Critical

Mass

Code by: Caleb Barynin, Lucas Cowell

Teacher: Keith MacDonald

Course: ICS4U, (grade 12 Com Sci)

DATE: 1/19/2018

Contents:

What Is *Critical Mass* Now?

.......................................................................................................................................................2

Game End Goal

.......................................................................................................................................................2

Program architecture

.......................................................................................................................................................2

Program Flow

.......................................................................................................................................................3

In depth module important module anylsis

.......................................................................................................................................................5

main.py

...........................................................................................................................................5

gameDisplay.py

...........................................................................................................................................5

playerClass.py

...........................................................................................................................................5

displayBarClass.py

...........................................................................................................................................5

buttonClass.py

...........................................................................................................................................5

imageLoader.py

...........................................................................................................................................6

Conclusion

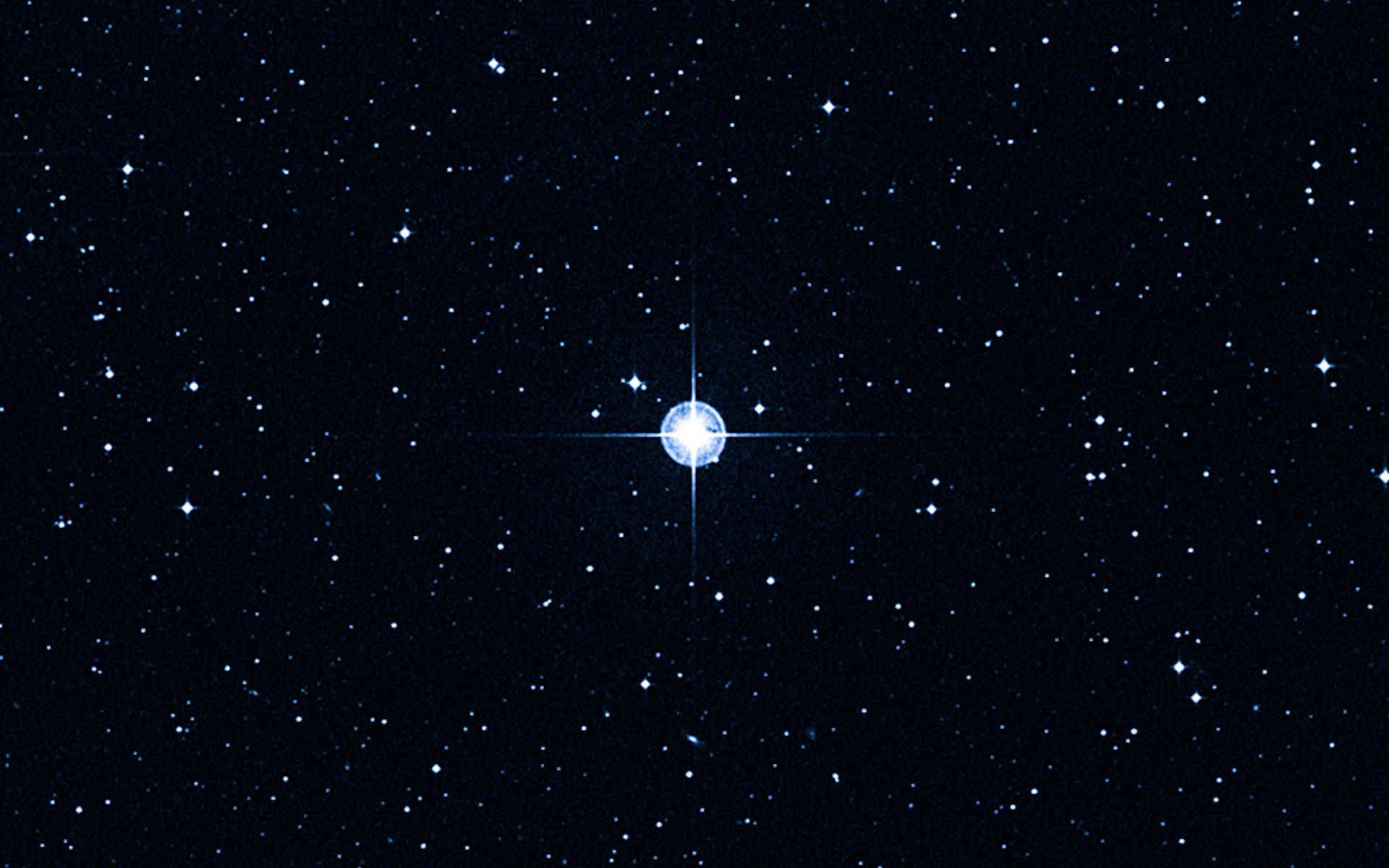
.......................................................................................................................................................6

Appendix A, Screen shot

.......................................................................................................................................................7

Appendix B, Python, Pygame, References

.....................................................................................................................................................10



What Is *Critical Mass* Now?

