Chewy Lokum Legend Game

Arda Keskiner
Ceyda Dirin
Ege Ilgaz Şencan
Rahime Cansu Ünalır

Meeting Agenda

Progress Report

After last week's meeting, we concluded our open issues:

i. We decided on how to handle special lokum combinations,
ii. When the player exits the game without intentionally saving the game, system will ask player if he/she wants to save the last state.

We did bit of change in our domain model after the meeting. We decided on that 'saveGame' and 'loadGame' are more likely to be functions instead of domain objects.

Meeting Plan

For this week's meeting, we will be discussing on our class diagram, both the one we built first and the updated version of it.

Issues to be discussed at the meeting:

- Level requirements such as number of moves, minimum score to successfully finish a given level,
- In a specific level, will the lokum arrangement be same for every time player pick that level or will it be randomly arranged

After the class diagram discussion, we will be talking about our interaction diagrams.

Next Week

We will start implementing and testing our design. cdirin

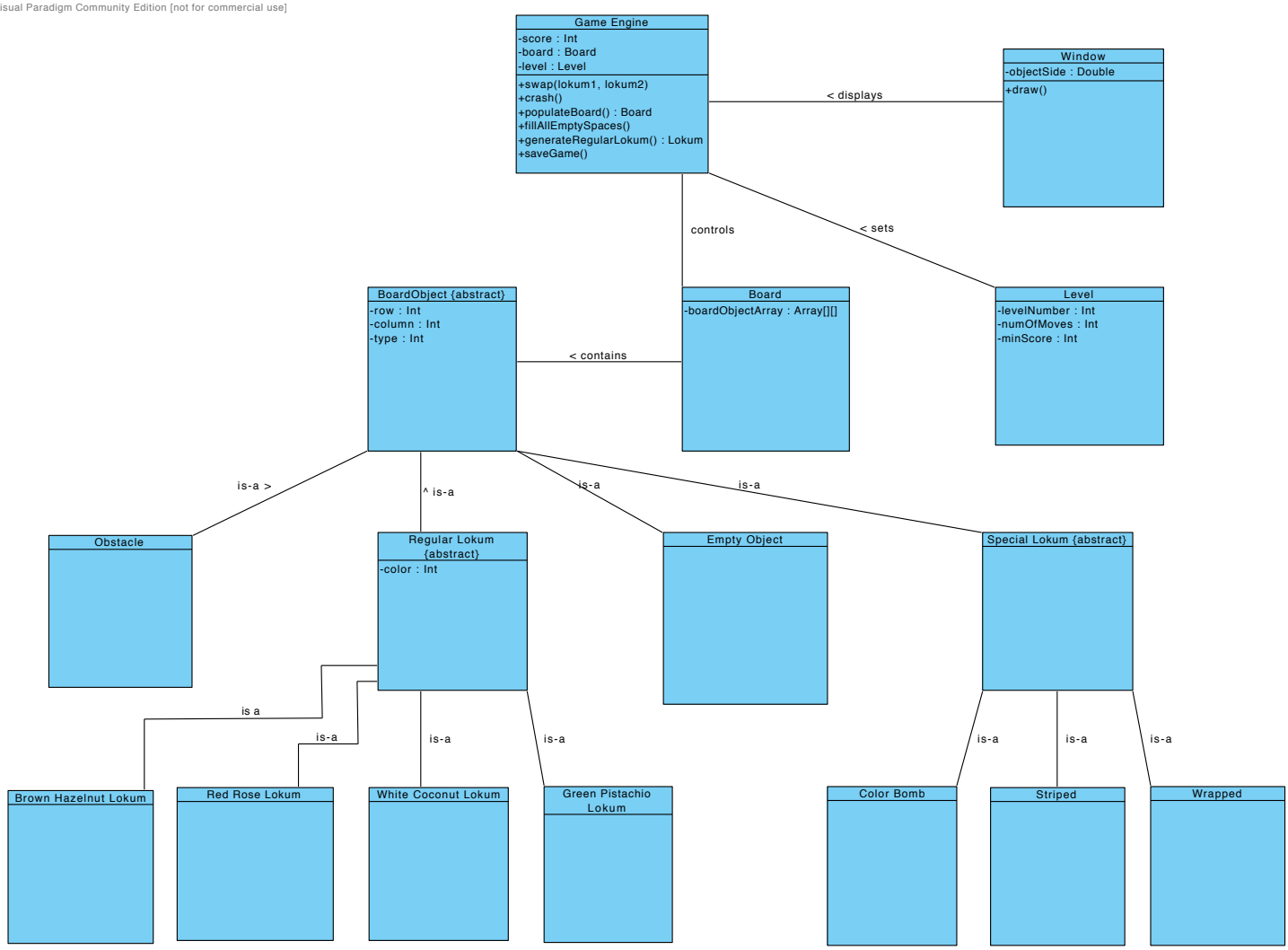
Alternative Designs and Discussions:

While designing our classes we came up with several problems and we tried to solve them by changing our design.

First, obstacles and lokums were different classes and they were connected to “Board” class. Then, we created an abstract superclass called “BoardObject”. We changed the way of thinking because we needed to keep both obstacles and lokums in the same data structure (BoardObject array in this case). Also, when candies crush, instead of making null entries in the array, we convert them to “EmptyObject” which is a subclass of “BoardObject” class. In addition, we seperated Special Lokums and Regular Lokums and created different classes for each type of lokum. This separation was required for modularity.

Secondly, we thought that each type of lokum should have its own crush method. After our discussion, we realized that special lokum combos would be a problem with this design. Then, we got opinion from our TA, and decided to have a separate “GameEngine” class that will keep the game rules (crush rules, combo rules, score updates etc.).

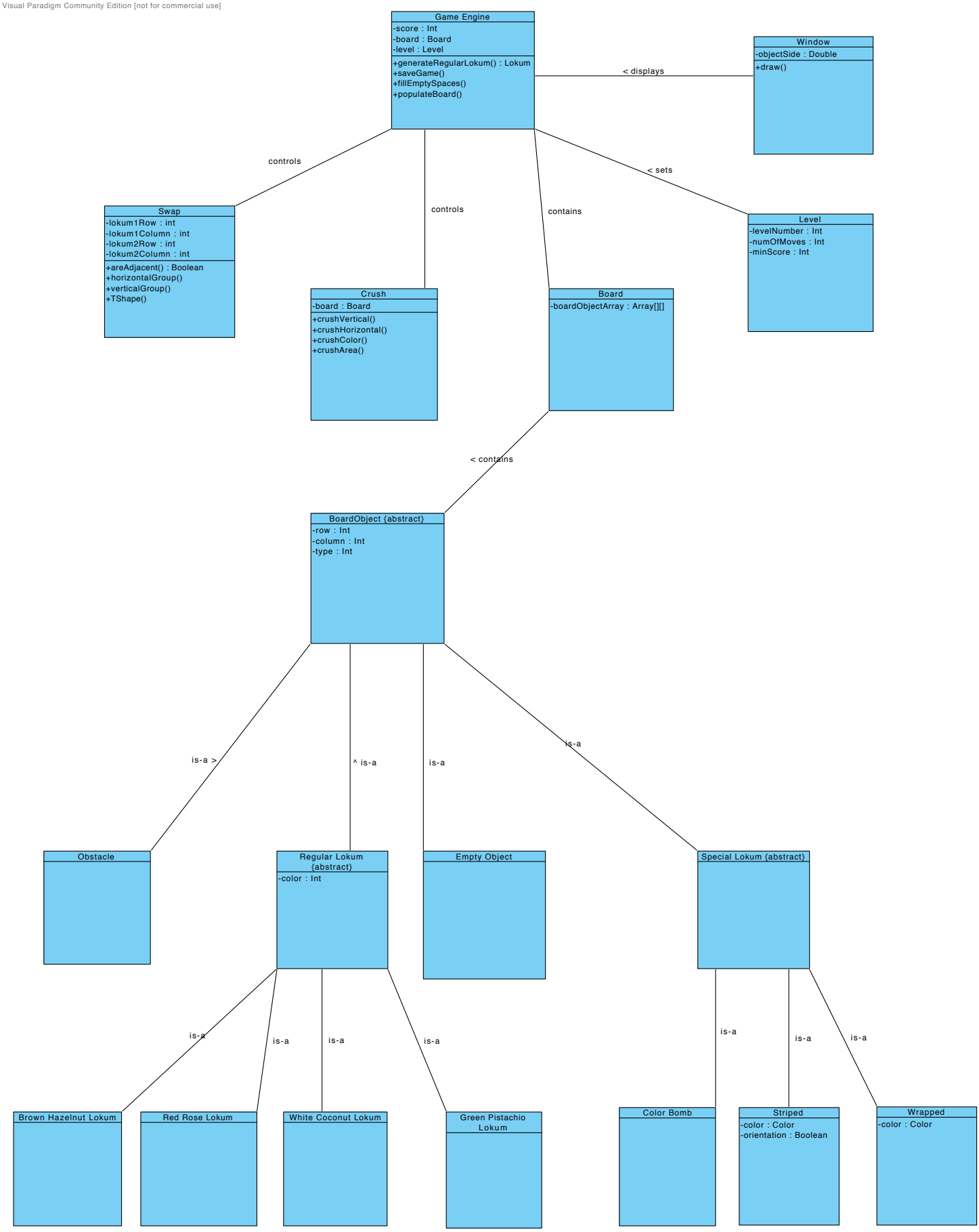
Lastly, we moved Game State to the “GameEngine” class while transitioning from domain model to class design because GameEngine will be responsible for handling the game rules.

