Group 1 Final Project Report (Supah Pew Pew)

By Matthew Barragan, Julian Bostick, and Nghi Truong

Problem Statement-

Supah Pew Pew is a side scrolling spaceship, top down shooting game. The objective of this game is the spaceship (player) to score as many points as possible before losing their three lives. The player has the ability to move up, down, left and right all while shooting enemies, each enemy destroyed corresponds to a set amount of additional points, which is added to the player's overall score.

After the game is launched, the player(user) will start at a main menu screen where you will see the game name and the game description. After the main menu screen the user will be presented the ship screen, where they can pick one of four ships to play. Each ship fires a unique bullet and has a unique sprite with a unique animation. A player can rapidly fire bullets at enemies that generate randomly on screen, these enemies will also fire randomly, which in turn causes the player to have to maneuver around enemy fire while simultaneously returning fire to score points. Colliding with an enemy ship or getting hit by enemy fire reduces the players life by one. A player has three lives, once they run out, the game is over. Once the game is over the user will be taken to the high score screen and if the user scored within the top 25, the user will be prompted to enter their name and then their score will be added to the high score list in the rank they achieved, which consequently will cause everyone beneath them to move down in rank, and the 25th person is no longer on the high scores list.

System Boundary-

Our project has different menu screens and once you finish with the current screen you move onto the next screen. The first gives the option to start the game, and gives the necessary controls, the user can move past this screen by pressing any button on the keyboard. The following screen gives the user the ability to choose their ship to play the game. Each ship is contained within a selectable box, and once a player clicks the box containing what ship they want, the game will begin. Once in the game the user can maneuver and shoot enemies that spawn randomly on the opposite portion of the screen. Each enemy spawns on the right side of the screen and moves in a straight line in the negative x-direction towards the player.

Functions that were excluded from the project were a pause screen, unique damage by certain ships, and unique enemy movement. The pause function would allow the player to pause the game during the middle of the game with options to continue or quit. All enemy ships cause the same damage to the player, the function that different enemies cause different damage was excluded. Also excluded was different enemy movement. This feature would allow the enemy ships to move in an unpredictable pattern making the game harder for the player.

Domain Analysis-

In our game there is a driver that runs the whole game and is the class used to run the GUI. The menu screens are broken up into 4 classes: Start Screen, Ship Screen, Game Screen, and High Score Screen. Each screen extends JPanel and Start Screen, Ship Screen and Game Screen implement Runnable. Start Screen, Ship Screen and Game Screen implement Runnable because they have animation for the ships and the title that run a time delay. Furthermore, the Start Screen has a grid to

contain the words displayed, and using KeyListener to control the action to pop up the follow screen Ship Screen. In the Ship Screen, the rectangles were drawn around each ships and user can use the Mouse to pick the ships they want to play by implements MouseListener to the screen.