In its current state, the program we have created, is capable of bliting ships to a screen, projecting possible movement options, creating interactive buttons, boxes and bars that can be used to call functions and display information to the user (not to mention they look nice and everything works efficiently and smoothly). It can also upload picture and create ships (or sprites really) that can have photo surfaces uploaded to them and efficiently blit them to the screen. The current program is very malleable and once the base functions were written the program is almost fun to code! Every single value is a variable meaning all of the components of the game can be optimized for whatever game the user of this program could want. Critical Mass is not in the state that it will be but while the program does not have a lot to do and may not encourage you to play it for hours on end, it is phenomenal pygame UI that could be adapted to any turn based strategy game. (This is really useful tool to have for future years.) If there’s one thing to take away from *Critical Mass* it’s that you haven’t seen the last of it. DAY ONE PATCH!

Game End Goal

The game we set out to create, *Critical Mass*, will be a turn based strategy game where 2 – 4 opponents can have a space battle with star trek style ships. The way the game will flow is by each player deciding what orders they want their fleet to follow and at the end of the round. Once everybody confirmed their orders by each pressing the COMPLETE TURN button, there would be a massive animation of all the ships and projectiles. The orders would either be movement based or attack based, and would be declared by presetting the trajectory of the object. Additionally, ships could collide with other enemy ships but this would be a costly tactic. Each projectile would also have a price to be fired but no projectile would be the same and it would be there differences that would allow for more strategy. In terms of the in-game goals, there would be multiple types of games to play. Examples are who can destroy a predetermined amount of ships in a predetermined amount of turn, destroy opponents flag ship, destroy opponents fleet and so forth. Each ship would have a range of different attacks and speeds along with other stats to make each ship different and allow for each player to develop their own strategy and out-wit their opponents.

Overall, we looked into games that exist like this and were surprised to not find any, but after thinking about it, we realized that though strategy is popular not many game designers have not been able to take the strategy components that make board games so much fun and combine them with a video game environment.

NOTE:

All following text describing the program is referring specifically to the program submitted.

Program architecture

It is important to note that each file Directory in the background file has (or had) a module that starts with the word “main” and then the directory’s name. Each of those files are just used for the mass import of all modules of a certain purpose. Not all import statements in the program will have a full import of a directory because sometimes the directories cross reference and this can cause issues but the computer is very detailed in describing the bugs.

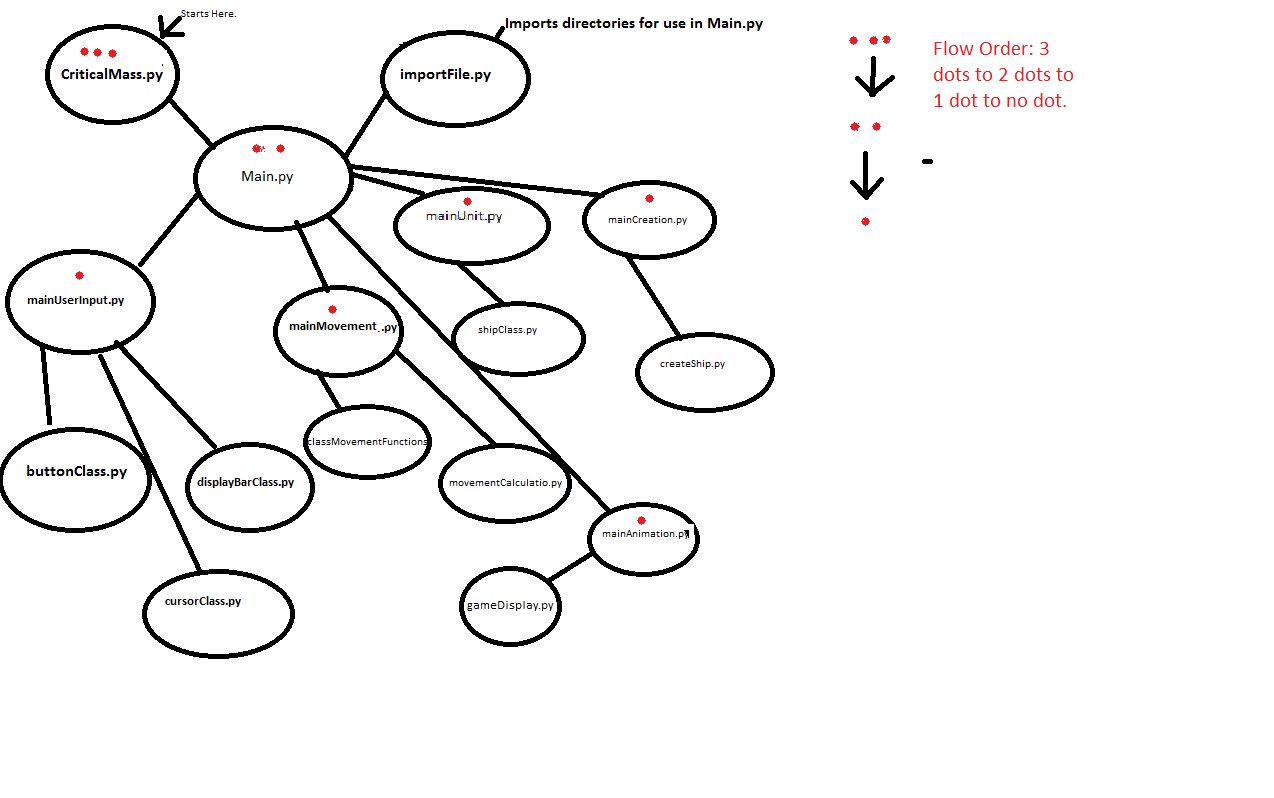
The programs architecture loosely follows diagram 1.a

diagram 1.a

In words, the whole program starts with the Critical Mass folder and then access the main.py module. This is described more deeply in module analysis but what id does is it opens all of directories and creates the run() function, which sets up the mainProgramWindow object from the gameDisplay module, which is where all of the logic and animation happens. Program flow deals with the mainProgramWindow object more in depth. From there each parent module imports its child module in a need by basis and for simplicities sake we didn’t worry about possibly over importing a module (importing the module in both a parent module and its child module).

Program Flow

The mainProgramWindow class is the meat of the program and all buttons, orders and any other events are dealt with in class. The program starts by creating all of the objects so it the program runs optimally.

The initial objects consist of:

* Players
  + Info bar (sprite group)
  + Fleets (sprite group)
  + Path setting lines (appendix A.Screenshot.3)
* Interactive bars (sprite group)
* Alert block (sprite group), (appendix A.Screenshot.5)
* Cursor (sprite group)
* Basic window status variables
* Background (sprite group)

The program follows a sequence:

1. Maintaining the FPS
2. Checking events
3. Updating
4. checking ship selection(used for movement preseting) or buttons
5. Draw everything to the screen.

Unless there’s an alert box in which case the whole program freezes for about 3 seconds and the sequence continues.

More in depth look into each turn:

Step 2, Checking events:

This step is really simple because it only checks for keyboard inputs to end the game because Ive moved the majority of the event getting into sprite class’s update() functions, so that its easier to understand what is and isn’t important to that sprites update().

Step 3, Updating:

Updating took a little bit of time to work on but it just makes sure that each sprite group is reacting to cursor location and buttons.

Step 4, checking ship selection(used for movement preseting) or buttons

Depending on which buttons have been pushed (specifically the movement buttons), the program will check the to see if any relevant sprites have been activated and if they have then a corresponding function will be called.

Step 5, draw everything to the screen

This step draws the background, then any trajectory lines, then all fleets, then info bars, then display bars.

Additional info:

* The shop bar allows for the addition of new ships to the current players fleet
* The endTurn() function is called when the COMPLETE TURN button is pressed and sets the current player to the next player in the or completes the round and restarts the round sequence
* Once each player has hit complete turn the game round is ++
* The Trajectory lines don’t save any data so ships can’t actually move. (That’s as far as I got...)