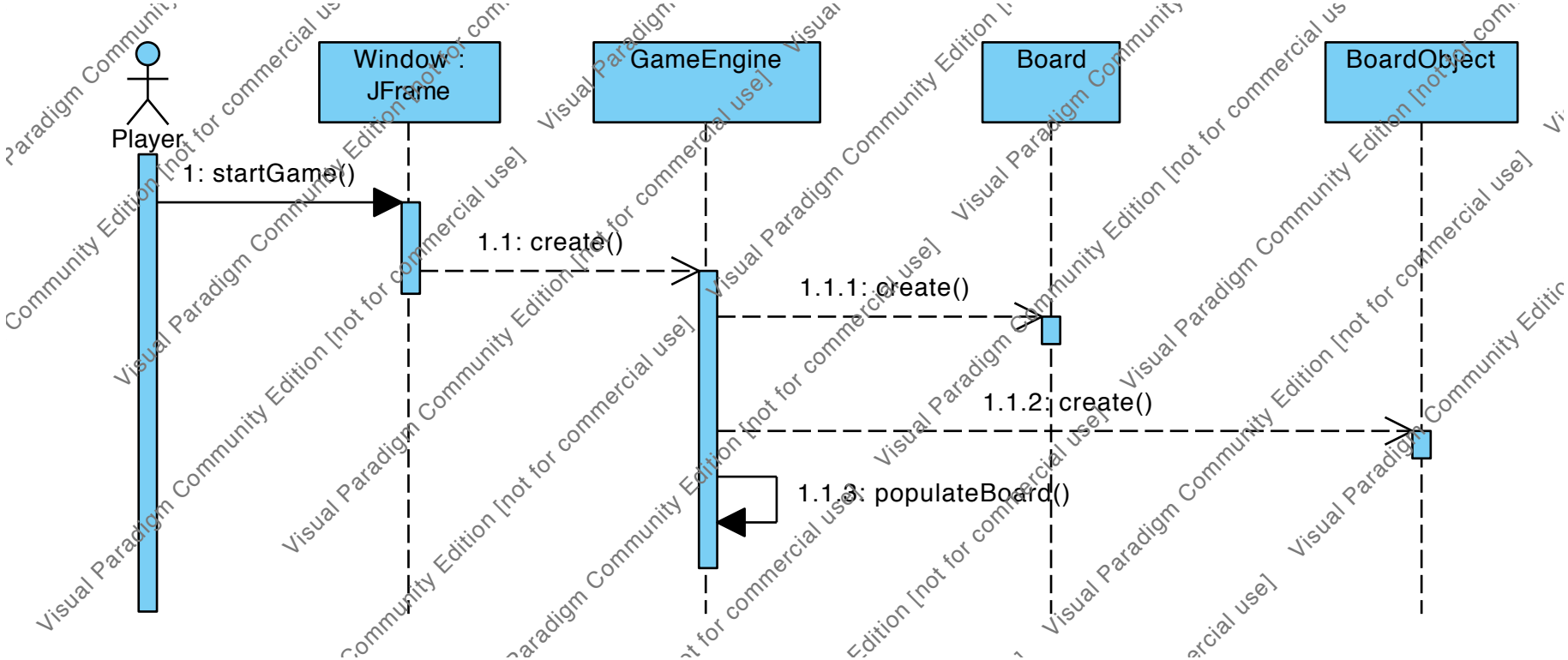


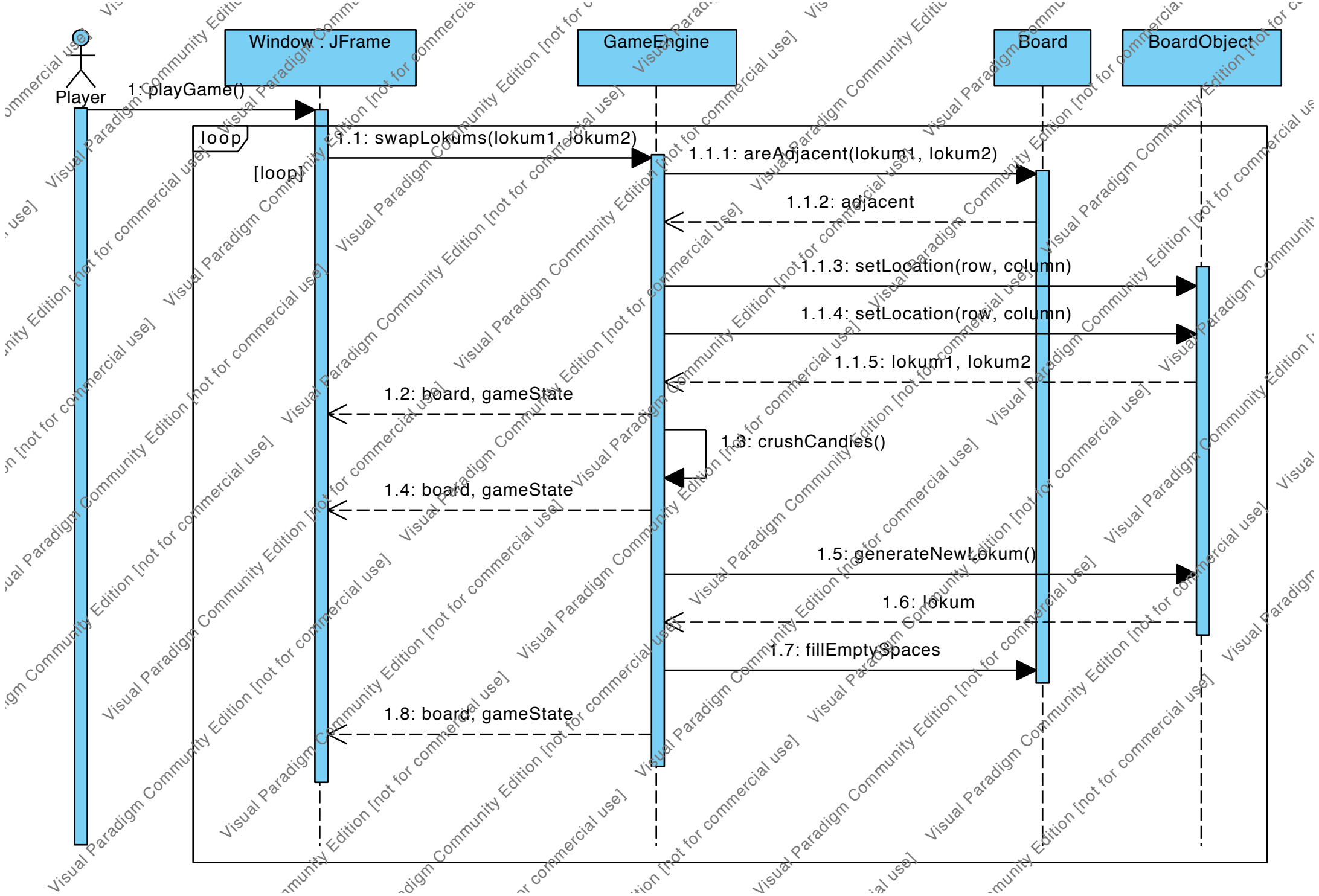
After the Project Meeting

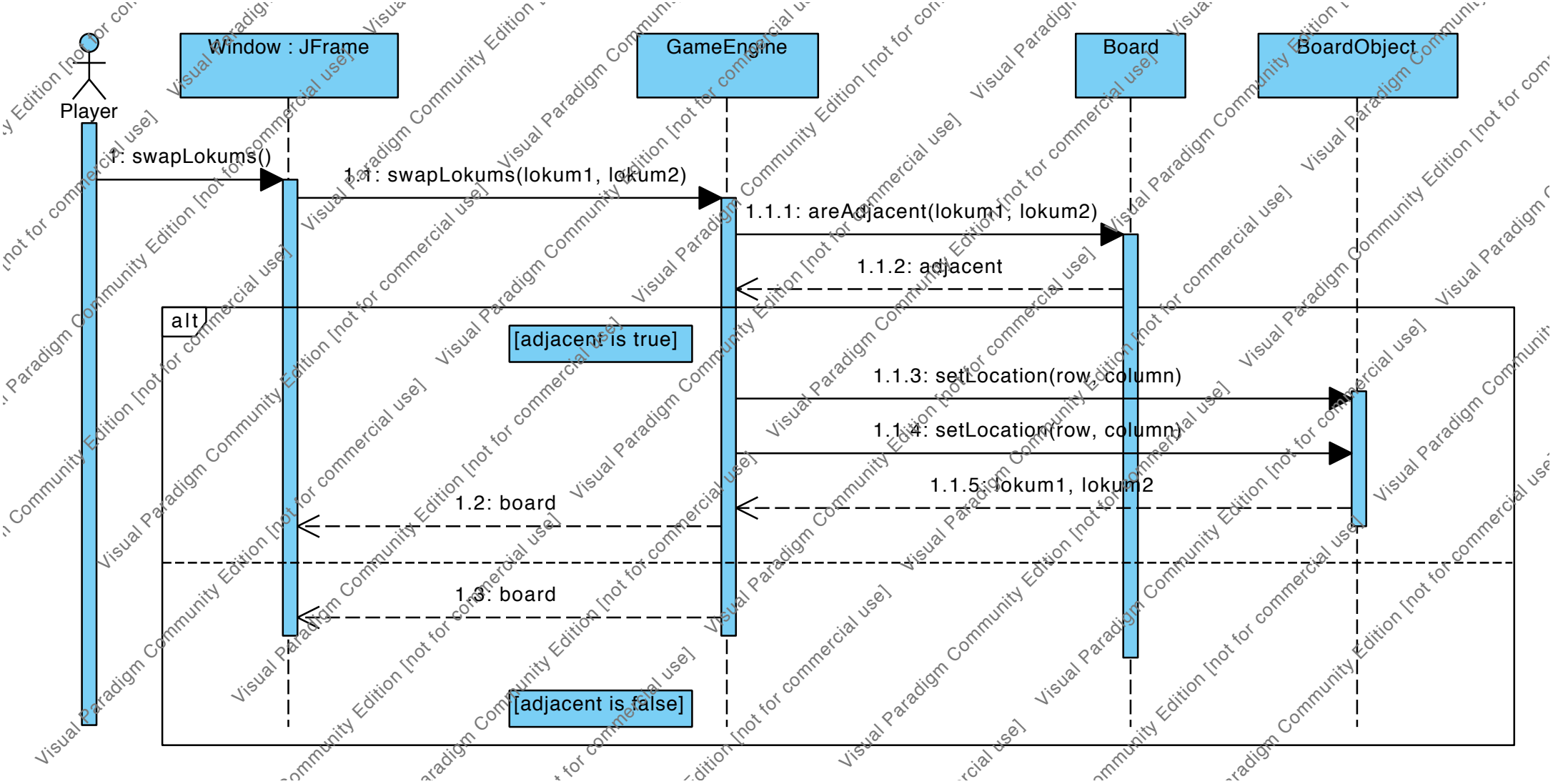
We used to have class diagram as seen in the previous page, but after the project meeting, we have made few changes in our class diagram considering our teaching assistant's guide. We have created two new classes for swap and crush, which were used to be functions of the Game Engine class. Swap function takes four integers as arguments to specify the location of the two lokums, and checks which type of lokum grouping is formed. Crush class takes Board as an argument and crushes the lokums on board according to specificied direction or color.

