**Chess Royale Final Report Summary**

**Group 5: Dan Hrubec, Julian Gonzales, Joseph Canning, Victor Fong**

Chess royal is a combination of the classical game of chess, blended with the battle royale style of games. In the game, you will be loaded in with 3 other AI controlled players to be the last man standing. After each player makes the moves twice, completing two full rotations of player’s moves, the board begins to shrink in creating a red ring around the outer edge that increases in size. To aid in player’s strategies, there is the addition of green and red tiles scattered throughout the map. If you were to land on a green tile, it will allow you to upgrade your chess piece to the next best available piece. Landing on a red tile will downgrade your chess piece, if you are already the lowest tier chess piece being the king, you will be eliminated from the game. The project was created using the Unity game engine and was coded with the usage of C# script files that control the game flow, as well as all of the different interactions within the game. To test it, we have created various tests to make sure that all aspects of the game are functional, and the interactions within each of the script files are accounted for.

This project was originally proposed by group 15 of the Fall 2020 semester. Many of the features mentioned within the original report has been implemented in the game, but some features had to be cut for a full development of the game. Our project prototype did remain faithful for many aspects and features of the game but could not be accurate for every game feature. Some of those features that had to be cut due to different constraints include a full 8 player version of the game, a full multiplayer aspect with a client-server architecture, as well as other interactions with the green tiles. A lot of the core features have been implemented within the game including the map shrinkage to keep in the flavor of the battle royale element of the game, upgrading/downgrading pieces, as well as piece capturing. We did add other features that were not originally mentioned in the original proposal being the difficulty selection. We had added this feature to diversify the game experience, making it more enjoyable for the potential clients. Having this feature allowed us to change a few aspects of the game including the board size, the tile spawning rate, as well as the speed in which the map begins to shrink it.

The development of Chess Royale was split into three different release, with four sprints for each of the releases, allowing for four weeks of development for each release. In each of the releases we would practice different software development methodologies, and experiment with different ones to see which worked well compared to others. In the first release, we needed to create a strong structure to build the game off of, in which we created a underlying grid system that would account for each tile and allow for the players to play on top of. We added in the red and green tiles to be randomly dispersed across the map but left the interactions with them for future a future release. We implemented some basic movement and loaded in the first chess piece being the king. In this release the development methodologies we focused on was a blend of the Scrum methodology as well as Kanban. We would lay out all of our tasks and assign them across sprints. Once each task was completed, it would be moved to an in-review section, where a different group member would then review the code and approve of it, closing out the item. In the second release of the project, we had a strong structure to build upon, and focused on many of the interactions that would have happen during gameplay. So we added the functionality with the red and green tiles, we loaded in other players to have them play as simple AI, implemented the board shrinking, a difficulty selection, and allowing the player to capture others. During the development of the second release, we had a strong focus on XP, with the practice of pair programming working the best, as it was easy to communicate and spectate the coder via Discord and the screen sharing option it provides. In the last release, we primarily focused on created an end game scene as well as many bug fixes that might have occurred in previously developments. We would focus on the development methodology of TDD, but it was difficult for us to use Unity’s testing framework and NUnit testing, making it an obstacle for our development, and something to investigate before further development.

For the testing of the project, we decided to do more physical testing of the game by primarily play testing the game to ensure that our code was functioning, and our script files were taking every interaction possible into account. This was mainly as we had difficulties setting up the Unity testing framework and NUnit testing. The testing of the code took place over three days and three group members working on each of the tests fully detailed in the final report document. In this most of the tests ended up passing with an exception to one due to a known bug in the game. The bug in the game allows the AI players to sometimes make a move twice in a row before going to the next player. This affected the test in the board shrinking as it messed with the calculations of the player moves, shrinking the board earlier than intended.

The inspection of the code was a physical review of the code to make sure that it was readable, had good structure, and was fully functioning. We collaborated over discord to make this happen and used an inspection guideline for the University of Toronto. We would first individually inspect each of the files for the consistency and took note as to different areas of the code that could use more work or be review in closer detail. We then would discuss with each other our findings and work out potential solutions to fix it, if any problems had occurred. An overall theme that needed improvement for our code was documentation. Little to no comments on the function for each variable and methods created. So, we would need to go back and add precondition and postcondition requirements before each of the function declarations. One file did not pass our inspection, as it was messy and disorganized being the MoveToGrid.cs file. A full refactoring and overhaul will be needed on that specific file before any further developments are made to prevent any issues arising around it.

Lastly, as this was a simple prototype of the game, we discussed some of the bugs that arose that are still in the current prototype as well as talk about any further developments that we can make to the project. Some of the open issues that arose is that the scoreboard in the main game is not dynamic to different screen sizes and is at a static point on the screen. We would like to be able to resolve this in the future by allowing the player to select their resolution and adjust the display according to the player’s selected resolution. The known bug previously mentioned will need to be resolved as it takes away from the gameplay, where the AI player would sometimes make their move twice in a row. Other open items are left as further developments. To stay true to the original design document, we would want to implement all the features that had to be cut for the time being. With the major items being the multiplayer aspect and allowing for a client-server architecture, to allow you to play the game with friends. Secondly, being the max allowed players be 8. In the current version, it only supports 4 players, and we would like to extend this to allow for any number from 2-8 players to play at any given time.