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 are 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----------

Mazes & Monsters

Authors: Jay Dalal, Junbok Lee, John Tahk
Revision: 04/12/22

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?

You are a secret agent in the CIA who specializes in finding lost historical artifacts. So far, you have acquired da Vinci's lost paintings, relics from medieval empires, and books from the Library of Alexandria that were previously thought to be gone, reduced to atoms. However, you have been assigned one of the hardest missions yet: acquiring the legendary spatula of the infamous Habanero Flubberwickles, worshiped by the ancient and powerful Qelthok clan.
Luckily, you have already been tipped off on the location of the item: a haunted castle located far far away that was built a long time ago. However, you must now navigate through the complex corridors of the castle in order to eventually gain access to the map.
In this game, a user-controlled character encounters a complex maze of walls and pillars. As they navigate through the maze with a limited top-down view, they encounter monsters that might obstruct their path. The character may navigate around monsters by wearing cloaks (which are single-use and only work for a limited amount of time) to appear invisible. Try to solve the intricate maze, pass through monsters, and succeed in solving the puzzle of the ancient swords.

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?]
The player will use WASD controls to move in a 2D top-down plane. The E key will be used for any interactions, and mouse clicks will be used to navigate the menu. When the program is launched, it will direct the user to a starting menu. On the starting menu, there will be a play, tutorial, and exit button which the user will be able to click on. Hitting the ESC key while in the game will bring up a menu that allows the user to resume the game or return to the title screen.

Features List (THE ONLY SECTION THAT CAN NOT 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)]

- Maze - The user will control a character that will navigate through the complex maze. The maze contains obstacles such as pillars, walls, and dead-ends, which the user will have to navigate through.

    - There will be 3 mazes of increasing difficulty. In the first maze, it will be a normal maze only with the blindspot mechanic. In the second maze, it will introduce the first type of monster along with the cloak, and the final maze will introduce the last two types of monsters.
- User-Controlled Character - The user-controlled character is a secret agent who navigated through the maze. The player will be able to control the character using the arrow keys, but running into monsters unprepared can cause the character to die and the user to lose and restart the game.
- Monsters - These monsters will move in predictable patterns(in a circle, back and forth, etc.) until a player comes close enough to the monster, then it will start chasing the player until the player is caught or the player runs out of the detection zone. Monsters' movements will differ from when they are passive and aggressive.
    - MonsterMove: First type of monster is a monster which moves at 60% of the player's speed and will follow around the player until the player is either caught or the player runs outside the monster's radius.
    - MonsterDodge: Second type of monster: when the player enters the monster's radius, the monster will signal an attack on the ground which gives the player a few seconds to dodge the attack. During this time, the monster will stand still.
    - Monster Third type of monster will remain stationary, but shoot a laser at the player. The laser will track slightly behind the player but if the player stands still, it will catch up. To dodge the laser, the player must stand behind a wall or pillar.
- Cloaks - Cloaks are items on the ground that can be picked up by the player. With the cloak, the player will stay invisible and not attract monsters. The player's sprite will also change so it is more translucent than before. The cloak has a timer, so the player cannot stay invisible forever, but picking up another cloak will extend the timer.
- Blindspots - In the maze, there will be pillars or walls which will block the player's line of sight. That will black out that portion of the screen and the player will not be able to see what is in that area until the player comes into the line of sight.

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.]

- Radius with Flashlight - This is similar to the blindspot in the sense that the user's vision is limited. Essentially, the character would be holding a flashlight that provides light for a certain radius outside the character. The space outside this radius would then be blacked out.
- Introduction Screen - Before being directly placed in the game, it is a good idea to have a home screen that includes simple and readable instructions and provides a "Play" button for the user. In addition, this may be where the user can choose their character (this is a Stretch Feature)
- Levels - If the user can successfully finish the first level, we could include a continuation of the maze with slightly harder obstacles. For instance, harder levels could open doors that need to be opened by switching levers or different variations of monsters.

- Sprite animation - In order to improve the aesthetics of the user experience, we could animate certain parts of the program. For instance, we could make the user-controlled character look as if it was running when it moves or animate the monsters to move around the maze so that they seem more menacing.
- Food - This could be another challenge that the user has to deal with. We could include food icons, which the user will have to collect every minute to avoid dying.

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.]

- Sound - Similar to the Sprite Animation, adding music and sound effects can improve the overall aesthetic and experience of playing the game. Adding background music could get the player to be more interested in playing. Including sound effects during important moments, such as when the user puts on a clock or dies, can help add realism to the project.
- Boss level - The boss level will include some boss enemy that the player will have to defeat through a series of puzzles and mechanics. The boss will have a variety of attacks depending on the player's distance from the boss, making it a dynamic foe to play against. To defeat the boss, the player will need to complete some puzzle or mechanic in a set amount of time to deal damage to the boss or will die and have to restart the level.
- Chose character - In the Introduction Screen, we could prompt the player to choose between multiple characters. We plan to have different looks for the characters, but every character will have the same speed.

<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 a list format and not to use full sentences.

Screen - an abstract class representing the screen the user will be seeing

- **StartingScreen** - extends Screen, the screen that is first seen when the game is launched
- **GameScreen** - extends Screen, will be displayed when the game is running
- **EscapeScreen** - extends Screen, displays the menu for when the ESC key is pressed
- **ScreenSwitcher -** allows switching between screens.

Level - an interface that represents the levels in the game, which includes monsters, invisibility cloak, walls, pillars, doors, the player, and possibly more

- **FirstLevel** - extends Level, the information needed to display the first level
- **Obstacle**

Sprites - a class that represents objects that will interact with the user(and includes the user)

- **Player** - extends Sprite, the sprite which the user will be controlling in the game, and the behavior will be determined by the keys pressed which are determined in DrawingSurface
- **InvisCloak** - extends Sprite, the sprite which the player can pick up if it collides with the Player, and will grant the player invisibility

- **Monster** - extends Sprite, and is not player controlled. This class will have subclasses for the different types of monsters, where the methods which control movement or player tracking will be overridden

DrawingSurface
Main

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.

Below we have listed how we split up the class.

John: Level class and subclasses

Jay: Set up the inheritance hierarchy between Screen and its subclasses GameScreen, EscapeScreen, and StartingScreen, created the interface ScreenSwitcher, implemented ScreenSwitcher in DrawingSurface and ensured that the StartingScreen was properly illustrated, updated UML for Screen class and subclasses, ScreenSwitcher interface, DrawingSurface, and Main classes

Junbok: Javadoc for Sprite class and subclasses

Each team member contributed to the classes in the UML diagram that they coded. Everyone collaborated on the Grid, Location, DrawingSurface, and Main classes.

The structure of the classes in the Screen class were used based on GamePhysicsDemo