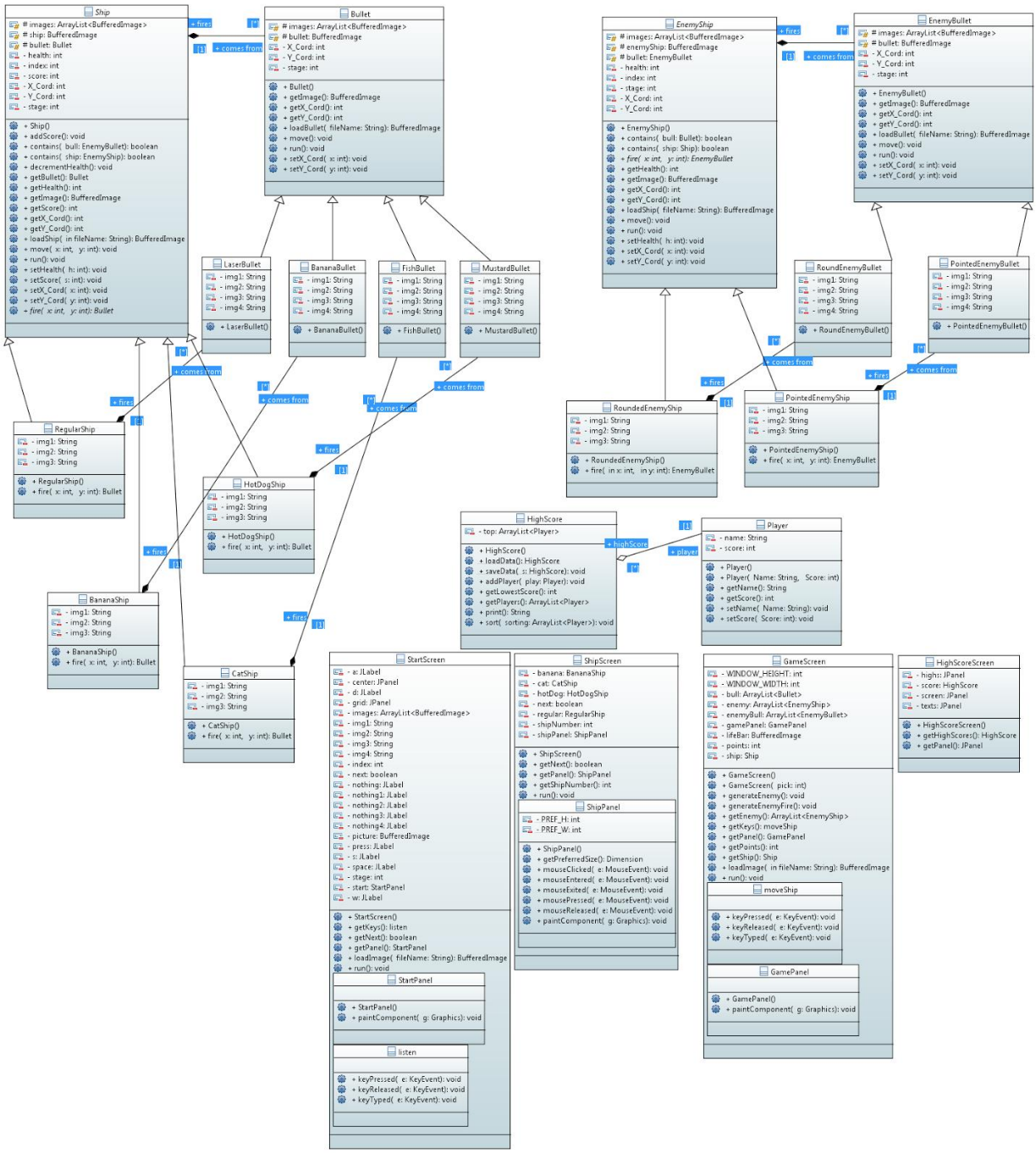
For the ships there are two types of ships: Player's ships and Enemy ships. Both type of ship classes implement runnable to handle the animation of the ship's pictures. There are four different ships that extend the Player's ship class (Ship): Regular Ship, Banana Ship, Cat Ship, and HotDog Ship. There are two different ships that extend the Enemy Ship class: Rounded Ship and Pointed Ship. Ship class and the Enemy Ship parent classes contain most of the methods, the subclasses contain the constructor that has the images for that particular ship and implements the abstract method fire which creates an instance of a bullet that is specific to that class.

There are two types of bullets: Bullets and Enemy Bullets. Both types of bullets implements runnable to handle the animation of the bullet's pictures. There are four types of Bullets which correspond to each different type ship: Laser Bullet, Fish Bullet, Banana Bullet and Mustard Bullet. There are also two types of Enemy Bullets which also correspond to each Enemy Ship: Round Enemy Bullet and Pointed Enemy Bullet. The parent Bullet classes contain the methods and the subclasses contain the constructor that loads the images for each bullet.

The High Score class which saves the player's name and their score to a file. The Player class which has a name and their corresponding score. The HighScore class will save data each time player get new high score and sort it into the new save data file then load it in the High Score Screen. The High Score screen will contain the

