AP Computer Science Final Project - README Template

Instructions:
The first step in creating an excellent APCS final project is to write up a README. At this stage, this README file acts as your **project proposal**. Once you've filled in all components, Shelby will read through it and suggest edits. Ultimately, you need a document that adequately describes your project idea and **we must agree on this plan**.

Have one member of your group **make a copy of this Google Doc**. Then, they should share it with all other members **and with Mr. Shelby** so that every group member has edit permissions, and Shelby can add comments on your ideas.

There's a lot of parts of this document that you might not have full answers for yet. Because you haven't written the program yet, it's difficult to think about the **instructions** or **which group members will do which parts**. Even though this is hard to think about, you must have something in these sections that acts as your current plan. However, during the course of the project, you'll **continuously update this document**. This means that you will not be *held* to exactly what you put here - components of this document can change (and it's pretty common!).

There is one exception: the **Features List** section. Once Shelby OKs your README, the Features List section **cannot be modified**. For this reason, it is most important that you get a solid idea of what you want to make and the primary features it will have *now*.

Talk with your group. Consider drawing some pictures of what you think your project might look like. Be precise. When you're ready, fill this out together. Each component in brackets below ([these things]) should be replaced with your ideas. Note that there are several sample READMEs posted on this assignment for you to use as guidance.

----------When README is finalized, remove everything above this line----------

Chess Royale

Authors: Pranav Gunhal, Nishil Anand, Haochen Feng
Revision: May 13, 2022

Introduction:

Arise soldier! Battle has begun! You are the army commander of an unnamed kingdom that is at war. Your job is to protect the king, who is guarded in his castle against the enemies, while also trying to capture the enemy's king to get them to surrender. You have at your disposal numerous

battalions of skilled warriors, each with their own abilities and needs. These warriors are loosely based on chess pieces, and move in a similar fashion.

When battle begins, you will be able to choose between a few warriors at a time. Once you select them, they will march out on their own. Be wary! Some warriors want to attack the other side only, while others want to hunt down and kill the enemy soldiers.

This game is played against another player, through a network (if you dare!) or on a local device. The enemy's goal is the same as yours: capture the king by destroying the castle. Good luck and may the best army win!

Instructions:

During a match, users will see a board, with two kings on either side. Each player also sees a set amount of "energy" that constantly replenishes at a certain rate, displayed as a bar next to the king. Lastly, players will see a selection of chess pieces. Chess pieces all have a set energy cost which differs according to their overall "power", and can only be placed if the player has enough energy. After being placed, the corresponding energy amount will be expended. Players will be able to place chess pieces on the board by clicking on the image of the desired piece, then clicking on a square on the board, placing the piece at that position. One player is able to place white pieces on the left half of the board, while the other player places black pieces on the right half of the board. Pieces on the board will then automatically move similar to how they move in chess, except adjusted accordingly to fit the game. Unlike in chess, where pieces die instantly when attacked, chess pieces in chess royale have a set amount of health and attack strength. Pieces will constantly try to move to the enemy side to attack the enemy king. The king in charge of white pieces is on the left side of the board, while the king in charge of black pieces is on the right side of the board. If an enemy piece moves within a certain range, however, the pieces will try to move closer to each other to attack. If a piece's health reaches zero, it will disappear from the board.

The properties of the pieces that are available to players as described in the features list will be described to the player in game through various means - mainly the cards displayed on either side of the board.

Features List (THE ONLY SECTION THAT CANNOT CHANGE LATER):

Must-have Features:

[These are features that we agree you will *definitely* have by the project due date. A good final project would have all of these completed. At least 5 are required. Each feature should be fully described (at least a few full sentences for each)]

- Players are able to place any piece (they are able to select from the currently implemented chess pieces and place them) according to current energy and energy cost
    - If players have enough "energy" (which increases every second or so, in a clash royale manner), they can place a certain piece by selecting the desired piece and clicking on

the tile on the board on which they wish the piece to be placed. The energy corresponding to the cost of the piece will then be expended.
    - Pieces placed by players will be on different sides, both being the “enemy” of each other
- Pieces each have a set health and attack statistic
    - Pieces have an attack statistic, which is the damage they do on one instance of attacking an enemy, reducing that enemy’s health by the attack amount. If a piece’s health is reduced to zero or below, it will be removed from the board in that instance.
- There is a game board on which pieces can move and interact
    - There will be an elongated grid of tiles, similar to a chessboard on which pieces can be placed.
- Winnable
    - The “king” has a set amount of health, which attacking pieces can decrease by attacking any tile on the rectangle representing the “king”. Whichever player decreases the enemy king’s health to zero first wins.pieces can damage the king by being on any point on the other team’s side of the board.
- As the game progresses, players place pieces on the board and existing pieces move freely
    - Pieces move similar to how they move in chess(elaborated later)
    - All pieces move simultaneously, one step at a time. At every step, the game will prompt both players to either place a piece or skip their turn(no networking implementation)
- Visual information about the match is available to the player
    - The board has a sideways orientation, with kings on the left and right side, one side for each of the players
        - Kings are represented by a rectangle with height spanning the entire side of the board. However, they are not part of the board, and pieces cannot be placed or moved onto them. On the rectangle is a fraction indicating the current health of the king
    - Further to the left and right side are images of the various chess pieces that players can click on and place arranged in a column
        - Next to each image is the piece’s cost in energy
    - Above these columns are numbers indicating the current energy each corresponding players have
    - Above each piece is a fraction indicating the current piece’s health statistic
- Pieces implemented: rook, queen, pawn (kinda)
- Pieces’ individual movement, attack pattern, and behavior
    - Behavior for all pieces
        - (Applies to all pieces except pawn) attacking and targeting
            - “Range” of a piece: all possible tiles they can attack if the target is on it.
            - If a piece does not already have a “target”, it constantly scans in a predetermined range around itself for enemy pieces. If it finds an enemy piece, it sets that enemy as the target(prioritizes closer enemies if multiple enemies enter scanning range, selects random if multiple enemies are the same distant)

            - If a piece has a target, it takes the best moves, or the move that will close the most distance(if multiple moves close the same distance to the enemy, a random move is selected) until it comes within range of the enemy(maximum range).
            - A piece will not change target until the target dies or exits the “scanning range”
            - If a piece has a target, it will only attack the target. It can only damage other pieces if its attack on the target also happens to hit those pieces(area damage)
        - Pawn can only move forward towards the enemy king and automatically attacks any pieces(elaborated later)
        - A piece, if it does not have a “target” will constantly take optimal moves to the enemy’s side. It will then continuously attack the king
            - (king counts as a “target” when the piece is attacking it)
        - In each “move” of a piece, a piece first physically moves according to their own pattern, then attacks the target if it is able to. A piece will not move physically if it is already in range of the target
        - Pieces that move multiple spaces linearly can cut their move short(for example, a piece that can move a max of 3 tiles left can also move 1 or 2 tiles) if a shorter move already accomplishes the “goal” of the move(such as moving within range)
        - Pieces, except for Knight, cannot move when blocked by another piece in its path. If it is blocked, it will still select the move that will bring it closest to the target. In this situation, it can also move less than its maximum range if that is the optimal move.
        - Pieces cannot move past the boundaries of the board.
        - Pieces cannot move on top of each other
    - Individual piece movement and attack
        - Pawn
            - Movement: can only move one move at a time towards the enemy side
            - Attack: when called to attack, the pawn damages any enemies on the tile to the front right and front left of it
        - Bishop
            - Movement: can move up to a set number of tiles in any diagonal direction
            - Attack: when called to attack, if the target is adjacent to it(including diagonal) it will attack it
        - Knight
            - Movement: moves 3 tiles up and 1 tile 90 degrees of the 3 tile movement, in an “L” shape
                - Can “jump over” or ignores any troops in its path except for the one at the end

        - Attack: when called to attack, if the target is within 3 tiles to the knight(within a 5 by 5 square of tiles with the knight at the center) it will attack the target
    - Rook
        - Movement: moves a set number of tiles up, down, left, or right
        - Attack: when called to attack, damages all enemies adjacent to it(including diagonals)
    - Queen
        - Movement: moves a set number of tiles tiles up, down, left, right, or diagonally
        - Attack: when called to attack, if the target is within 3 tiles to the knight(within a 5 by 5 square of tiles with the knight at the center) it will attack the target

Want-to-have Features:
[These are features that you would like to have by the project due date, but you're unsure whether you'll hit all of them. A good final project would have perhaps half of these completed. At least 5 are required. Again, fully describe each.]

- More pieces implemented: rook, knight, bishop
- Any health statistic visible to the player is now represented in the form of a "health bar"
- Players place groups of pawn at a time
    - For example, instead of individually placing pawns, spending 1 energy at a time, one placement can spend more energy and place 3 pawns simultaneously in a formation with the center being the desired tile for the placement
- Networked 2-player version.
    - Two players can play against each other on different computers.
    - This allows for more fluid gameplay- pieces constantly move and players have to time their placements accordingly, more similar to Clash Royale.
- Player vs Computer version.
    - The computer logically picks pieces to place, also allows for more fluid gameplay.
- Player skill ranking system.
    - Would allow players to be matched to play with people of closer skill level (would work if there are multiple people, like 10, trying to play the game at the same time). This would be determined by the number of wins of a player, and other information pertaining to their in-game performance.
    - As players win more games their skill level would go up. As players lose more games their skill level would go down. If they win or lose by a lot, their skill level would change more.
- Game music/sound effects.
    - The music would play in the background while the game is running and can be different for the menu and in-game. Sound effects would play when players place a piece or while a piece is damaging another piece.

- Game timer/countdown.
    - When the timer ends it will start another timer and it will be sudden death (the next player to destroy a tower wins). Once the new timer ends, whoever has dealt more damage to the other tower will be the winner.
- More advanced piece behavior
    - Different pieces start to attack differently, maybe have new abilities, new movement patterns, etc. For example, the rook can become a sort of building that stays stationary and shoots at approaching enemy pieces
- The king attacks enemy pieces

Stretch Features:
[These are features that we agree a *fully complete version of this program would have, but that you probably will not have time to implement*. A good final project does not necessarily need to have any of these completed at all. At least 3 are required. Again, fully describe each.]

- Match history & player statistics.
    - Could show information like information about your last few matches (if you won/lost, match duration, date/time) and information about the play (# games won, # games lost, time spent playing).
- A tutorial mode.
    - This would explain how to play the game and maybe give some tips.
- Emotes.
    - There would be a menu that can be opened (probably a button next to where cards are selected). On the menu would be different emotes. The player would click on an emote to use it. The emotes would be basically emojis that both players can see. They can be used during a match (using a basic networked chat room).
- Coins for upgrading and purchasing items.
    - These would be acquired through winning matches, and could be used to improve the health of a character or increase the damage it does.

<u>Class List</u>:
[This section lists the Java classes that make up the program and very briefly describes what each represents. It's totally fine to put this section in list format and not to use full sentences.]

- Main - runs all the code in one place
- DrawingSurface - Holds the database stuff and connects all the screens
- GamePiece - The basic design of a piece which a player can use
    - **Queen -**
    - **Bishop -**
    - **Rook -**
    - **Pawn -**
    - **Knight -**
    - **King -**

- Player - Represents a player in the game
- Board - Holds the actual playing field for the game
- Post - Represents a post to the database
    - UserPost - Represents data regarding the user specifically
- ScreenSwitcher - Holds data to switch between different Screens
- Screen - a basic design for a screen
    - ScreenMenu - the start screen with a few beginning options
    - ScreenLocalNameCreate - screen to input player names
    - ScreenOnlineNameCreate - screen to input your player name (only online)
    - ScreenQueue - a buffer screen while waiting for an opponent to play against (only online)
    - ScreenLocalGame - the screen that holds the game and its elements for the user to interact during a match (placing pieces, seeing enemy pieces, etc)

Credits:
[Gives credit for project components. This includes both internal credit (your group members) and external credit (other people, websites, libraries). To do this:

- List the group members and describe how each member contributed to the completion of the final program. This could be classes written, art assets created, leadership/organizational skills exercises, or other tasks. Initially, this is *how you plan on splitting the work*.
- Give credit to all outside resources used. This includes downloaded images or sounds, external java libraries, parent/tutor/student coding help, etc.]

Team contribution at Beta release:
Pranav: Timer, choosing pieces, Rook, ScreenLocalGame, ScreenSecond
Richard: GamePiece + most subclasses
Nishil: DrawingSurface, ScreenLocalGame, and networking/database

Graphics library: Processing
Game Sprites: Google Images
Networking: Firebase/Google