This artifact is a chess game based on Java and JPanel principles. The initial game was very simple, abiding by the basic rules of chess (no advanced rules / moves) and only allowing for two-player matches (no AI). It was created last Winter during a Java course at Grand Valley State University. This project was initially a test of endurance that made us implement a large number of different classes and features while juggling a pre-created UI that had very little documentation. Working with this allowed me to understand more about this UI while also refining the features of the project.

Chess is a game that holds a large amount of potential for automation. From simple rule-based AI to deep learning arrays with massive numbers of epochs, automating chess has always been an interesting facet of computer science to delve into. This project gave me a great opportunity to fulfill the algorithms milestone while looking into a part of chess that I find interesting. It also already had a UI that was unintuitive to work with. This allowed me to learn more about JPanel while also coming up with a creative solution for displaying turn data. Overall, the artifact was improved via the inclusion of additional on-screen data (turn count was used over a timer due to the display not being updated live) alongside the addition of a MinMax Chess AI. This shows the player a bit more about the game while also allowing for single-player matches.

The two-course objectives I listed and set out to achieve for this artifact were 3 and 4 (Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices (data structures and algorithms / Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals (software engineering/design/database)).

Objective three was met via my work with both of the new features. The new UI information system makes use of a limited doubly linked list that removes values from the head as more values are added to the tail. This gives the impression that the turn history is scrolling upward as it's updated. This has the upside of being memory efficient and being very easy to work with at the cost of losing access to any data that’s outside of the size limit. I was initially using an ArrayList and printing out the most recent 10 values added, but I decided this more compact option felt more suited for the task due to me not using many of the features of ArrayLists.

Alongside this, I chose to pursue a single depth (one turn per player) Minmax algorithm for the AI. This has the upside of providing an opponent who actually seems to weigh the pros and cons of options and acts based on them. The main downside posed is that once you understand the AI, its shortsightedness is very apparent. This can be amended either via making use of techniques to trim the number of branches explored by increasing the depth or by providing more processing power as depth increases.

As for objective four, I focused on targeting efficient and modern techniques for both goals. The Minmax algorithm is still used today, and ways of trimming down its processing time are still being looked into. I implemented a very simple form of it, but it has a massive amount of potential for improvement. Alongside this, I made use of a linked list for the UI information due to its fast insertion times and its memory efficiency. This made it seem like a prime choice for the task. Alongside this, I have begun documenting this project for my final submission, which will include code and Word documentation to allow outside parties to work with this project.

The big two things I learned while working on this article are related to JPanel and MinMax computing. JPanel is an interesting setup for creating UIs in Java that I had not worked with a massive amount before. I now know that it seems like a great option for implementing UIs that are detached from the logic of the class it's displaying info from. On top of this, I learned a large amount about how to implement and improve MinMax algorithms.

The main challenges I faced had to do with my time away from the project and uncertainty about the tools I had already generated. The UI was challenging to work with due to its lack of documentation, and I found myself wondering how I wished to grab the number of each piece per player. The interesting part about both of these challenges is that they would both be avoided with better documentation. This is why I plan to release Word docs detailing commands and use cases in my final submissions.