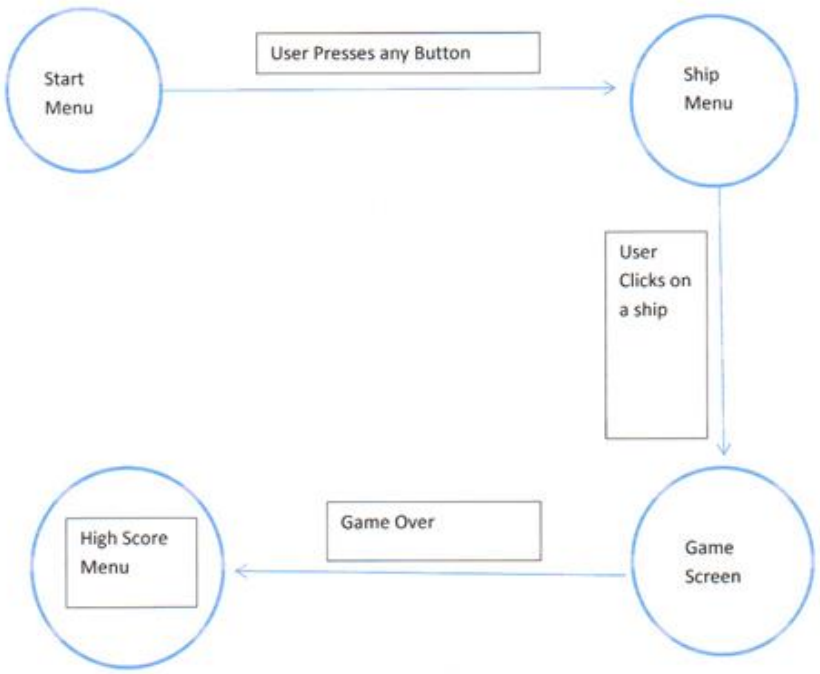
player's name and their score will be displayed from top to bottom sorted by score in a descending order.

UML:

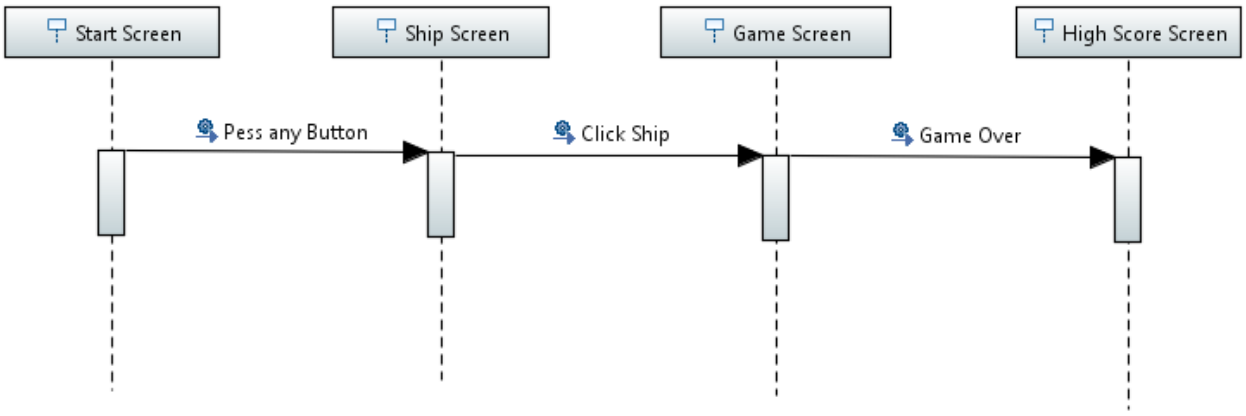


Interaction Analysis-

Case Model:



The user starts with the start menu, to continue and get to the next screen the user can press any button. Once at the ship screen the user clicks on the ship they want with the mouse and the user moves onto the next screen; the game screen. The user can use the W, A, S, D, and space keys to interact with the game and move the ship. Once the game is over, and if the user gets a high score a pop up screen will appear and the user can type in their name to be place on the high score screen. Then the names and scores of players will appear on the screen.



Boundary cases for the start screen are that you will not move onto the next screen unless you press a button. the boundary case for the ship screen id you will not move to the game unless you click on the ship, if you click anywhere else you will not move on. The boundary case for the Game screen is that you can't exit off the screen.

Implementation-

Implementation was hard due to all the different interactions between everything on the screen. Bullets had to interact with both the player's ship and the enemy ships. The player's ship has to interact with enemy bullets and the enemy ships. The enemy ship has to interact with the player's ship and bullets. All of this interaction has to have a lot of methods to make sure all interactions happen. Each object on the screen has to

have a contains method in order to know if something is touching that object. All object have a simple movement, usually left and right except for the ship which moves left and right and up and down. We didn't have enough time to make different movements for the different object. We also could not implement the boss class which since we ran out of time. Also since each ship fires a different bullet we had to implement an abstract method in the ship parent class to allow access to the fire method in the game screen. We wanted to keep one screen and not keep generating new screens, in order to do that we had to implement one main frame in the driver and just delete and add panels which contained the different screens.

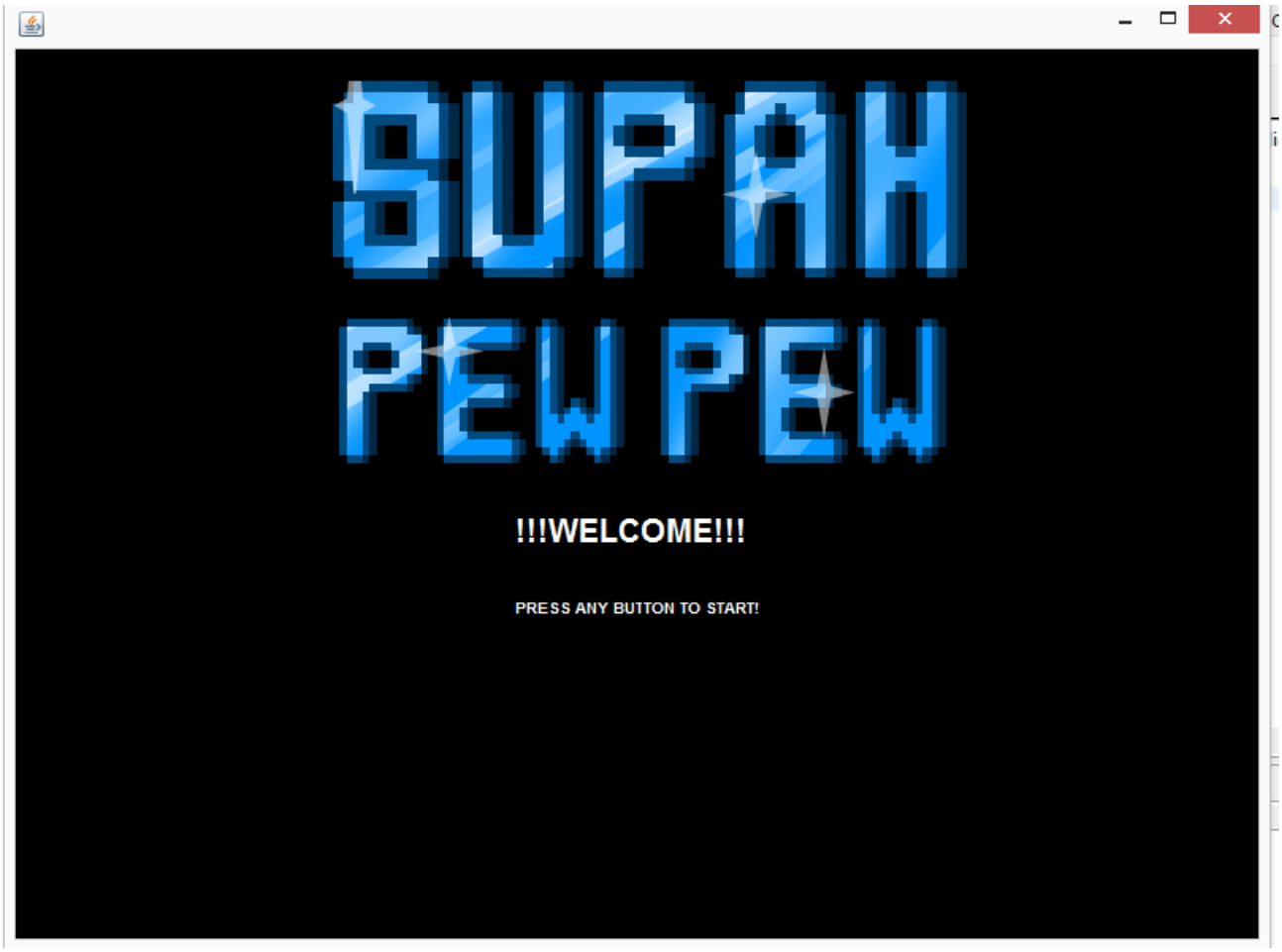
Testing and Results

Testing

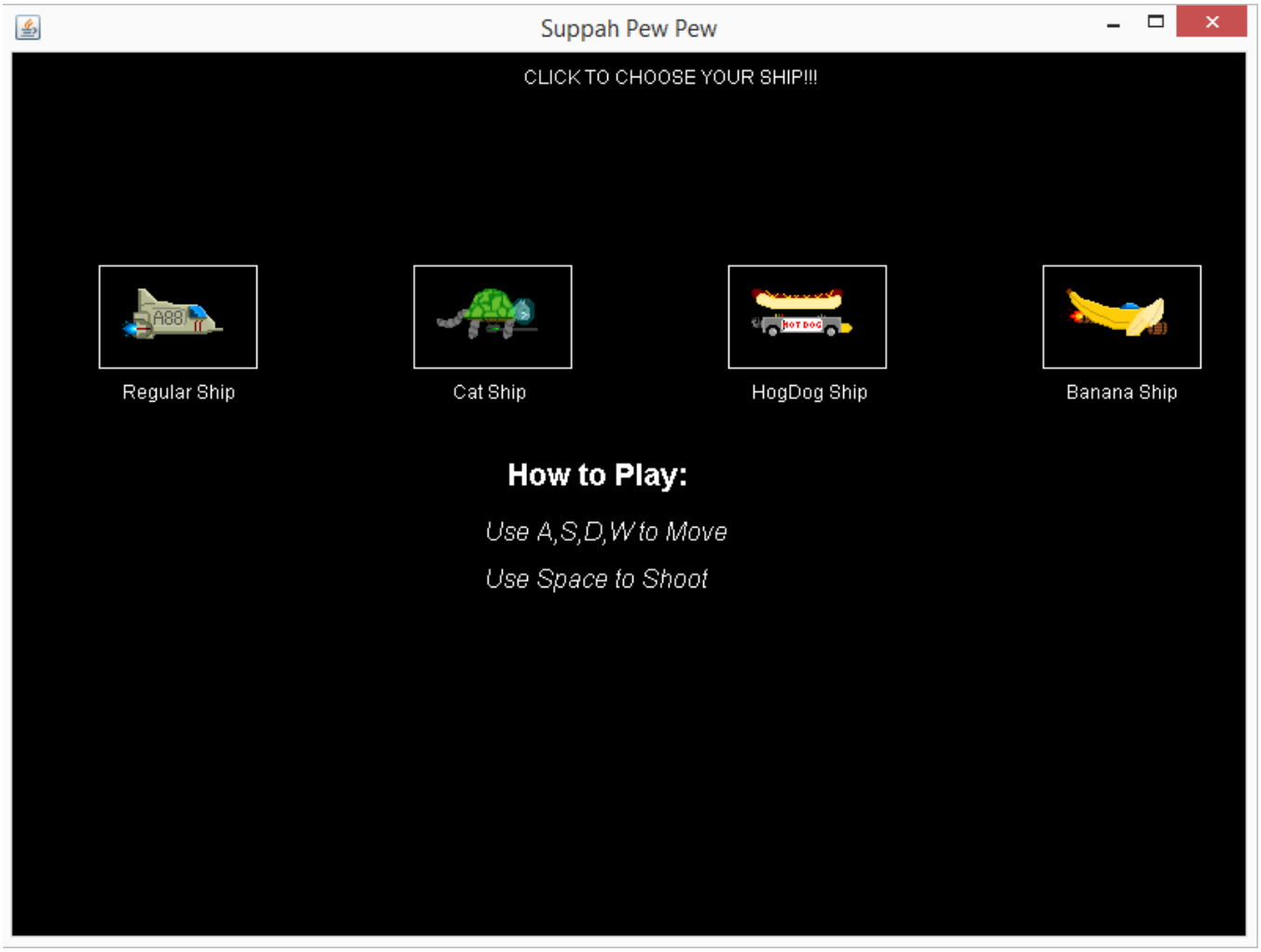
The game is run by Driver class that's help bringing each stage one by one from Menu Screen, then Ship Screen, after that the Game Screen start to play and finally finished by asking user typing their name and displayed on High Score screen. To test this we tried to run the program from the driver to see how each stages of the game can perform. Initially all the class run fine, just the display words and pictures stay in wrong coordinates, so after some update here the result.

