Refactoring Project Presentation

By
Eric Lin, Jack Corrigan, Moisés Lora, Doanh Pham, Jarrod Cummings, Abdulrahman Alfahad

GUI Refactoring

Description: Firstscreen and Secondscreen both had large amounts of duplicate code, along with a complex conditional statements. In addition, both classes had nested try catch exceptions which reduced code readability.

Evidence: Duplicate Code, Blob Method, Cyclomatic Complexity

Refactorings: Replace exception with test, decompose conditional, extract method

Rules Simplification

Description: A simplification of the Rules class, with emphasis on reducing complexity and length of methods.

Evidence: Multiple several hundred line long methods, complex nested conditional statements, repeated code.

Refactorings: Extract methods, decompose conditionals

Checker Piece Decoration

Description: Placed movement responsibilities in checker piece classes. Pieces can be dealt with generically. The methods in King build off of those in Single. Also overlapped with simplifying Rules class.

Evidence: The initial Piece classes did not really serve any purpose beyond holding color and type. There was a lot of type checking in Rules.

Refactorings: Piece classes now handle finding possible moves. King acts as a wrapper for a Single. Complexity of methods in Rules class reduced.

State Refactoring

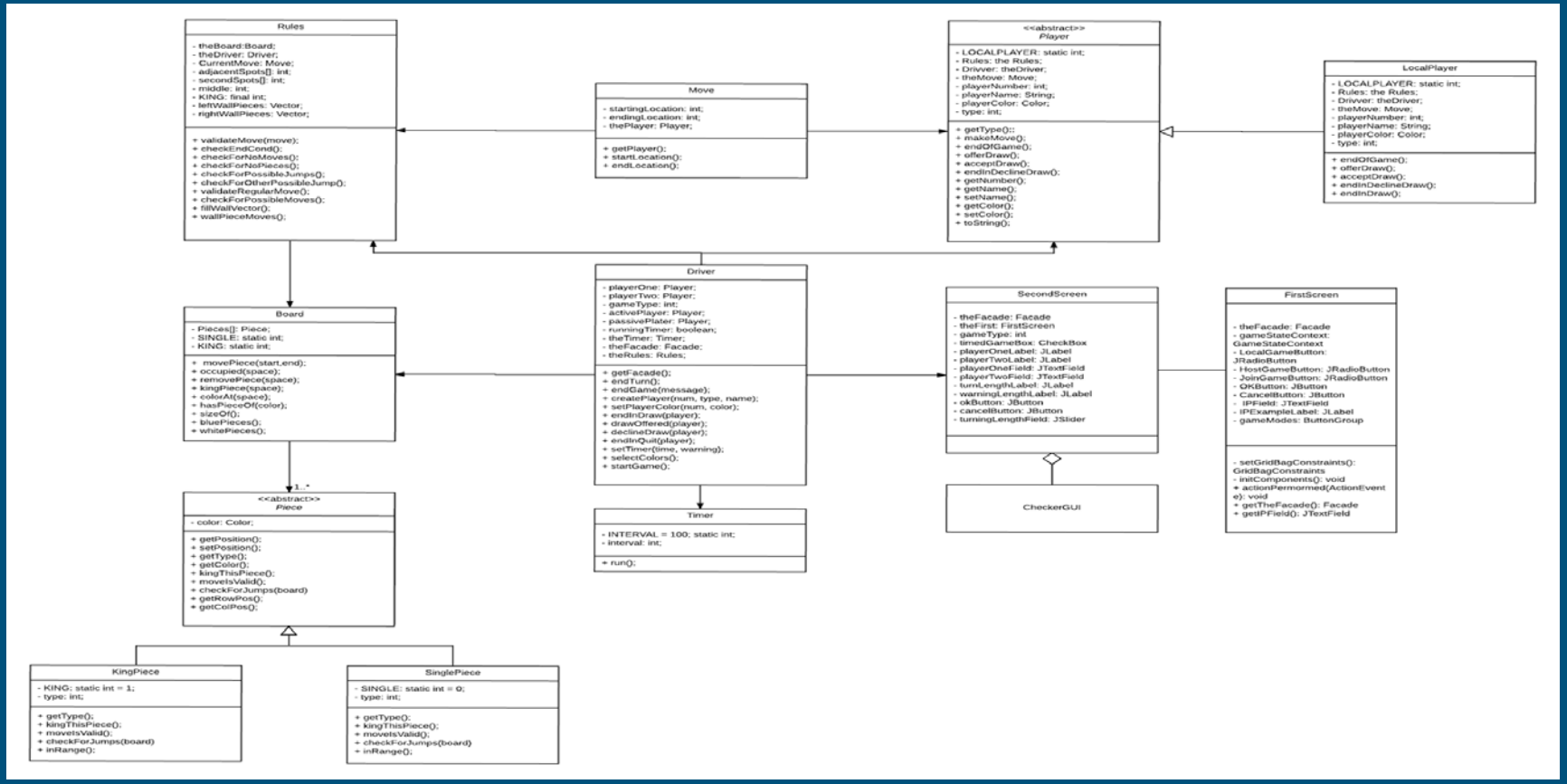
Description: Firstscreen's actionListener() method contained a large conditional that handled all the functionality of each game mode. This was wrapped in a large try catch block that caught generic exceptions thrown by any of the game modes

