Alex Tzinov
Kyle Wiese
Jacob Brauchler

Checkers Palooza Final Report

Implemented Features:

- Smart AI that follows all rules, looks ahead to find best move, and can save pieces that are currently in danger of being jumped
- User interactivity through a GUI
- Mouse interactivity
- Checks to make sure user can only make legal moves
- Highlight selected piece

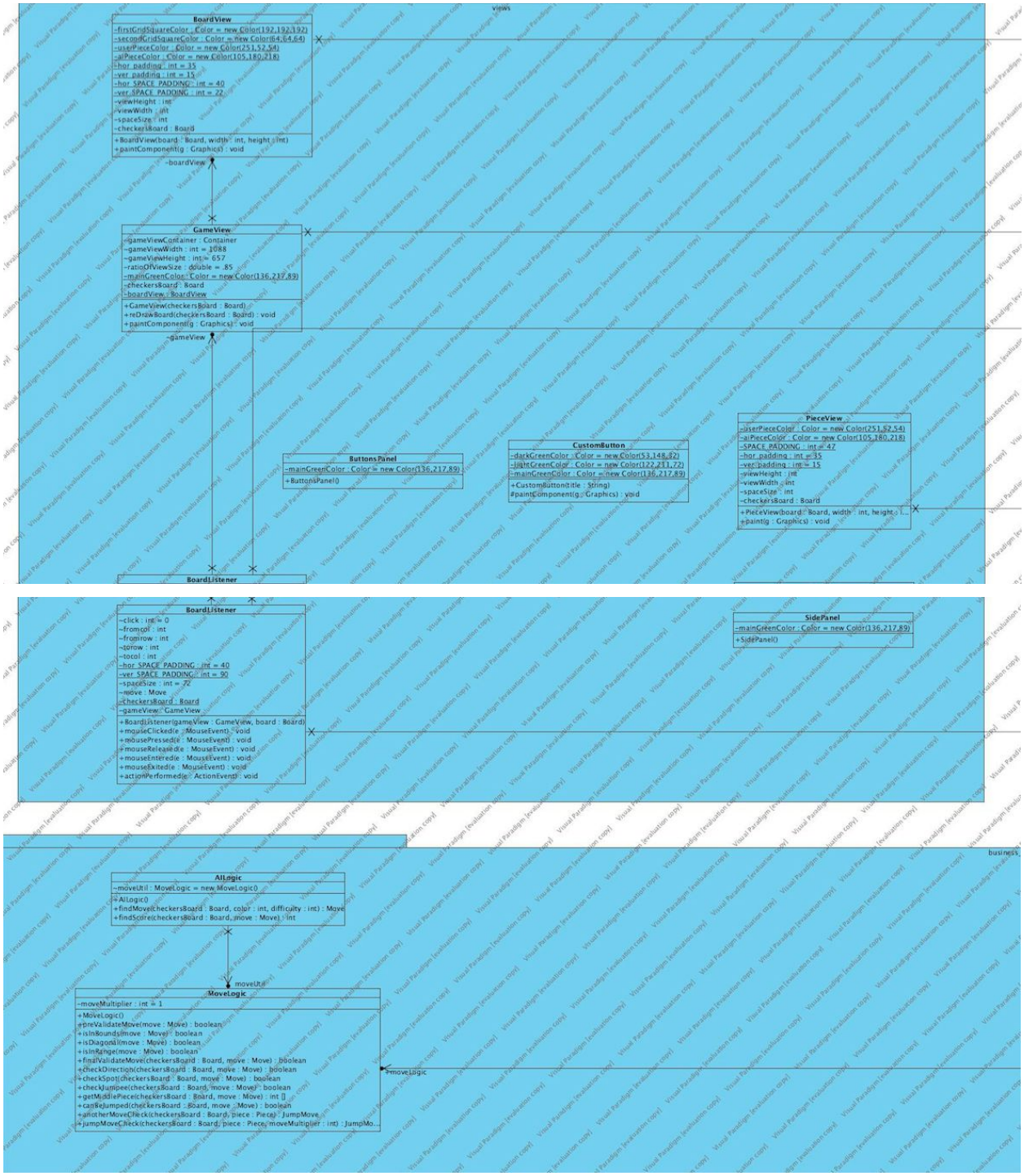
Unimplemented Features:

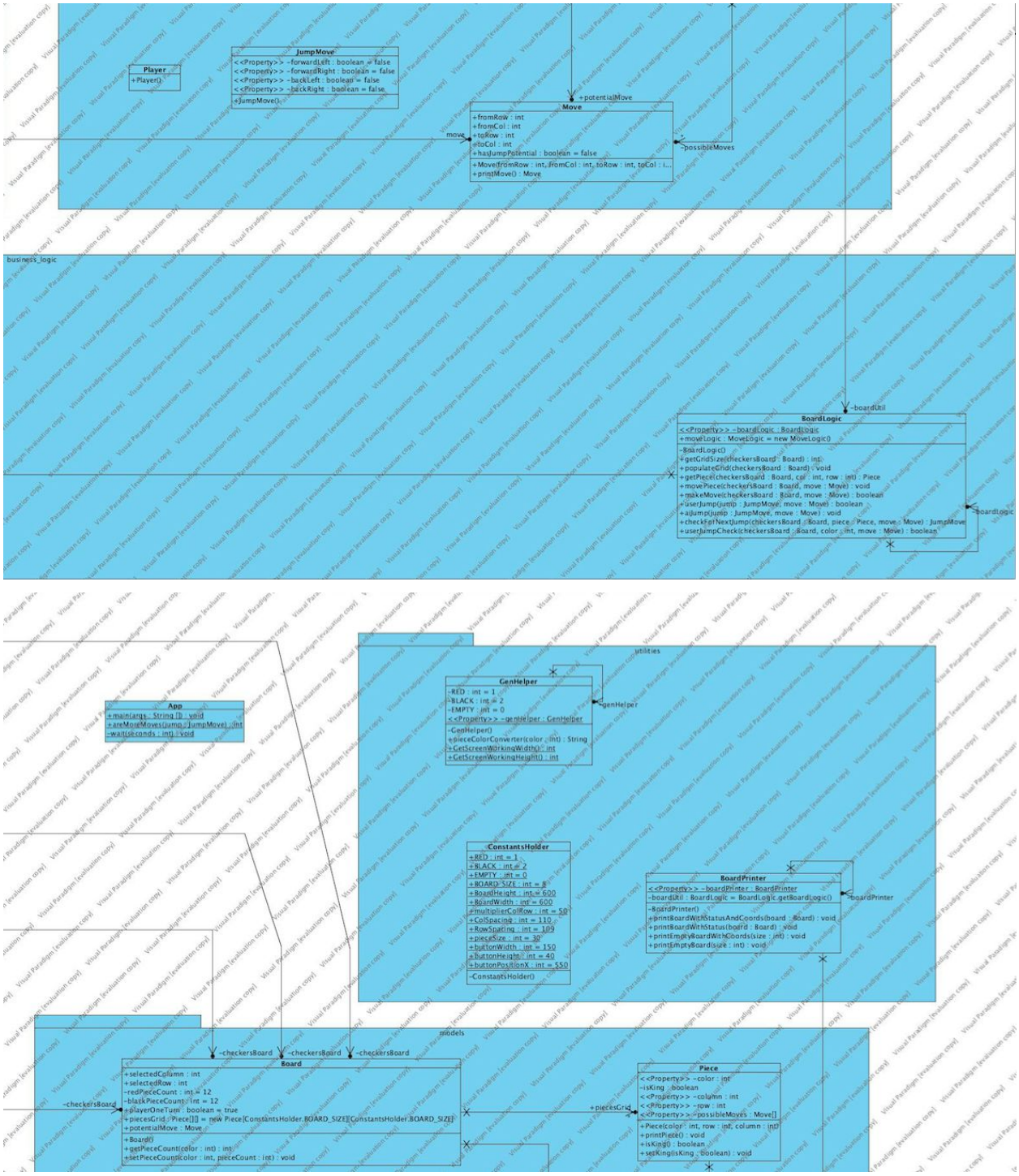
- Menus and menu options
    - Save/Continue Game
    - Pause Game
    - Resume Game
    - Go to Main Menu
- Suggest Move (Logic is present, but not implemented in gui)
- The ability to see available moves for a selected piece (Logic is present, but not implemented in gui)
- Varying Levels of AI (Logic implemented, but not gui)

Previous Class Diagram

Final Class Diagram:

(Better View in Repo
https://github.com/jabr9983/OOADProject/blob/master/Checkers_FinalClassDiagram.pdf)





Differences and Reasoning:

- We ended up using a MVC architectural pattern that we did not plan for when first creating our class diagram so it changes most over arching designs.
- We did not end up adding the database in order to save our game so there is no database class in our final class diagram.
- We didn't have a Game class instead we used and App class, that would be our driver.

- We added a lot of utility classes that were not planned for because it wasn't until implementation that we saw the need for separate utility classes.
- Since we didn't end up creating the pause and main menus we didn't have classes for them and only had the buttons to call them.
- The Swing components weren't thought of until later in the project.

How the Pre-Planning Helped:

- Even though we changed our overall class schema, project part 2 allowed us to consider the interconnecting parts of the application. Thinking about different use cases that the user could encounter also turned out to be extremely useful in the end. By considering the use cases, we were able to come up with an idea of how processes should be carried out and what the user should be able to interact with. The use cases also allowed us to be able to come up with more possible features that were/would be useful to implement.

What Pattern(s) Were Used:

- MVC we used a pseudo MVC architectural pattern in order to build our Checkers App
    - Views were used to draw the board, pieces, and gather user input which was then passed into a driver class. This driver class acts in a similar way to a controller in that it gathers input from the view and interacts with the correct models and business logic. The main thing that differed from the normal MVC layout was that our models didn't directly interact with the views. This aspect could be refactored in the future to better represent the MVC architectural pattern.
- We used Singleton for our board utility and helper class in order to force that a single class instance was used across the application.

What Pattern(s) Could be Used

- If the game were to be moved to a higher scale (like a online server running the game), Flyweight could be used to draw the different pieces more efficiently
- If more time were available and the menus were implemented, the state design pattern could have been used to deal with user input throughout the menu selections
- Decorator could have been used to add more elements to the game window and make a more user friendly window (i.e scrollable and resizable)
- Iterator could have been used to hide the board implementation when accessing multiple pieces
- If we were to implement an "undo move" functionality, the memento design pattern could have been utilized to save the previous state of the board
- Observer could be used in order to make the asynchronous mouselistener activity integrate better with the rest of the application

What We've Learned:

- We learned that the coding process works easier and smoother if you plan out which design patterns you will use before trying to code. This makes it so you don't have to go back and do as much refactoring in order to include design patterns later
- Communication with group members is key
- Focusing on making reusable functions is a huge help later in the application development process
- Utilizing objects to store information helps the code remain organized and easy to use
- As design decisions come up in the actual implementation, it's okay to not follow what we originally decided in the planning stage, or in other words, Agile is a good methodology to follow
- Sometimes a project can be bigger than expected, so understanding your time limits is an important step for future projects. For instance, we underestimated the problems we were going to encounter with implementing the GUI, which in turn caused certain aspects of this project to take a back seat to more crucial features

Note: We didn't realize that Jacob Brauchler was not showing up as a contributor on github until the end but if you look in the commit history he indeed was.