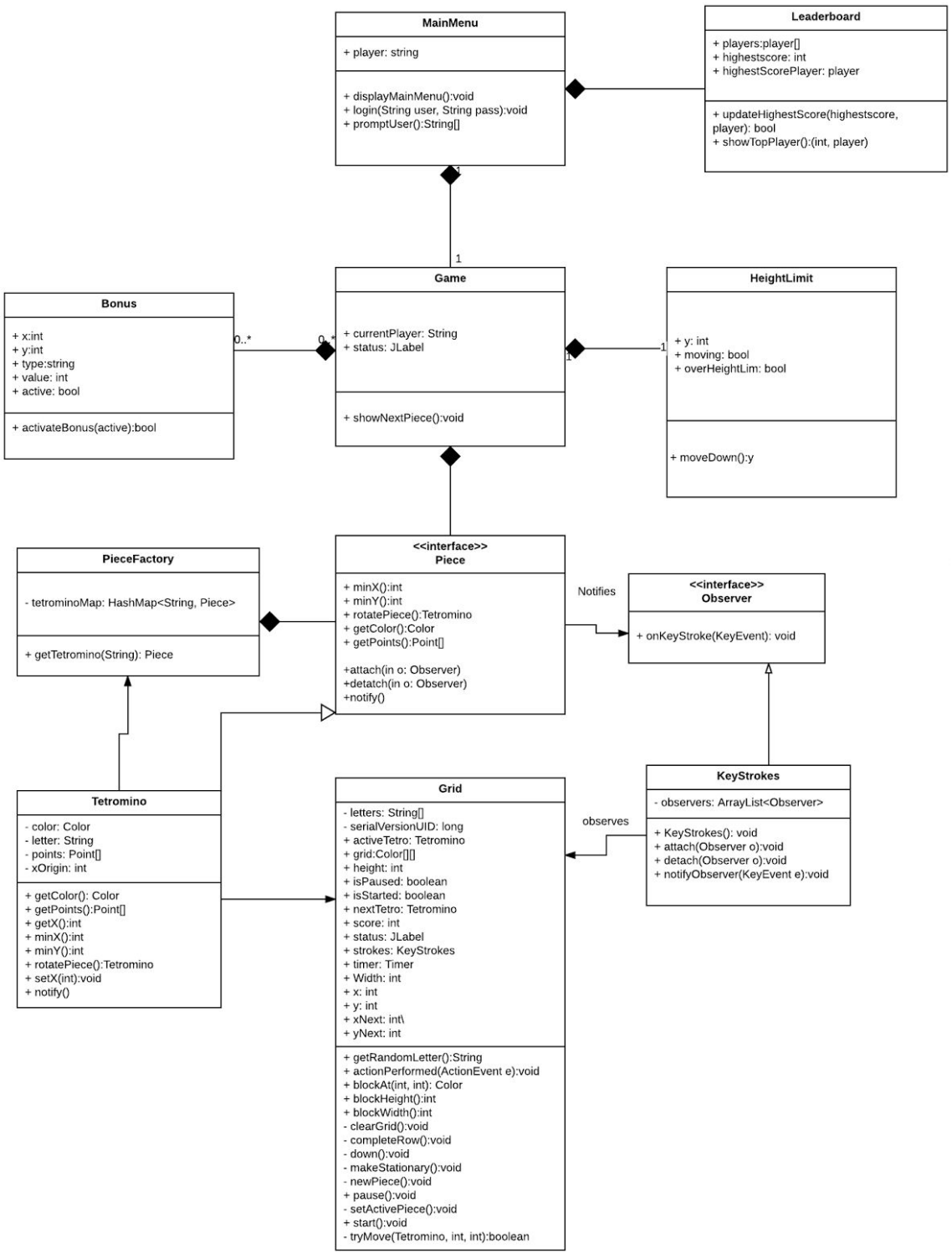
Non-Functional Requirements			
ID	**Requirement**	**Topic Area**	**Priority**
NFR-02	Documentation of how to play must be created	Usability	Medium
NFR-03	Users login must be stored safely in database	Reliability	Medium