Visual Evidence: Nested try catch, large conditional

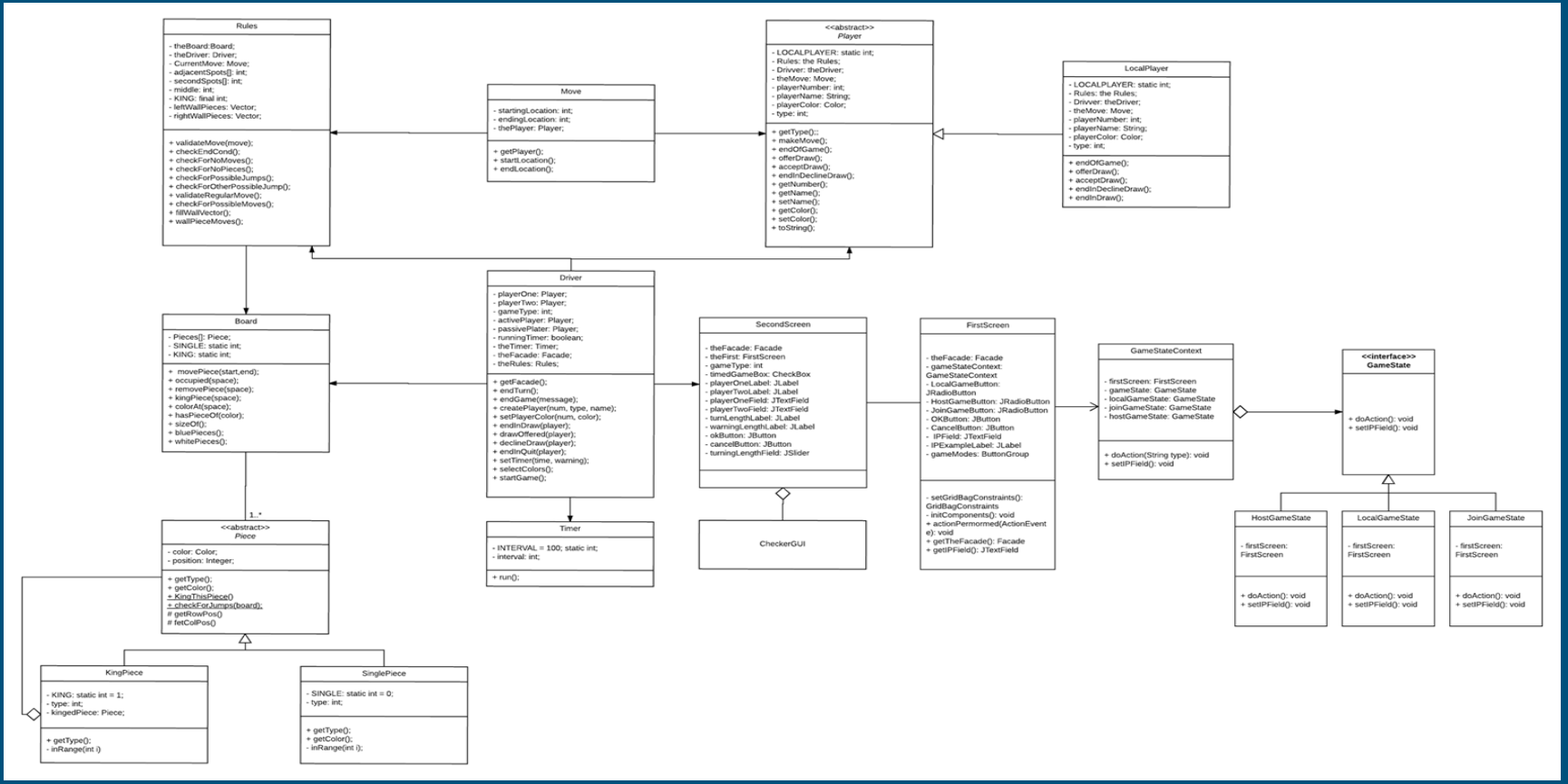
Refactoring: Replace conditional with strategy (in this case, state)

