Machine Problem 02: Documentation
Mr. Edgardo Felizmenio

Name and Section of group members:

- Arnante, Mary Josephine - CS11 B
- Cariaga, Justin - CS11 B
- Cayetano, Anthony Van - CS11 E
- Macalisang, Harold Andrei - CS11 E

Name of Application: Gusto Kong Sumabog

1 The Program Structure

The program is divided into three parts: the main program file, the resources, and the game engine.

```
game/
      main.py
      engine/
            __init__.py
            game_objects.py
            window_objects.py
            resources.py
            util.py
      resources/
            audio/
            data/
            visuals/
```

The main.py is the main program file that is executed by the user to run the game. The GameWindow class is both the game window and the controller of the game state. All objects (game objects and window objects) are instantiated in this class. This class is the main controller of the game state, as it contains the main game loop (the update method) which is executed 60 times per second via the pyglet.clock.schedule_interval method; thus updating all the objects' state every fraction of a second. This allows the game to have approximately 60 frames per second.

The resources/ is a directory divided into three directories: the audio/ which contains the sound effects and the background music; the data/ which contains the json data of the highscores and the data used by the MgaWords objects; and the visuals/ which contains the artworks and sprites used.

2 Batches and Groups

The sprites in the game are drawn by batches, not individually. The game utilizes two batches: the gameplay_batch and the menu_batch. The gameplay_batch is the batch used by sprites that are present when the player state is “alive,” i.e during gameplay, otherwise, the menu_batch is used. A batch is drawn on the on_draw event handler method depending on the player state. The following code summarizes this:

```
def on_draw(self):
    self.clear()
    if self.player.state == "dead":
        self.menu_batch.draw()
    else:
        self.gameplay_batch.draw()
```

Groups are basically the layers of a frame. They are the order on how the objects are rendered. The game utilizes 6 groups; the first three are arranged so that the background images are rendered first, next are the foreground objects (such as the Button containers) and last are the texts. The last three groups have similar pattern but are for popup windows (which should always be on top of the frame).

3 The Game Engine

The game engine is a package that contains 4 modules: window_objects.py, game_objects.py, util.py and resources.py. The __init__.py file is used to mark the engine/ directory as a Python package. If no __init__.py is found inside a directory, Python will no longer look for modules in it; thus, attempts to import the package will fail.

3.1 Window Objects

The Window Objects are the interface objects, which are objects that are not part of the gameplay. These include the buttons and background images.

All the Window Objects are contained in a dictionary in main.py called window_objects. This dictionary has two keys “alive” and “dead”. Each key corresponds to a list of Window Objects depending on whether these objects are present in a certain player state. The following code block summarizes the window_objects dictionary:

```
self.window_objects = {
    "dead": [self.menu_screen, self.start_button, self.help_button, self.quit_button, self.back_button, self.help_window],
    "alive": [self.gameplay_screen, self.explosion, self.retry_button],
}
```

Hence, when the player state is “dead,” only those Window Objects in the value of the key “dead” should be updated.

```
for obj in self.window_objects[self.player.state]:
    obj.animate(dt)
```

```
for obj in self.window_objects[self.player.state]:
    obj.reset()
```

3.1.1 The WindowObject Class

The WindowObject class is a class that inherits the pyglet.sprite.Sprite class, which means that this class is just a sprite itself, but with an added property: animation. There are two types of animation defined in this class. The first is the slide animation. This animation changes the x and y position values of the object until it reaches its target position. The second animation is the explosion, which changes the scale_x and the scale_y property of the object until it reaches its target scale.

Examples of WindowObjects are the background images, the help window and the explosion sprite.

3.1.2 The Button Class

A Button is an interactive WindowObject; thus, the former inherits the latter. Like a WindowObject, it also has an animation. However, this object has a custom method which is executed when the user clicks it. The custom methods for the buttons are defined on the GameWindow class, which are passed to the Button objects when they are instantiated.

An example of a Button object is the start button, whose function is to start the game when the user clicks it (the start method in the GameWindow Class).

3.2 Game Objects

The Game Objects are the gameplay objects, which are the objects that affect the gameplay.

3.2.1 The Player Class

A Player is the user itself. It is an object with a name label, score label, lives, and input label (this is the input when the player is playing the game). A Player also has a key state handler, which is the on_key_press method. This allows the Player to record its input and store it on its name or input attribute depending on the context.

During the gameplay, whenever the Player presses Enter, whatever is the value of the input label will be compared with the texts contained by the two MgaWords objects. If the input is the same as one of the MgaWords objects, then the Player's score attribute will be increased by a number of points depending on the type of the MgaWord object.

3.2.2 The MgaWords Class

The MgaWords Objects are the "enemies" of the Player. These objects are of two types: "di_masama" and "masama." The type of the object is stored in its type attribute. MgaWords objects of different types have different initial speed, acceleration and animation.

The MgaWords Class gets its choices of words on the mga_words.json file, which is located in the resources/data/ directory. The file is a dictionary with two keys: "di_masama" and "masama." Each key corresponds to a list of words, which is passed to a MgaWords object depending on its type. The list is shuffled at the start of the game via the random module.

If a MgaWord object manages to reach the bottom of the screen without the Player typing correctly its word, then the Player loses one of its lives.

It is important to note that only two objects of this class are instantiated, though it may seem that a new object is instantiated whenever an object reaches the bottom of the screen. This class has a pop method that sets the position of the object above the screen whenever it goes below the screen, hence the illusion of having more than two instantiated objects.

3.3 Utilities and Resources

The utility functions are stored in the module util.py. These are functionalities that don't need Pyglet to function, such as the get_optimal_window_dimensions and write_data functions.

All the resources located in the resources/ directory are loaded in the module resources.py. These include the images, music, sound effects, and the font. This is done so that all the resources are stored in only one place.

4 Event Handling

Mouse events such as mouse motion and mouse press are handled by the GameWindow Class on its on_mouse_motion and on_mouse_press event handler methods. The on_mouse_motion method checks via the is_mouse_over function of util.py if the mouse is over a Button and enlarges the Button if so. The on_mouse_press is also similar; it executes the Button's function if the mouse is pressed while it is over the button.

Keyboard events are handled by the on_key_press method of the Player Class. A KeyStateHandler Class was first instantiated inside the Player Class to allow the on_key_press method to be an event handler. To "register" this handler, it is necessary to push it to the pyglet.window.Window object via the push_handlers method. It is important to note that we need not use push_handlers on the mouse event handlers because those event handlers are already defined on the GameWindow Class (which inherits the pyglet.window.Window Class). The on_key_press event handler allows the Player to record and store its input on its attribute (name and input label).

5 File Input and Output

Files are stored as a json file and are read via the json module. This is so that the encoding and decoding of data will be easier, as a json object is basically just a Python dictionary. The read_file and write_data, located in the util.py module, are the functions used to input and output data from a file.