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Part 2 Class Diagram



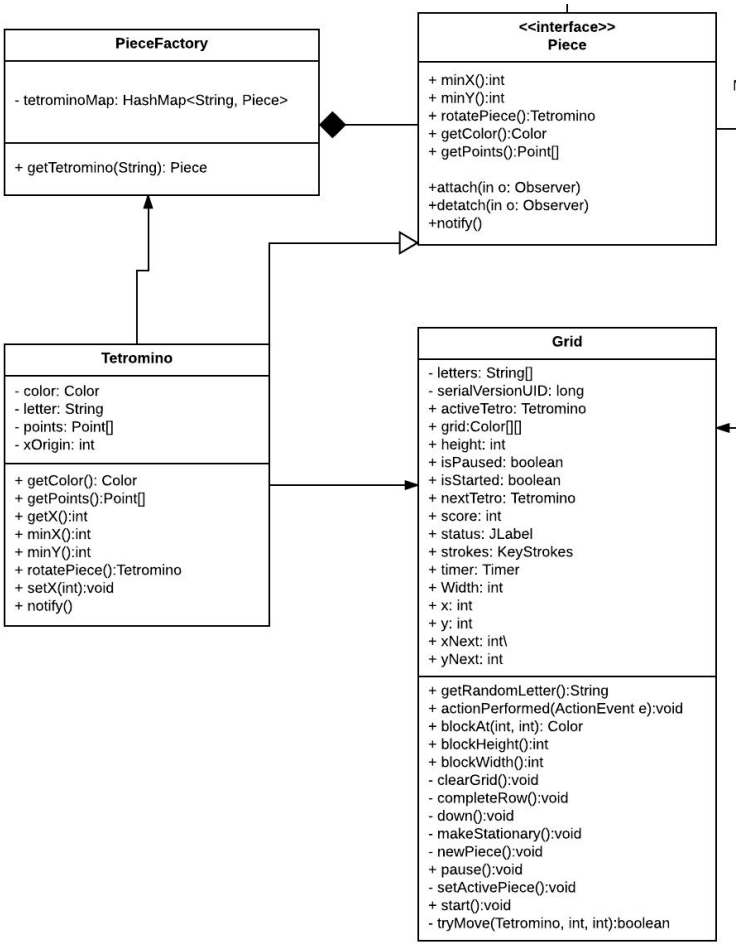
Final Class Diagram



We changed many aspects about our original class diagram. The first major change was the removal of a user. Instead of a singular user class we built these properties into the main menu class. The next major design change came with how the area of filled tetris pieces was represented. Instead of being a solid object class where we stored the area covered by tetrominoes, it is represented as a set of piece objects. Since the original diagram we added an implementation of the observer and flyweight design patterns. These classes monitor and update the pieces, as well as handle the generation, selection, and movement of new pieces. The game board is now represented as a grid. This made it easy to store what type of object was at each location of the playing field. Height was changed to Height Limit. Instead of changing the height of the board, we just set and update a maximum y value that a piece is allowed to cross. Most of the responsibility of Game was distributed among other classes to simplify code.

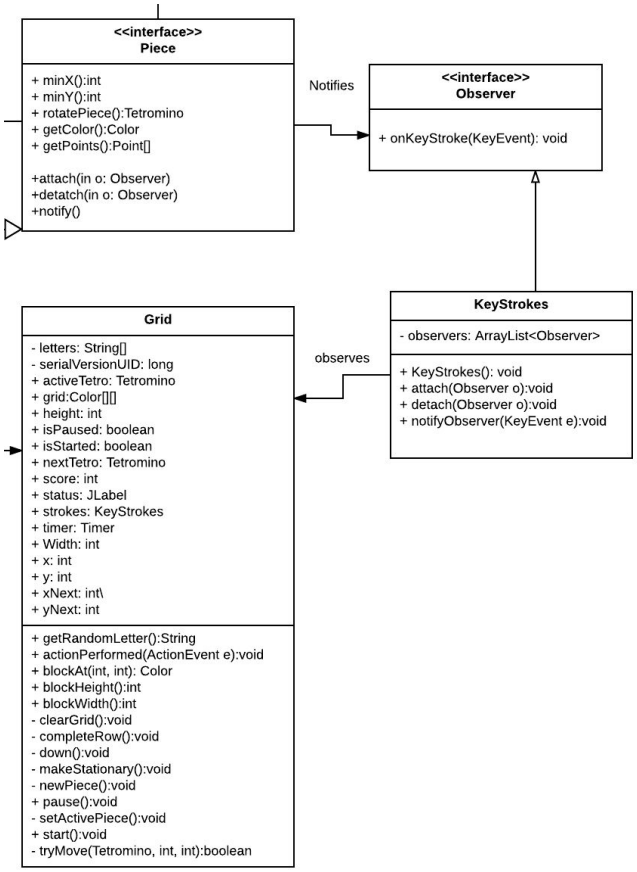
4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

For our final project prototype, we were able to use the Flyweight and Observer design patterns. When making a game with pieces, both the flyweight and observer came in handy. The flyweight design pattern was designed using four classes/interfaces: Piece, Tetromino, PieceFactory, and Grid. With Tetris, there are six different shapes of pieces, so we were able to manage these fine grained objects effectively. Within the PieceFactory class, we are able to generate one of the six shapes used in the game. The Tetromino class manages the position of the Tetromino while it is in the game and we store the points of each piece object generated. Then using methods in other classes, we can manipulate and keep each piece as its own entity. The Grid class is our client class and is where the client and system controls the aspects of the game. Grid combines the Factory as well as the Tetromino classes to manage the entire game. It allows for piece manipulation as well as setting each piece as active or stationary, which will be explained more within the observer pattern. Flyweight was a very useful tool for creating and managing the pieces within our project.



Flyweight Pattern

We also implemented the observer design pattern to monitor keystrokes and update the piece. When a keystroke is entered it calls to the Grid class to decide what to do with that input. Grid then calls the observer onKeyPress() and updates the piece to the proper position. With observer, the active pieces are monitored and can both move and rotate at the same time, helping the game run smoother and allowing the user more control. This design pattern was very helpful and pretty easy to implement making it a nice addition to our project!



Observer Pattern

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Over the course of the semester and most of our careers at CU, we have been taught a lot of theory and methods on how to go about programming and being a computer scientist. In this class, however, we were able to take what we learned in class and put them directly towards a project. The defining of the requirements and creation of a class, sequence, and activity diagrams prior to writing a line of code allowed for us to really think about and see how our project would take shape. Thinking about how each of the classes would work together rather than doing it as we code created a much cleaner code and better understanding of the project as a whole. This process is also great experience when working in industry. This class was also a great demonstration of how a software project adapts overtime. Our initial class diagram was very different from out final one, and all of these different designs were implemented as we developed. As explained in class, as well as demonstrated in some of our Senior Projects, we will have to create a list of requirements for real people and work based off those. Having learned this process, it will make the real experience easier and also make us more qualified. Overall the benefits such as cleaner code and greater understanding of the project make the experiences learned within this class and this project invaluable.