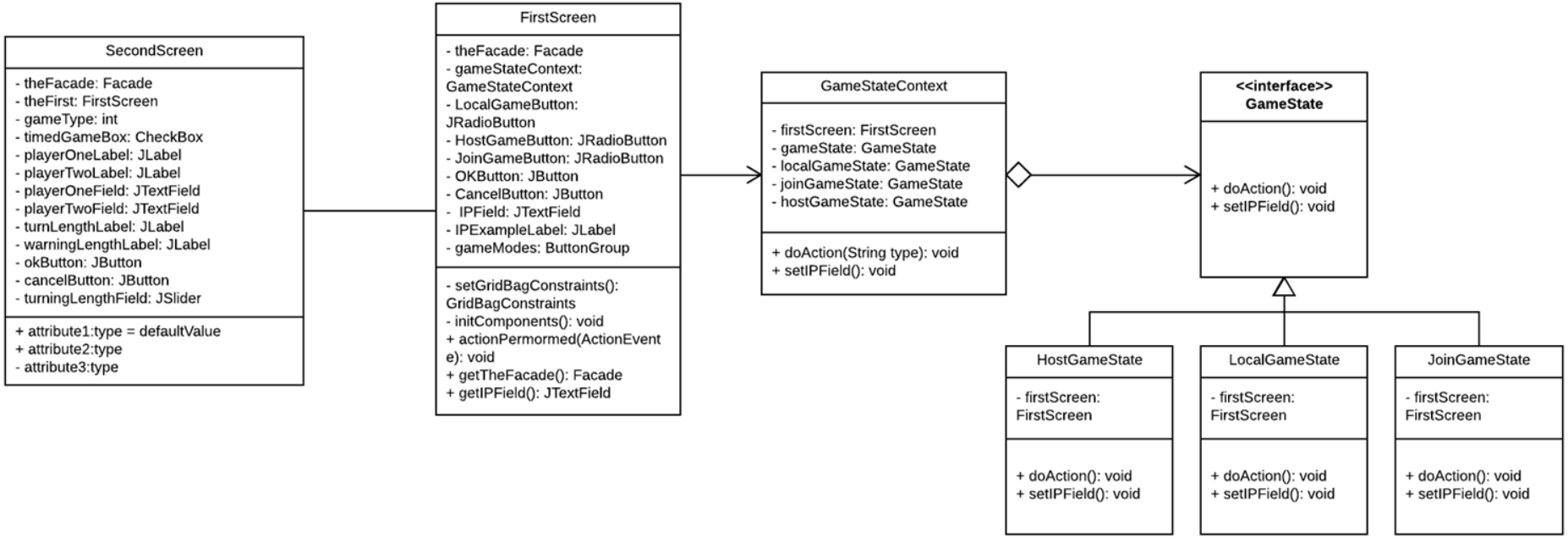
- Each game mode is a new state, handling its own functionality