Result:

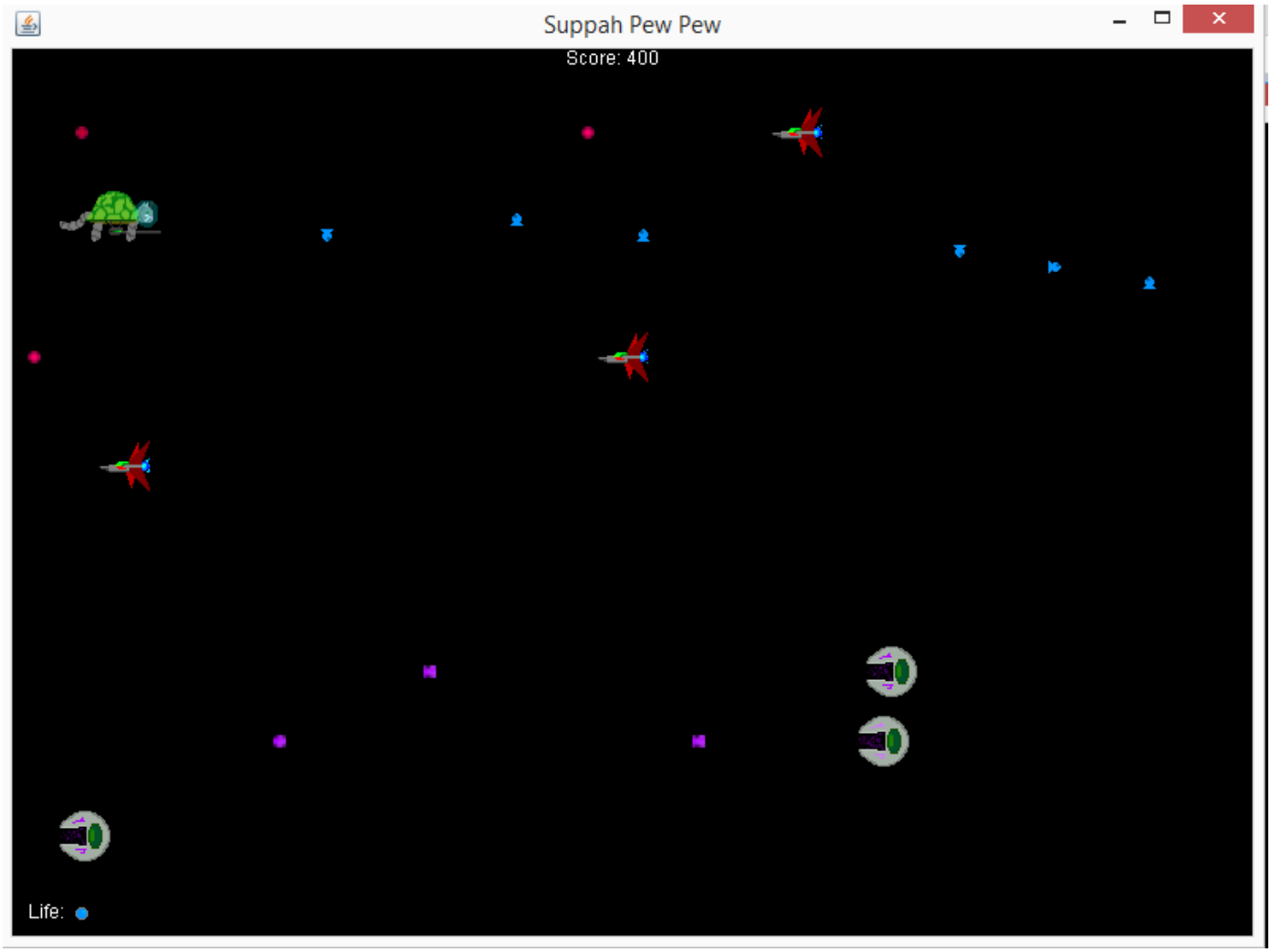
+Menu Screen:



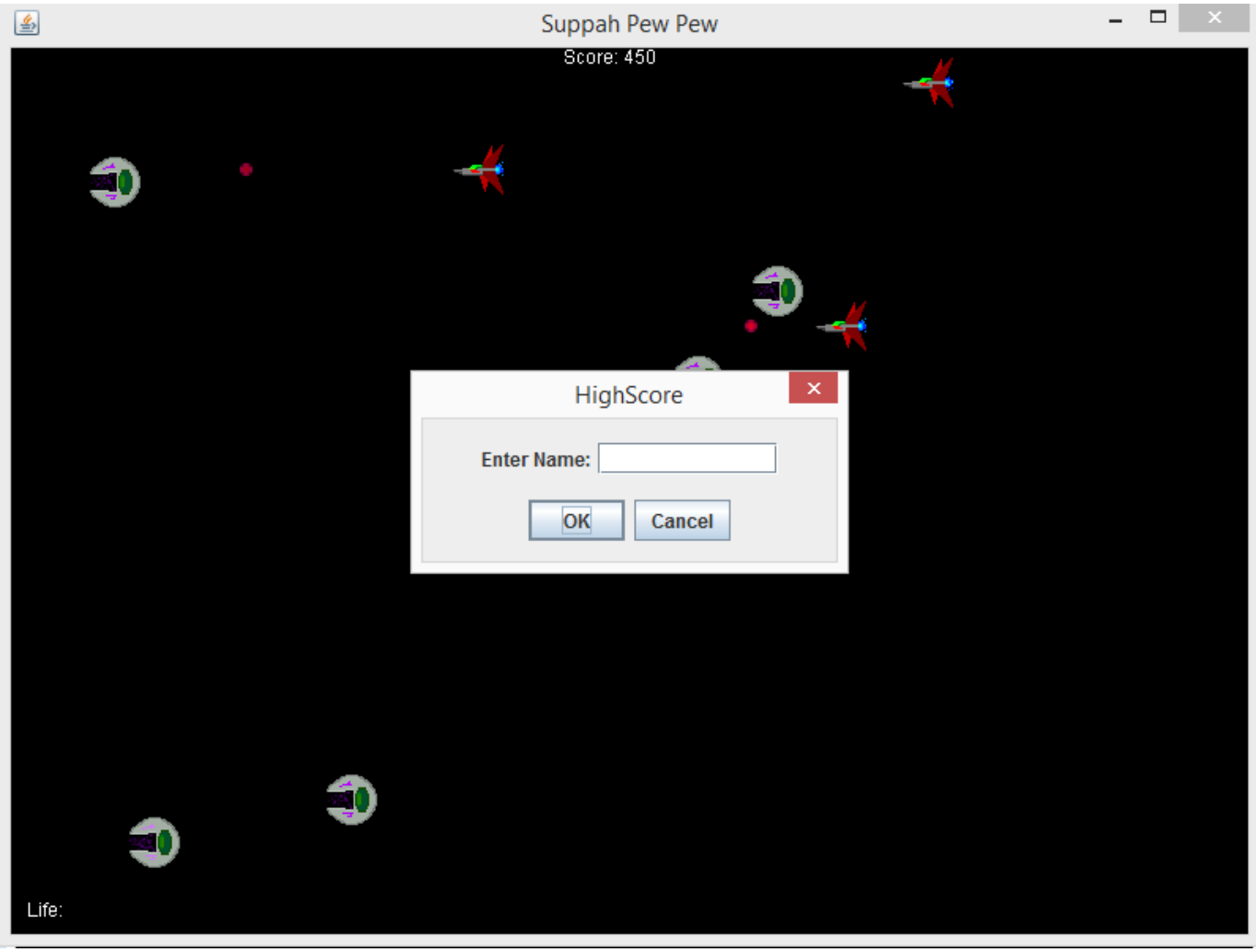
+Ship screen:



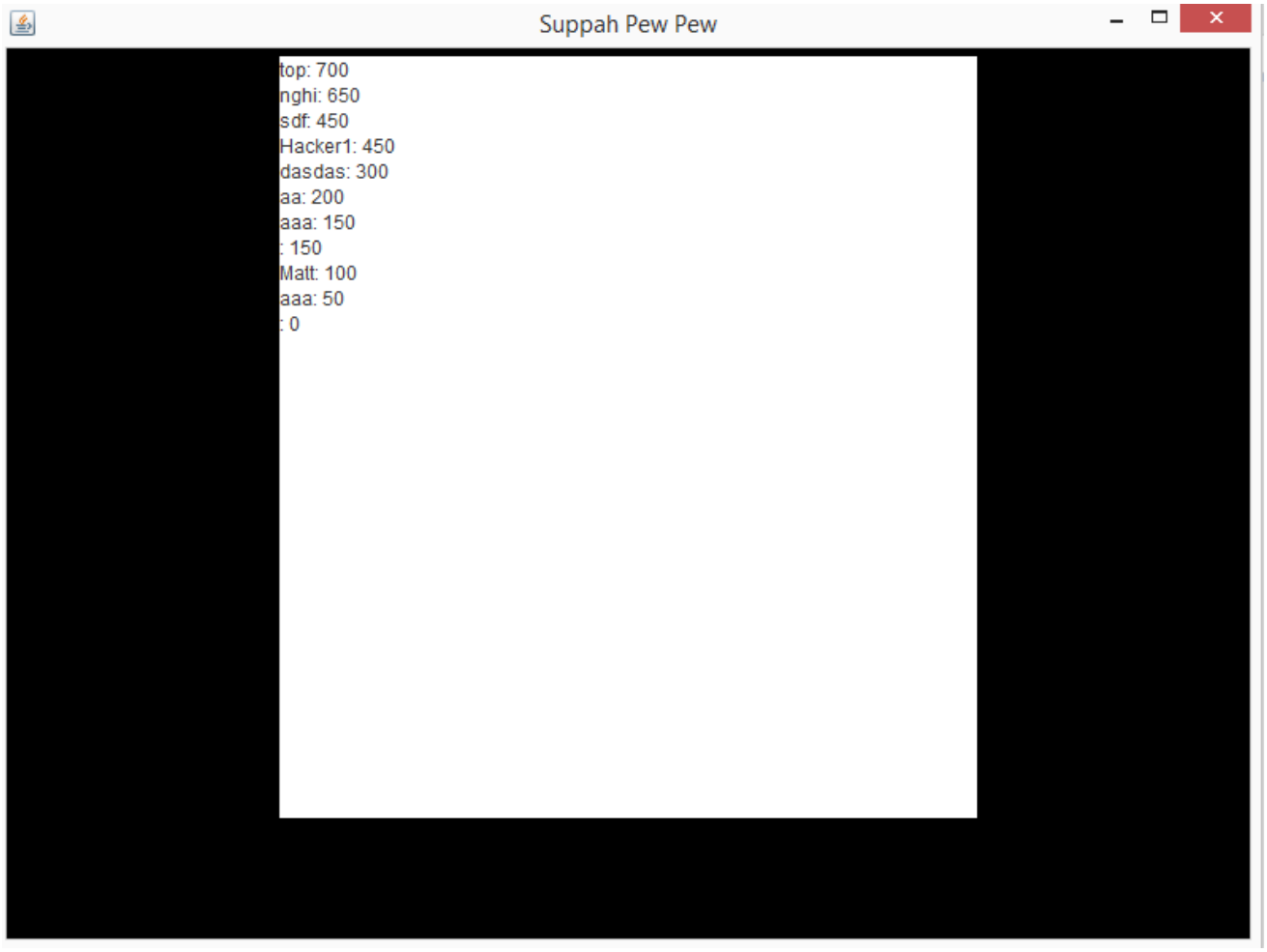
+Gameplay screen:



+Highscore screen:



+Highscore Screen:



Extensibility-

The program created for this project could be extended by add more animation about explosive ship when it get hits and background games look more in space rather than just only black screen. By adding the stages event to the game and implements some more classes like mini boss or time challenge could make the game more interested and more attracted to play. The game might be able can control by mouse also by adding mouselistener to the ship and user can move it freely in space and can shoot just by clicking Right or Left buttons. Also by create some more classes like Shop and Upgrade Store so the user can have more choice to upgrade their ships or

weapons and modify more stuffs as they like. For the HighScore Screen, it could be improved by adding Ranks, Title, and Scores columns and can be displayed with the ship's image that play by the player. Overall the program can run successfully by applying all the knowledge that learned in class.

Work-

We split the work evenly; each group member did ⅓ of the work. Julian Bostick drew all the pictures for the game and also wrote the all the player ship classes and also the player ship bullets. Nghi Truong wrote the start screen, that ship screen, the high score screen, and the player and the high score classes. Matthew Barragan wrote the game screen and also the enemy ship classes and the enemy ship bullet classes.