Developer's Manual

Introduction:
This program is written in C++ and uses SFML libraries. To run Space Mission 2200 do the following for Windows or Mac OSX:

Windows:

- Install the IDE Visual Studios. Finally install SFML by clicking here and following the tutorial for SFML on Visual Studio

Mac OSX:

- Install the IDE Xcode. Finally install SFML by clicking here and following the tutorial for SFML on Xcode.

General File Summary:

- *cScreen.hpp:* The run function determines if a screen can be displayed.
- *entity.hpp entity.cpp*: The entity hpp and cpp define objects that are used used many times through the code.
- *Main.cpp:* This file prepares all of the screens and runs the code. All the cpp, hpp files and the vector and string libraries are included. The window function setmousecursor(false) makes the cursor not display on the screen. A vector holds all the screen objects which changes the number associated with the screen value according to player interaction with the game. The screen value access a the corresponding screen obect by indexing to run its window.
- *missile.hpp missile.cpp*: The missile hpp and cpp files create two types of missile objects (regular missiles and stars). Missiles have movement physics controlled by update, rate of movement by missilespeed, damage dealt defined by int damage, and hit detection. Missile.update(int,int) takes 2 integers. The first integer controls its x-movement and the second controls its y-movement. The rocketship's missiles take in the following values for x and y: 0 and -missilespeed, because it will want to shoot upward. The other update function sets the the missile rect position on screen.
- *monsters.hpp monsters.cpp*: The monsters hpp and cpp create three minor monsters (dinosaur, yoshi, nyan-cat) and the boss monster. The minor type monsters have random spawn positions and each correspond to a different health and image file. The no arguments update function resets the position of the monster. The function update(int x, int y) moves the monster down the screen. The SFML graphics library and audio library are included.
- *Powerup.hpp Powerup.cpp*: Powerup files create the powerup block that enables the rocketship doubleshot. The update function updates missile position and restricts the direction of missile movement.
- *Rocketship.cpp Rocketship.hpp*: These files create the player's rocket and update its position and speed. The Rocketship constructor initiates the object's general characteristics as well as health, movement speed, and killcount. The update functions sets the position of the rockets rect. The updateSpeed function increase the speed of the

rocket and implements the keyboard controls. The SFML graphics library and audio library are included.

- *ResourcePath.hpp ResourcePath.mm*: These files make it easy for the IDE to access appropriate SFML Frameworks Files to run the game.
- *screen_0.hpp*: There are five sf::Text objects. Each text has font, character size, initiation of string to menu variable and position.The text used is sansation from sansation.ttf. If play = true, then your selections are “continue” or “exit.” Else makes choices “play” or “exit.” Keyboard input is taken, which allows user to select the text and the screen to change accordingly. The SFML graphics library is included.
- *screen_1.hpp:* The actual game is implemented into Screen_1.hpp. This is where we call all of the classes, such as the Monsters and Missiles. Clocks are also implemented to slow down movements because if the movements were based off the default ticks of the game, everything would be moving too quickly to control. Collisions between rocketship and monster, missile and monster, and missile and boss as well as others were accounted for in screen_1.hpp. To spawn random monsters, they are first pushed back into a vector according to a clock. Then the vector is iterated through with a for loop to draw the monsters out. The same format is done with the missiles. Score is incremented when missiles collide with the monsters. Health is decremented when monsters move pass the screen or when monsters collide with the rocketship. Levels are also implemented in this screen; they are changed based on user score. From this screen, screen_gameover.hpp, screen_congrats.hpp, and screen_0.hpp can be called. The SFML graphics library and audio library are included.
- *screen_congrats.hpp:* This screen draws three text objects using sf::Text. Keyboard input is taken, which selects either the text “Exit” or “Play Again.” Press enter to proceed accordingly. The SFML graphics library are included.
- *screen_gameover.hpp:* The gameover screen is similar to screen_0 in that it allows the user to restart the game or exit the game altogether. The only difference is that there is a sf::Text drawn to the window. The SFML graphics library is included.
- *screen_instructions.hpp:* This screen creates and initializes five instructions as sf::Text. The five instructions are then drawn into window. To proceed to the game(screen_1.hpp) press enter. The SFML graphics library is included.
- *Notes:*
    - The axis by default in SFML are formatted the up/down: -y,+y and right/left : +x/-x.
    - The screen functions are independently implemented because each run as a separate application in the game.