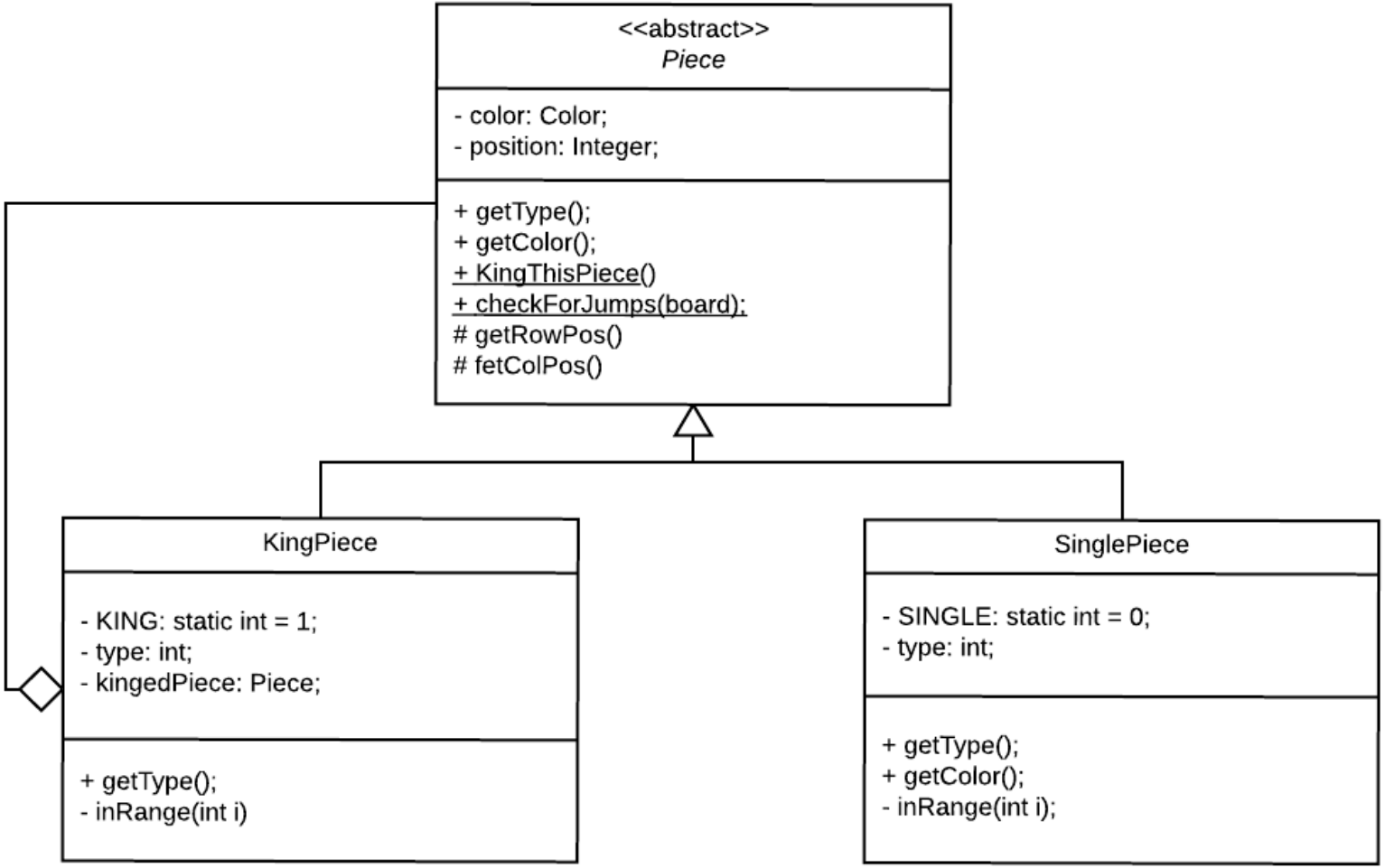
Class Diagram - Old



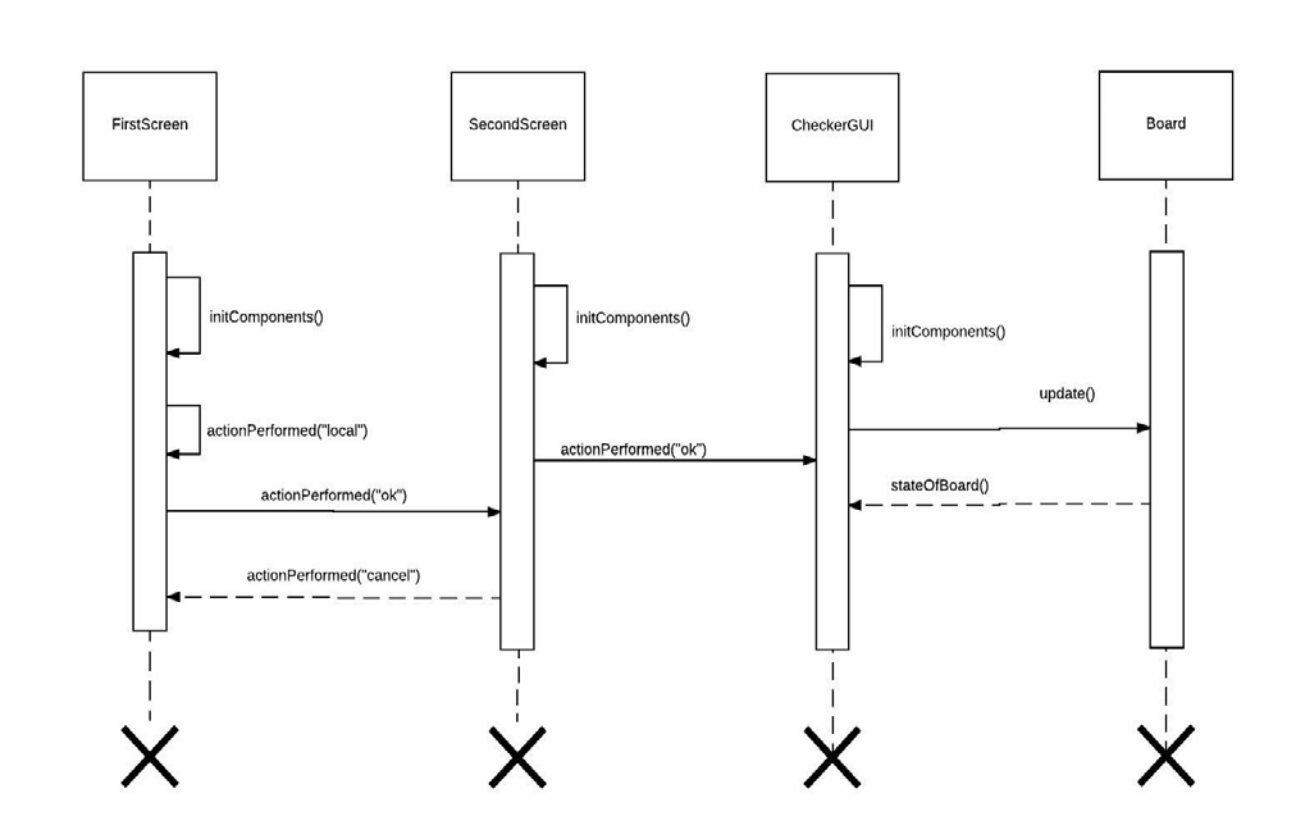
Class Diagram - New



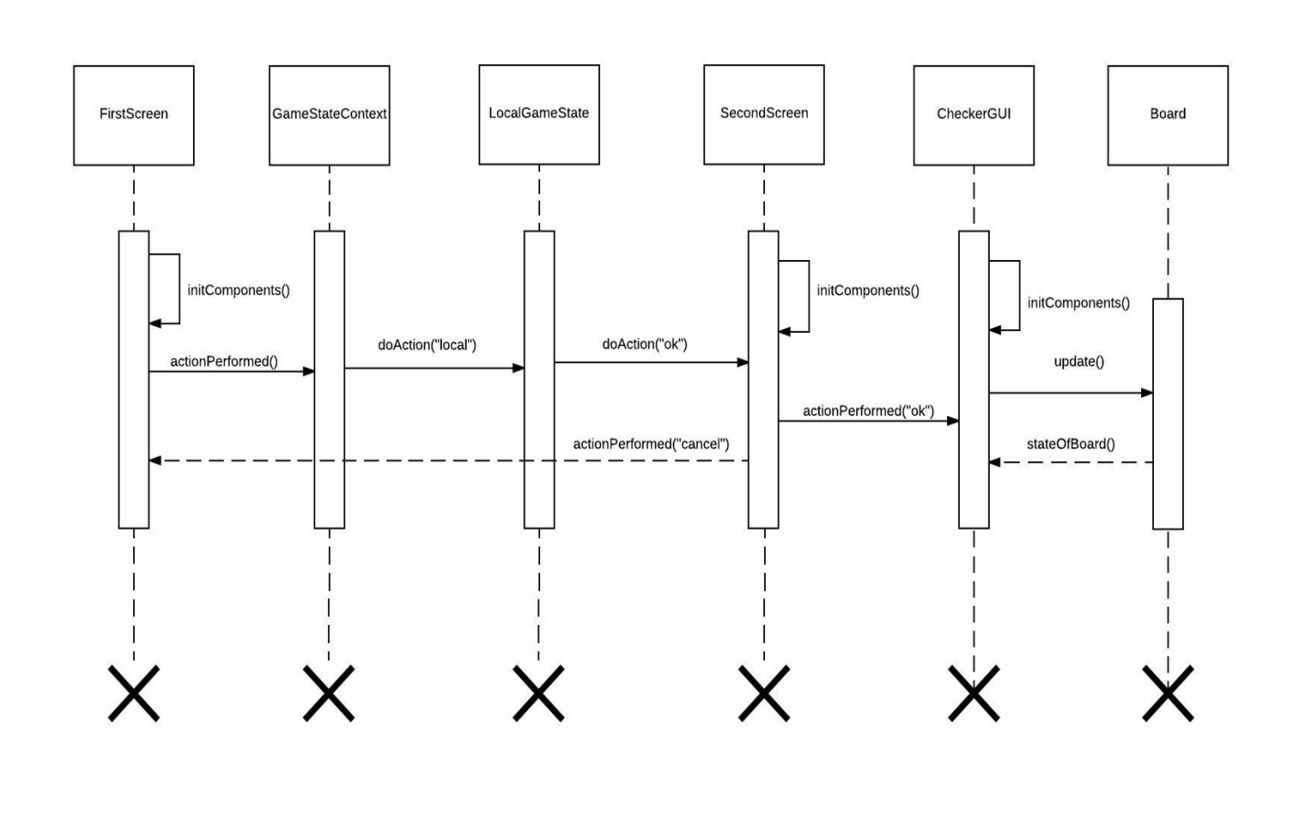

SecondScreen
- theFacade: Facade
- theFirst: FirstScreen
- gameType: int
- timedGameBox: CheckBox
- playerOneLabel: JLabel
- playerTwoLabel: JLabel
- playerOneField: JTextField
- playerTwoField: JTextField
- turnLengthLabel: JLabel
- warningLengthLabel: JLabel
- okButton: JButton
- cancelButton: JButton
- turningLengthField: JSlider
+ attribute1:type = defaultValue
+ attribute2:type
- attribute3:type
FirstScreen
- theFacade: Facade
- gameStateContext: GameStateContext
- LocalGameButton: JRadioButton
- HostGameButton: JRadioButton
- JoinGameButton: JRadioButton
- OKButton: JButton
- CancelButton: JButton
- IPField: JTextField
- IPExampleLabel: JLabel
- gameModes: ButtonGroup
- setGridBagConstraints(): GridBagConstraints
- initComponents(): void
+ actionPermormed(ActionEvent e): void
+ getTheFacade(): Facade
+ getIPField(): JTextField
GameStateContext
- firstScreen: FirstScreen
- gameState: GameState
- localGameState: GameState
- joinGameState: GameState
- hostGameState: GameState
+ doAction(String type): void
+ setIPField(): void
<<interface>>
GameState
+ doAction(): void
+ setIPField(): void
HostGameState
- firstScreen: FirstScreen
+ doAction(): void
+ setIPField(): void
LocalGameState
- firstScreen: FirstScreen
+ doAction(): void
+ setIPField(): void
JoinGameState
- firstScreen: FirstScreen
+ doAction(): void
+ setIPField(): void


<<abstract>>
Piece
- color: Color;
- position: Integer;
+ getType();
+ getColor();
+ KingThisPiece()
+ checkForJumps(board);
# getRowPos()
# fetColPos()
KingPiece
- KING: static int = 1;
- type: int;
- kingedPiece: Piece;
+ getType();
- inRange(int i)
SinglePiece
- SINGLE: static int = 0;
- type: int;
+ getType();
+ getColor();
- inRange(int i);

