**Name of the game: <Wizard’s Chess (Regular Chess with Griffindor Colors)>**

**This is a template. Please edit it as you see fit; you may add pages at the end. Hand-drawn diagrams are acceptable. Attach Javadoc pages, if any, at the end.**

**1. Program Concept**

<Describe the overall concept of your game>

This is a classic version of chess that follows the same rules. There are 6 different types of pieces: pawn, rook, knight, bishop, queen and king. Each unique piece has its own rules to follow. The pawn may only move one block straight ahead unless it is the first move in which case it can move 2 spaces ahead. It can also only take another piece directly in front and diagonal to the left or right one block. Once it reaches the opponents baseline, it is able to transform into any piece. The rook is able to move an unlimited number of squares up or down, left or right as long as there is nothing in the way. The knight is able to move in a special way which is an L shape 2 spaces straight foreword, backwards, left or right and one to the left or right or up or down. The bishop has the ability to move in a diagonal line with the same rules as the rook to move an unlimited amount of spaces within the board and until blocked by another object. The queen is a combination of the rook and the bishop. The king is only allowed to move one space in any direction and cannot move into a checked location. The objective of the game is to attack the king and make it so that the king cannot run away and will die. The regular rules of chess apply.

<List the major functionalities that your program has, and briefly describe each of them>

|  |  |
| --- | --- |
| Functionality Name | Description |
| Regular Chess Rules | The computer is able to know the valid moves, check, checkmate, stalemate, and the basic functionalities of a normal chess game. |
| A GUI with pieces | The graphical user interface has pieces that you can click on and click on a location on the chessboard and it will move the piece if it is a valid location and move |
| Pop Up windows for pawn evolution | When the pawn reaches the other end of the board, it has the ability to evolve into whatever piece it wants to and we have a pop up window that lets you select what piece you would like to switch to |
| Pop up window for checkmate | If a |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**2. Code structure**

**2.1 Class diagram**

<Draw a class diagram of your program. Use proper notation so that your class hierarchy, interfaces, and class relationships are clearly identifiable>

**2.2 Briefly describe the overall structure of your code.**

<Describe your program structure with consideration of: MVC architecture, polymorphism, encapsulation of classes, and use of interfaces>

**2.3 What are the data structures you used? Why did you use those specific data structures?**

<Write the data structure you used, its purpose, and why you selected that data structure over others>

**2.4 What are the major algorithms used in your program?**

<Briefly describe the algorithm of you major functionality with pseudo code>

**3. Test**

**3.1 Unit tests**

<List the unit test cases for your program, and briefly describe the purpose of each of them.>

|  |  |
| --- | --- |
| Test Case Name | Purpose |
| <method under test> | <describe the purpose, input, and expected output> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**3.2 System tests**

<List the system test cases for your program, and briefly describe the purpose of each of them.>

|  |  |
| --- | --- |
| Test Case Name | Purpose |
| <functionality under test> | <describe the purpose, input, and expected output> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**4. Known bugs, missing features**

|  |  |
| --- | --- |
| Bug | Impact or effect |
| <Briefly identify your bug, issue, or missing feature> | <describe the impact of your bug on your game. If you have any idea of how it might be fixed, describe it here.> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |