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----------

Devil's Diner

Authors: Iris Chou, Sohum Phadke, Chloe Poon
Revision: April 27, 2022

Introduction:
[In a few paragraphs totaling about ½ page, introduce the high-level concept of your program. What this looks like depends a lot on what type of thing you are making. An introduction for an *application* will look different than one for a *game*. In general, your introduction should address questions like these:
What does your program do?

What problem does it solve? Why did you write it?
What is the story?
What are the rules? What is the goal?
Who would want to use your program?
What are the primary features of your program?]

Welcome to Devil's Diner, a cooking game set in post-apocalyptic times where the player has three lives. The user manages a person moving around a kitchen with a top-down view of the room using arrow keys. The person's objective is to complete orders that appear on the top of the screen periodically. To complete an order, the user has to put all the ingredients into a hole in the wall/ground. The order in which ingredients are put into the hole doesn't matter (for example, the tomato can be put in the hole before the bread and cheese). However, if the order is incorrect, the order won't go away, even if the food is put into the hole. If the order shelf fills up with orders (5 or more), the player loses a life, and the order shelf gets cleared. The ingredients are themed with the post-apocalyptic theme of the game such as radioactive onions and roadkill rabbit. The orders start out coming at an interval of 1 minute apart, but as the game progresses, the orders come 5 seconds faster, capping at 10 seconds. The faster the orders come out, the more ingredients are listed on the order. Occasional disasters take place, requiring the player to complete certain actions in order to prevent or fix the problem. The user can address the flood by putting down a sandbag at the door, and the fire by using the fire extinguisher. If the user does not properly address the disaster, it will either damage their cooking progress or cause them to lose a life. The player can gain more lives by finishing 10 orders, with a maximum of 3 lives at a time. If the user walks into the hole, then the user will lose a life.

Similar UI to this:

Instructions:
[Explain how to use the program. This needs to be **specific**:
Which keyboard keys will do what?
Where will you need to click?
Will you have menus that need to be navigated? What will they look like?
Do actions need to be taken in a certain order?]

To move around, the user uses the up, down, left, and right arrow keys with the exception of sprinting. To sprint faster, the user has to hold the shift key down while pressing the moving keys. In order to pick up and drop off food, the user can press the spacebar. When the player is sprinting, they may use wasd or the arrow keys to move. To pick up/put down objects, the user presses the spacebar. The user can only have one object at a time. The user can only pick up ingredients and throw food into the hole if they are at most 10 pixels away. The user presses the spacebar to drop food into the hole in the ground. To put out the fire, the user must pick up the fire extinguisher, walk over to the fire, then wave the fire extinguisher around for two seconds while holding down the left mouse key. For sandbags, the user has to pick up the sandbags by holding down the enter/return key and letting go when the player has walked to where they need to place the sandbag. In the case that there is a power outage, the user needs to go to the light switch and press the return key to turn the light back on. If any of the disasters isn't resolved in 10 seconds, the user loses a life and the disaster is resolved and orders are cleared.

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)]

- Floods: The user gets a flood notice and they must put down one sandbag at the door in 10 seconds. If they don't the screen turns blue and the user loses a life or the game ends. To put down sandbags, the user must press the enter key to pick them up, hold onto the enter key as they move to the door, then release the enter key to drop the sandbags
- Cooking: The user uses keys on the keyboard to move ingredients around and make food to fulfill the orders of the customers. If the number of orders waiting to be filled is more than 5, then the user loses a life. To complete an order, the user puts the ingredients into a hole in the ground. If the user gets the order wrong, the order disappears into the hole and they have to redo it.
- Lives: For every 10 orders completed, the user gains a life. They can have a maximum of three lives. A life can be used if the place floods or burns down. If the player loses a life, any disasters and orders are cleared.
- Counter: A table to put down a single dish the player is holding. There is only one table in the entire kitchen.

- Start: The game starts with a black screen, and the user has to press ‘H’ to start the game. The start screen also shows the user how to use the different keys on the keyboard to control their motion.
- Ingredients: The player must pick up the correct ingredients for an order located around the kitchen, including roadkill and rotten tomatoes.
- Fire: The user sees a flame, and if the user doesn’t put out the fire with the fire extinguisher in 10 seconds, the place burns down and the user loses a life or the game ends. The user puts out the fire by waving the fire extinguisher in front of the fire while holding down the left mouse button for 2 seconds.
- Power outage: The screen goes dark as if the lights in the kitchen have gone out. The user must make their way to the light switch, while not being able to see. If the user doesn’t turn the lights within 10 seconds, they lose a life. To turn back on the light switch, the user must press the enter key.

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.]

- Increase the Difficulty (each time an order is completed, the next order joins the order shelf at a faster rate. The first order must be delivered in 1 minute, the second order must be delivered in 55 seconds, the third order must be delivered in 50 seconds.) As the speed increases, the difficulty of the order will also increase.
- Add more food and recipe options for the user
- Currency: The user gains 10 coins for every order they finish, displayed in the top right corner of the screen. After the game ends, the value is displayed on the screen, acting as the total points the player earned throughout the game.
- Sprint: The player has a bar that indicates their sprint. The user can hold the shift key to move faster, which will deplete their sprint bar. The sprint bar naturally refills over time, so that the total bar fills up within 1 minute. While sprinting, the user can use wasd keys or the arrow keys to move.
- Background music: During the entire game, cheerful and upbeat music will be playing in the background.
- Stoves: Some ingredients should be placed on the stove to cook the item before being added to the dish.

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.]

- Create moving customers who order their food in the kitchen rather than their paper orders
- More realistic looking disasters (flooding shows water slowly filling the kitchen, fire looks like fire, etc.)

- Multiplayer: 2 players can work together in the same kitchen, where orders come faster and disasters are more frequent.
- Sound effects (just like fire burning, pan frying, water running, orders coming, etc. → typical kitchen sounds)

Class List:
[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.]

- Player class: The player that the user controls and moves using the proper controls
- Orders class: The order class will be used to create an order, randomizing between the possible ingredients
- Ingredients class (constructor takes a different image and name for the ingredients): Ingredients can be picked up and have their own icons on the grid, extends the interactions class
- DrawingSurface class: The grid that everything is located on, and moves based on the keys pressed
- Disaster class: A disaster in the game that the player has to account for in a set amount of time.
- Main class
- Interactions class: A superclass for anything that can be interacted with (picked up and dropped/used)
- Counter class: Represents the table that the player can place a dish on, extends the interactions class
- Hole: The hole in the ground that the player drops food through, extends the interactions class

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.]

- Sohum: DrawingSurface, and Player class
- Iris: Counter, Hole, and Disaster class
- Chloe: Main, Orders, Interactions, and Ingredients class