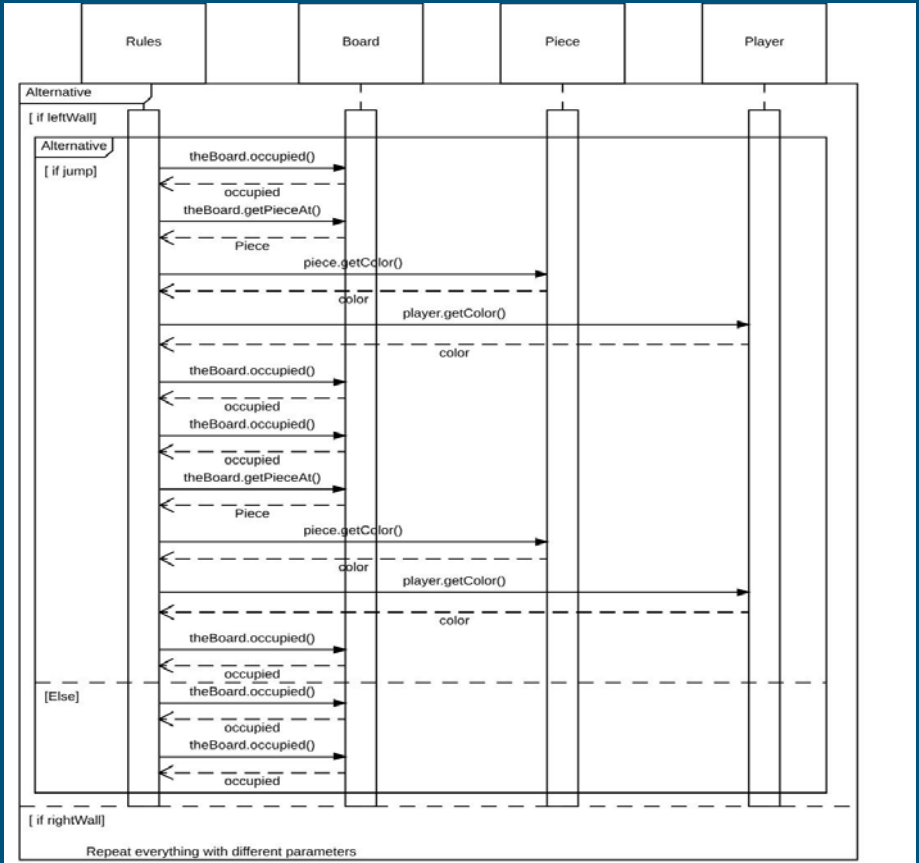
Sequence Diagram 1- Old



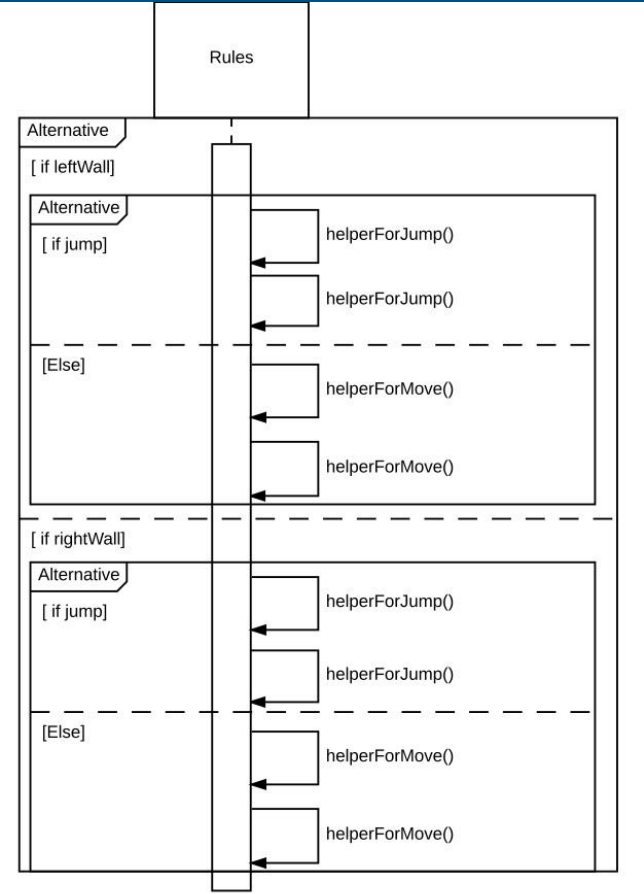
Sequence Diagram 1- New



King Piece Wall Move Sequence Diagram - Old



King Piece Wall Move Sequence Diagram - New



Metrics

1. Cyclic Dependencies: Rules decreased from 27 -> 3 (89% decrease)
2. Average Coupling Between Objects: (Project) 4.74 -> 5.74 (21% increase)
3. Lines of Code: 5444 -> 4466 (18% decrease)