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----------

Revolve

Authors: Vihaan Chinthakindi, Kevin Valencia
Revision: 4/14/22

Introduction:
Our program is a 2D platformer puzzle game where the player has to traverse several levels by rotating the room in order to get to the exit. We wrote this program as an homage to the many flash games we played in our childhood of which many do not exist anymore. <Story?> The player must use their minds to figure out the correct combination of rotations either clockwise or counterclockwise

in order for the player to avoid spikes and reach the exit door to the next level. The player has the ability to rotate the entire level in 90 degrees clockwise and counterclockwise , as well as the ability to move left, right, and jump. Hitting the spikes results in instant death and will restart the player back to the beginning of the level. There will also be other methods of death such as saws and projectiles for added difficulty. A level is completed if the player passes through the door at the end of a level. The primary feature of the game is the revolving mechanic of the levels, making for an interesting spin on platforming games, with background music and sound effects as well. The final goal for the player is to finish all of the levels and find out the reason why they are solving so many different puzzles and rooms. Each level can be played multiple times. We hope to bring a nostalgic feeling to the players and remind them of the exciting and fun flash games that we all enjoyed in childhood.

[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?]

<u>Instructions</u>:
[Explain how to use the program. This needs to be **specific**:
Which keyboard keys will do what?

- **WASD and Arrow Keys Will Denote movements left, right, and up**
- **Space Key for Jump**

Where will you need to click?
Click icons to change levels, quit game, skip ahead
Click ‘R’ to rotate the level clockwise 90 degrees.
Click ‘E’ to rotate the level counterclockwise 90 degrees.
Click ‘Q’ to interact with doors to complete a level.

Will you have menus that need to be navigated? What will they look like?

- **There will be a menu at the beginning of the game, instructing user to start first level.**

Do actions need to be taken in a certain order?

- **Players start out on Level One, and go to the next level if they skip or complete it. Any completed or skipped levels can be played back again.**

<u>Features List (THE ONLY SECTION THAT CANNOT CHANGE LATER)</u>:

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)]

- Main menu that initiates the game. Will have a select level button, a shop button, and an exit game button, with a shop and level selection screens.
- Different ways of death ie spikes, and saws, that when interacted with will prompt the user to restart the level.
- There will always be a door and end of the level, signifying the end of the level. The player will enter the end door to complete the current level and will be prompted back to the menu.The doors will be colored.
- Numerous levels of varying difficulty, with all other features described here, with a start point and a door to denote the endpoint
- There will be music for each level or menu, as well as sound effects for actions in the game such as death, jumping, completing level.
- Gravity that always points down relative to the user will be present in each level, as well as the ability to move left, right, and jump relative to the user's orientation.

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 detailed backgrounds/wallpapers for each level.
- Special Features/Powerups/Buffs in each level
- Currency System where coins are earned and a user can purchase permanent power ups
- Option to play as different characters with a menu detailing the advantages/disadvantages of each different character.
- Animated running/walking for the character so motion is smooth, animated death, and animations for the projectiles and saws.

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.]

- Multiplayer functionality so levels can be played with different players on different screens.

- Ability for the player to design their own level with all the different combinations of walls/spikes, ability to change settings such as jump height, colors/characters of the level, and then the ability to play it.
- Different game modes such as timed, classic, and a gamemode in which the player must beat the level before a White Fog envelops the entire game-screen.

Class List:
Player.java
Level.java
Spikes.java
Saw.java
MainMenu.java
Main.java
Coin.java
DrawingSurface.java
LevelMenu.java
ScreenSwitcher.java
ShopMenu.java
Trap.java
Door.java

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.]

Vihaan: Javadoc Classes & Methods, Jar-File, Finalized & Came up with many parts of UML (added class connections, new classes, potential & existing fields & methods), Created packages & classes, did basic LevelMenu with buttons)

Kevin: Javadoc Classes & Methods (& added javadoc file), contributed to UML, Added & Formatted buttons to MainMenu with text, made sure they worked & route to the correct menu by making sure DrawingSurface & Screen work as intended.