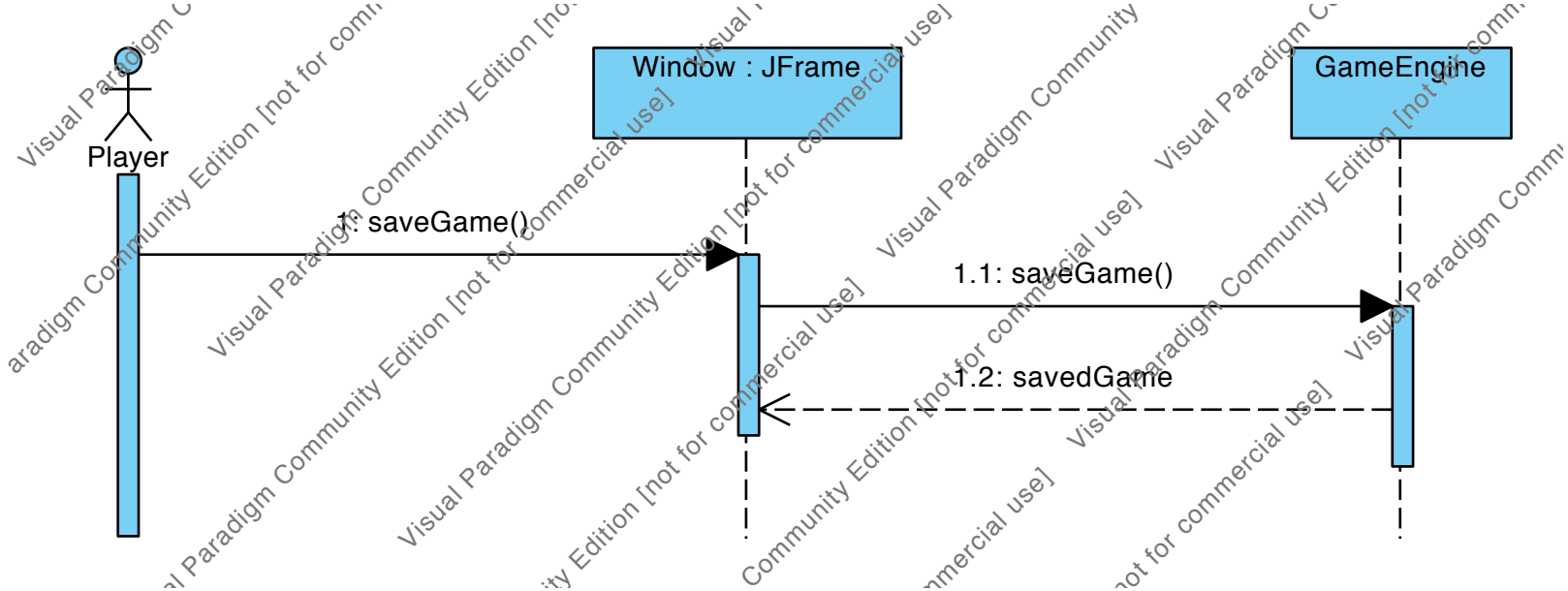
In this way, our Game Engine class will become much more modular and changes in the project will effect our methods less.




Player
Window : JFrame
GameEngine
Board
BoardObject
1: startGame()
1.1: create()
1.1.1: create()
1.1.2: create()
1.1.3: populateBoard()


Player
Window : JFrame
GameEngine
Board
BoardObject
1: playGame()
loop
[loop]
1.1: swapLokums(lokum1, lokum2)
1.1.1: areAdjacent(lokum1, lokum2)
1.1.2: adjacent
1.1.3: setLocation(row, column)
1.1.4: setLocation(row, column)
1.1.5: lokum1, lokum2
1.2: board, gameState
1.3: crushCandies()
1.4: board, gameState
1.5: generateNewLokum()
1.6: lokum
1.7: fillEmptySpaces
1.8: board, gameState


Player
Window : JFrame
GameEngine
Board
BoardObject
1: swapLokums()
1.1: swapLokums(lokum1, lokum2)
1.1.1: areAdjacent(lokum1, lokum2)
1.1.2: adjacent
alt
[adjacent is true]
1.1.3: setLocation(row, column)
1.1.4: setLocation(row, column)
1.1.5: lokum1, lokum2
1.2: board
1.3: board
[adjacent is false]


Player
Window : JFrame
GameEngine
1: saveGame()
1.1: saveGame()
1.2: savedGame