In depth module analysis:

There are more modules then the ones we have listed here, but these are the ones which are either called the most or were the basis for other modules or classes. We chose these one because we understand that you have limited time and we thought these modules gave you a good enough feel for how the UI for the game works. If you feel obliged to look at the more player oriented portion of the program, let us know and we can go more in depth for you.

main.py:

I, Caleb, usually put a function like this in each of my programs because it allows for a nice user friendly tree architecture. As previously stated in program architecture this file is the base initialize for all module imports. The module has three steps:

1. Adds new directories to sys.path, now all files located in background will be accessible.
2. Imports mainAnimation.py for the createMainWindow() function creates the mainProgramWindow object (which is where the magic happens.
3. Define the run() function (which is called in Critical Mass.py), which calls createMainWindow()

gameDisplay.py

Review program flow.

playerClass.py

The player class comes equipped with the basic player info and is used when updating and drawing things specific to the current player like trajectory lines.

displayBarClass.py

This is where the main UI bars (which are sprite groups) are define boxBar (for boxes) and interactiveBar (for buttons). Additionally the Majority of the pseudocode is here. The bars can be updated based on new information and do so quickly and efficiently (THIS WAS THE HARDEST THING TO GET

buttonClass.py

button class is a basic sprite class with several modes, and can return values if the program checks for them. The rest of the pseudo code is here.

imageLoader.py

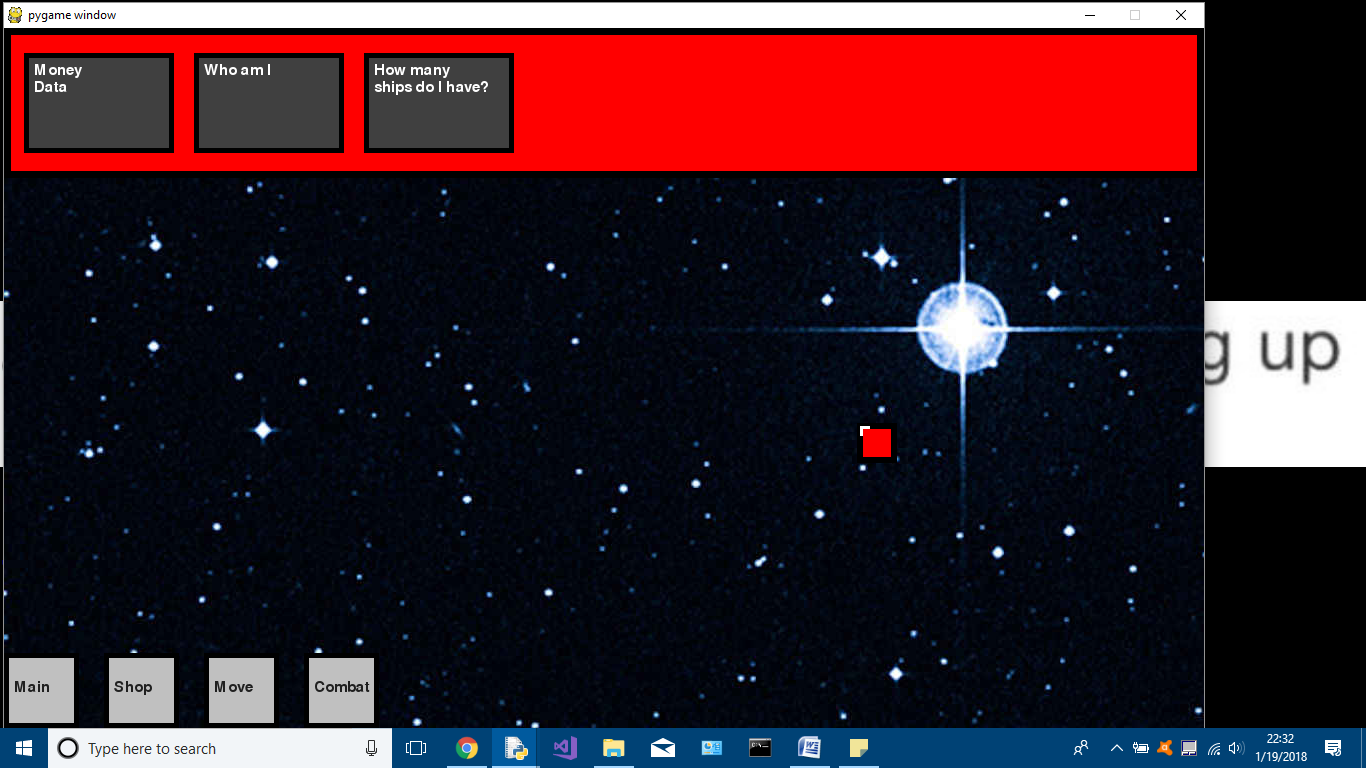
image loader imports image files and clears them of yellow (our color key, allowing them to be transparent in the right places) and set them to builtin variables so they are accessible anywhere in the program! It is for this reason the image variable don’t follow camel case (so there is no mix up).

