Final Project Report
Wengel, Julian, Garrett
CMSI 185
Dr. Mandy Korpusik

Acknowledgements

We used Tkinter to construct the entirety of the game, with widgets making up the interface, including the sidebar and board. We also used Python's random library to generate parts of the board, and ideally ship statistics. Finally, the file example.py was not implemented into the game, but was instead used as an example of code to understand how to properly use tkinter canvas. This code was found from user Bryan Oakley on stackoverflow.com. We also used Python's PIL (photo image library) to allow for the game to load in the images in the game. The images were found on google and were not created by us.

Team Responsibilities

Wengel:
- Design and create the sidebar of the game
- Support team mates

Garrett:
- Design and create the game board
- Support team mates

Julian:
- Integrate game across multiple files
- Support team mates

Challenges

File Integration: A large challenge that plagued our game was files communicating with each other. We opted to split our game up into multiple files to reduce merge conflicts and allow for individuals to code without interfering with others. This allowed for us to work without worrying about someone else pushing, or needing to push, at the same time. This comfort came at the cost of a more complicated code structure and made our code uniquely more difficult in two specific areas.

Data Transfer: The first issue our game ran into when integrating the 2 halves was getting the 2 halves to share variable information with each other. Multiple functions had to be created expressly for the purpose of sending and receiving data between the files which could have been done very simply with a global variable call in a single function. Variable data needed to be sent between the files to allow for each side of the game to update effectively with each other. The main example of this is the pulse_data variable. The pulse data variable is an information store variable that stores a Space object. This allows the game to hold onto the data of the tile you just clicked so that it may be used by other game functions like movement and firing.

```
pulse_data = None # Temporary store variable for holding object data
```

Can be found at the top of tkinter.py

Function Transfer: The second issue with file integration is getting functions to transfer between files. Main.py couldn't just run the other files (sidebar.py and tkinter.py) to create the game. The other files had to be turned into usable functions to be run by main.py to create the game. This created the unique problem of function definitions being inside other definitions. While functioning, admittedly, it feels and looks wrong to code this way, but it allowed for main.py to use the other files while allowing the others to run individually. A good example of this in our code was getting our dictionaries and board tile classes to update with the new Ship class or Space class tile information. This required the tkinter.py file to send new update data (via pulse_data variable) to sidebar.py so that when you moved your ship, the sidebar didn't show un-updated data about the tiles that just changed. Ie: Clicking on the old space tile the ship just moved off of and it still showing Ship tile data and the new ship tile showing only Space tile data.

```
# Defines data collection from clicking on a game board tile
def id_pulse(event):
    global pulse_data
    print(pulse_data)
    id = event.widget.find_closest(event.x, event.y)[0] # Id of tile clicked

    # Id dictionary filter. assigns pulse data value of tile object clicked
    if id in blue_ship_dict:
        pulse_data = blue_ship_dict[id]
    elif id in red_ship_dict:
        pulse_data = red_ship_dict[id]
    elif id in stand_dict:
        pulse_data = None
    elif id in rock_dict:
        pulse_data = None
```

Can be found at the top of tkinter.py

Continuous Updating: Another large challenge to overcome was how to get the game to "run". Due to tkinter being used to create mostly static entities, the game needed a way of continuously updating the tkinter frames to run. The way we fixed this was by having the game actually update every time you left click anywhere on the screen. That's why there is a short flash on the sidebar everytime you click. In code terms, main.py's refresher function actually recreates the entire game every time you click, just with new information to give the illusion of a continuously running program.

```
if travel_key == 0:
    def refresher(event):
        create_sidebar(side_frame, 0)
```

Can be found half way down main.py

Proposal vs Final

Besides adding animation and noises, we added most of our initial proposal. Early on we over took our minimum requirements and moved into implementing our additional features. Additional features beyond our basic goal we were able to implement was having four variables for the sidebar, including armor and shield statistics. These statistics added complexity to a units lifespan where some ships could take more damage than others, rather than instant kill. Our primary objective was to create a turn based strategy board game based on units that the player controls, functioning similarly to a game of checkers. Taking place in space, the objective for each player is to eliminate their set of spaceships, using strategic positioning and each spaceship's statistics. We achieved our grid game board and a variety of variables that would update statistics according to the user interaction.