Conclusion:

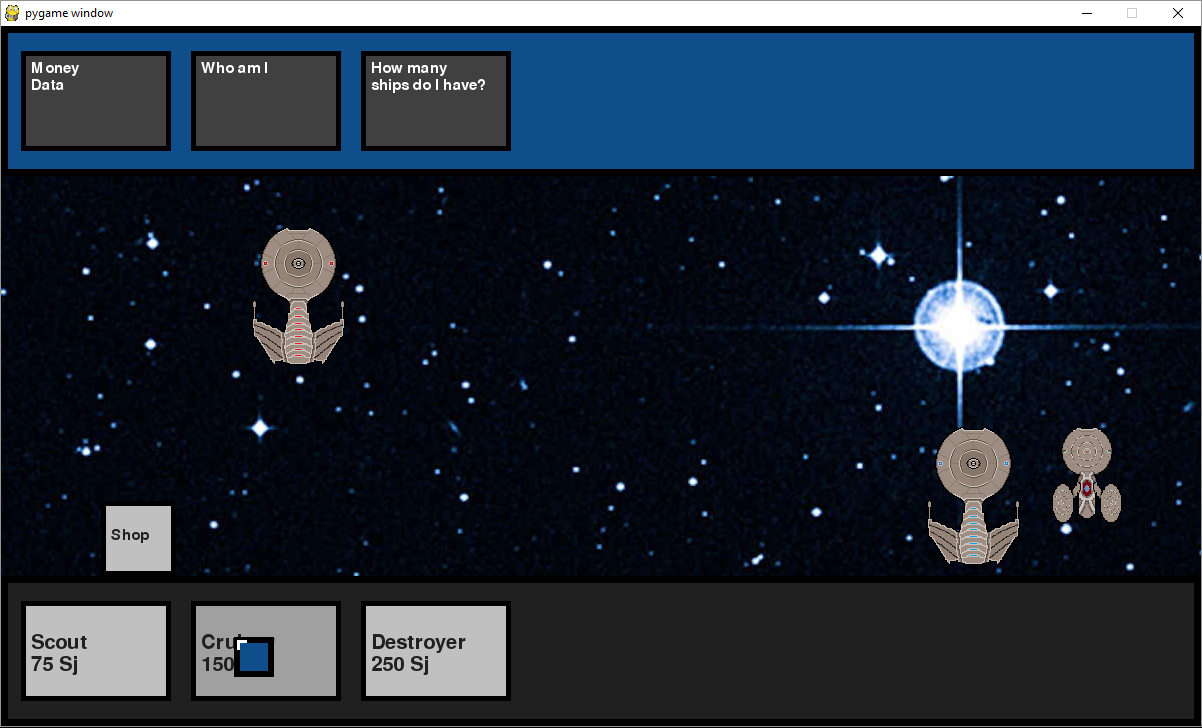
Working together on the summative we learned the importance of good communication and proper delegation of work. I think that had we done this a second time we would have made sure to have a basic architecture discussion to establish how variables objects and functions would be managed and called. Overall, would definitely do this again and we will actually continue to work on Critical Mass until it’s available the MASSes. Now time to submit!

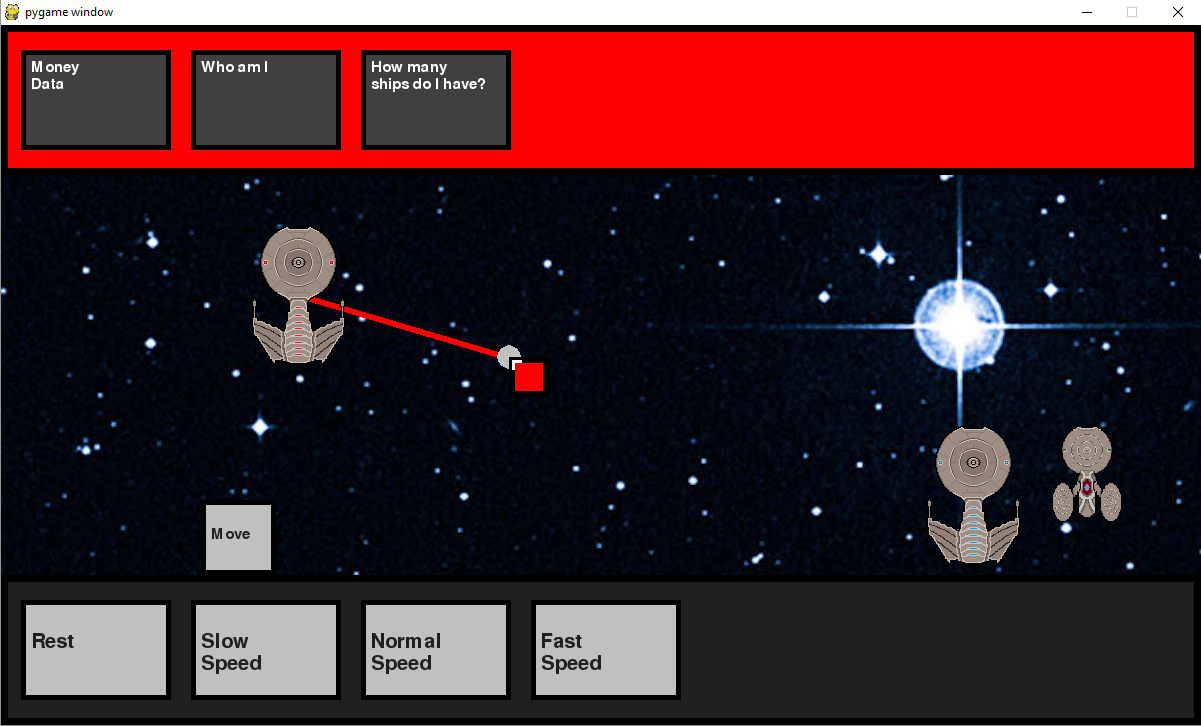


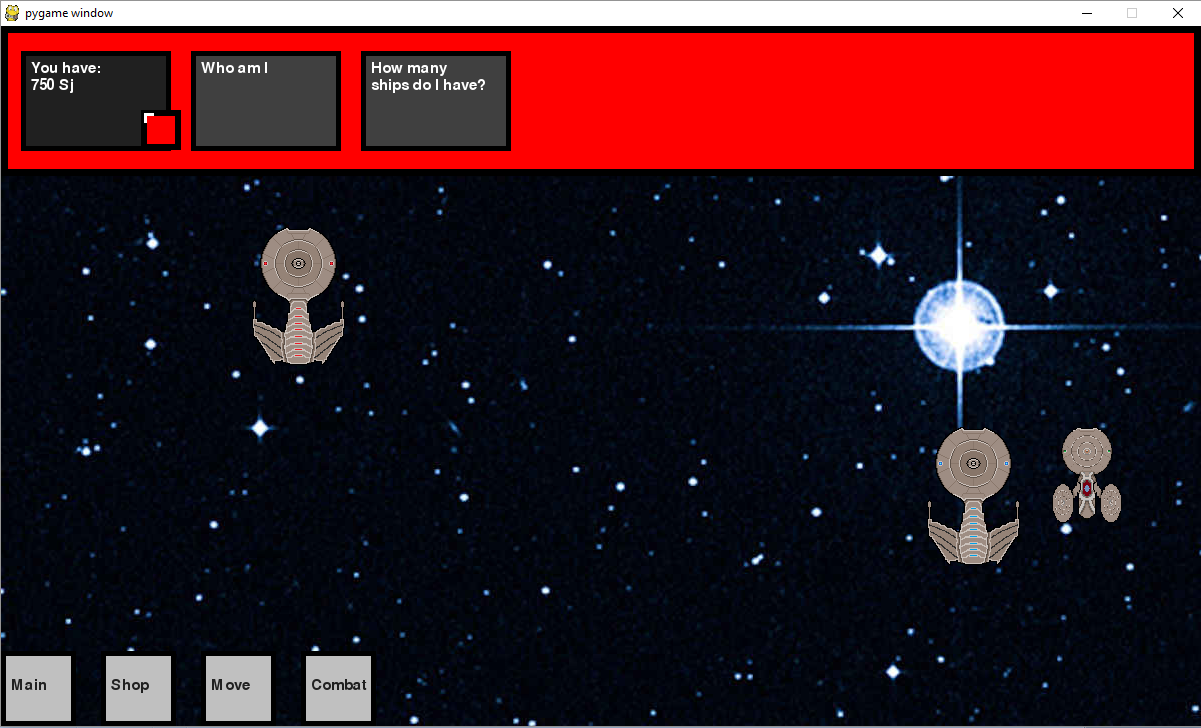
Appendicies:

Appendix A, Screenshots:

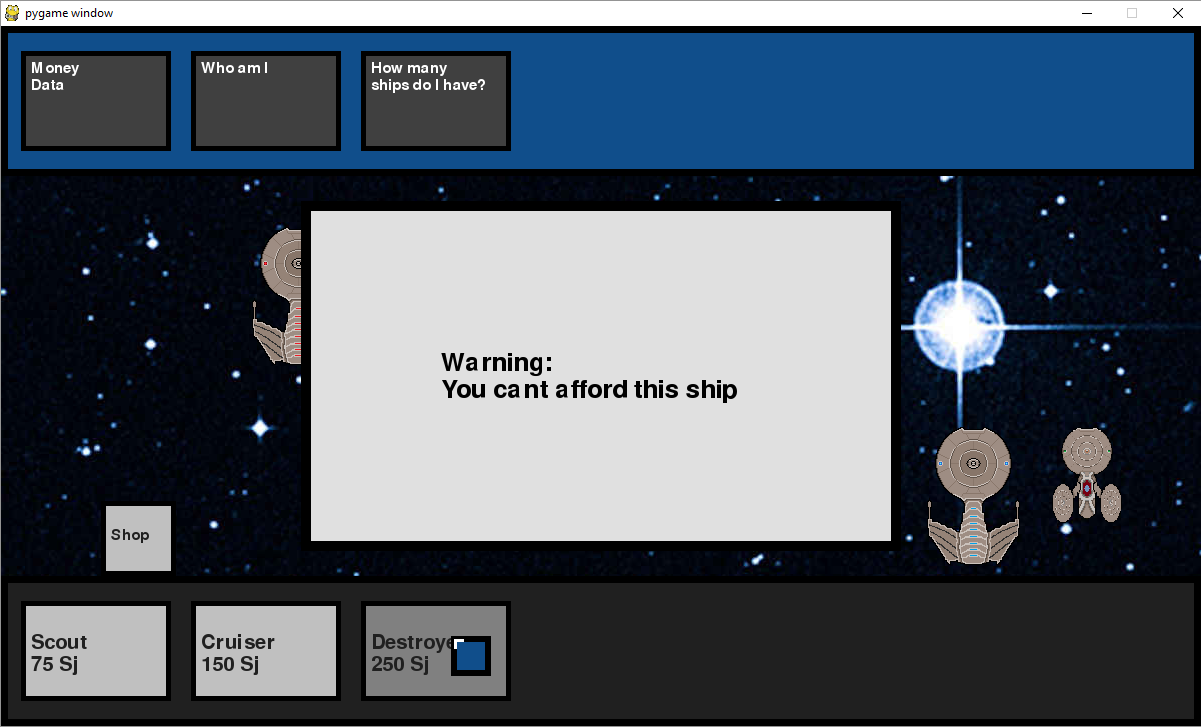
Screenshot # 1: start window, player 1

Screenshot # 2: purchasing ships, player 2

Screenshot # 3: setting movement, player 1



Screenshot # 4: hovering over money Box, player 1



Screenshot # 5: warning of insufficient funds, player 2

Appendix B, Python, Pygame, References:

The game was developed and tested on Windows, using Python 2.7.14.

The Pygame graphics modules tested were version 1.9.3,

Downloaded from https://www.pygame.org/download.shtml.

References:

Pygame tutorials at:

KidsCanCode: Pygame game development playlist

Channel URL: www.youtube.com/channel/UCNaPQ5uLX5iIEHUCLmfAgKg

Course notes: Mr. MacDonald's course notes for ICS3U