(Please read this first page before operation)

**\*\*\*IMPORTANT NOTICE\*\*\***

**This game was developed in Eclipse. Aludra does not run the game smoothly but it does, however, run the game. Professor Cheng has said a game may be graded in Eclipse, as long as the game ‘runs’ in Aludra, which it does. Please, I request my game be graded in Eclipse.**

**\*\*\*A NOTE ON IMAGES\*\*\***

**For my game to print, and display, properly, the ImageIcon locations must be correct. For this, I have provided the proper image files and included their extensions in the ImageIcon declarations. If you could please modify these to their proper location on your hard drive, that would be great.**

**Space Game Operating Guide**

Manual:

* Follow compile.txt instructions for running the game.
* Enter in a single word name to enter the database and begin.
* The game will pop-up, and your ship will begin to fly.
* Use the up and down arrow keys to move your shuttle up and down.
* Avoid asteroids, and collect green ammunition supply cases!
* Press spacebar to shoot bullets.
* To pause the game at any time, press ‘P’ and the game will pause. Press ‘P’ again to unpause.

Rules & Tips

* Player has 100 health.
* Bullets can destroy asteroids, and injure the boss.
* Asteroids will cause the player 20 damage if hit.
* Ammo cases will award 0-20 missiles if hit.
* The boss will fire missiles that will destroy asteroids, and cause the player 20 damage.
* Player wins by destroying boss, and reaching tunnel at end.
* Player will lose if:
  + Reaches end without killing boss
  + Dies from asteroids or missiles
* Final scores will be calculated and displayed at the end of each game.

Design Document

Description:

This piece of software is designed to register a user, and start a randomly generated scrolling space game. The main character is a ship, played by the user, which must avoid non-moving asteroids, and defeat a boss. The game will either lead to a win or victory, and a final score will be displayed.

Requirements:

This program is a side-scroller game. As such, the idea is to have a playable game, scrolling across a window, and an outcome that the player may either win or lose.

Some of the requirements can be listed as below:

* Non-moving objects: asteroids, and ammo cases
* Moving objects: bullets, missiles, player, boss, background
* Collision: items are affected and destroyed
* Player can pause the game
* Player enters username
* Final score must be calculated and displayed.

Class Description:

* Player
  + Int variables
    - Xcoord: players x value
    - Ycoord: players y values
    - Dx/dy, boardx/boardx2: incrementer values, used for change in x/y and board scrolling
    - Playerlife: player’s remaining health, initializes to 100
    - Ammunition: player’s arsenal remaining, initializes to 50
    - winCounter: used to increment on clocks, and tell whether or not the player has reached the final destination
    - playerScore: players final score
  + ArrayList<movable> bullets: the array is used to store all of the moving bullet objects in the players arsenal
  + Image: players icon image
  + Boolean isPaused: variables used to pause the game state
  + Move()
    - Function is called every clock to move the shuttle.
    - Shuttle will increase player’s xcoord until x=150, at which point the user may only move his ship up and down, and the background begins to scroll.
    - Shuttle will respond to player’s keystroked on whether to move up or down
    - Board x will increase to make the board background scroll
  + Fire()
    - As long as the player has ammunition left, it will generate a new bullet, which will begin to move forward
  + keyReleasd()
    - registers to stop the spaceship from moving up or down when the player lets go of the key
  + keyPressed()
    - changes dy to +/-1 depending on the player’s up/down
  + bounds()
    - returns rectangle that encloses it’s entire image, used for collision detection
  + hit()
    - function is called when the player has come into contact with an asteroid or missile
    - playerlife is decreased by 20
    - system prints out notification to the user
    - if the player’s life falls to 0, the game will quite, and inform the user of his score
  + refill()
    - function is called if player runs over an ammo case
    - randomly calculates a number between 0-20 and adds it to the players ammunition
    - players is informed of ammunition gained.
  + returnScore()
    - Function called before quitting program to display user’s final score
    - Score is calculated based on distance travelled, and bullets picked-up/saved.
* Application
  + Main()
    - Asks user to log-in to the system before beginning the game
    - User must enter unique one word username
    - System will enter into database and start the game
    - Applet will be created, and the game will automatically start for the player
    - All frames are created, and added at this stage
  + \*The application class is used to handle the user, and console input. Once they sign in , and the game begins, the game class takes over for all game logic.
* Immovable
  + Xcoord/ycoord: values for asteroids x and y
  + Image: stores asteroid icon
  + Move()
    - Asteroids are stationary, so if they player is moving (but actually staying at x-coord 150) we must decrease this xcoord by 1 so as to simulate the player moving through the field.
    - Xcoord is decremented once every clock
  + Bounds()
    - Returns rectangle with proper dimensions of icon, used for collision detection
* Movable
  + Inherits immovable
  + Redefines image to proper bullet icon
  + Move() is overridden to:
    - Increase xcoord by 2 every clock, so that the bullet is shot forward towards the enemy at a quick pace.
  + Bounds()
    - Returns rectangle with new icon dimensions for collision detection
* Boss
  + Inherits movable
  + Int variables
    - Shootcounter: used to increment, so that modulus may be used for every 100th clock, and a missile is shot towards the player
    - Bosshealth: initialized to 2000, and is used to detect whether boss is still alive
    - resetCounter: incremented on every clock , but used as a temp timer to display an alternate image when the boss is hit, thus simulating a ‘blink’ so the user knows of a hit.
  + ArrayList<Missile> missiles: used to store the array of missiles the boss has shot at the player at any given tiem
  + Boolean isAlive: used to return to the game function whether the boss is dead or not, and thus allow the player to win or lose
  + Move()
    - Boss will begin 3000 spaces in, and xcoord will decrement by 1 until xcoord = 600;
    - If boss is at xcoord 600 (and thus is displayed on the screen), boss travels in y direction by -1, until knocking upper bound, and travels in y direction by 1 until knocking lower bound
    - Boss thus flies up and down the right side of the screen
    - As boss flies, missiles are fired every 100 clocks.
    - Missiles are created, and shot towards player, thus being added to missiles array
  + Bounds()
    - Returns rectangle with proper dimensions
  + Hit()
    - Responds to a player’s bullet hitting the boss.
    - Boss life is decremented by 200 for each hit
    - If the boss falls below 0 health, he will ‘die’ and be deleted, and so will all of his missiles
* Game
  + ArrayLists
    - <Immovable> Asteroids: represents randomly generated aseteroid field
    - <Ammo> ammoRefills: represents randomly generated ammo cases
  + spaceship: initialization of class Player
  + boss: initialization of boss
  + MyActionListener()
    - Responds to timer delays and signals representing a clock, and calls actionPerformed()
  + KeyActionListener()
    - Extends keylistener
    - Responds to user input to the keyboard (space, up, down, P), and passes the key action along to the Players specific key action functions
  + Game()
    - Variables are initialized
    - Players is placed in middle of frame
    - All random positions of asteroids and ammo cases are generated
    - Tunnel for ending is created
    - Timer is started and game begins
  + actionPerformed()
    - case for winning is checked
      * victory displayed
      * score printed
    - case for losing
      * loss displayed
      * score printed
    - spaceship moves accorind to user button presses
    - asteroids all decrement 1 and check for collisions
    - all ammo cases decrease and check for player running over them
    - all current bullets fired move right by 2, and check for collisions
    - repaint is called every clock
  + paintComponent()
    - background is printed first
    - background is scrolled based off of internal iterators being checked by % operation after every 800 pixels.
    - Spaceship is scrolled forward first 150 pixels
    - After first 150, spaceship is scrolled up or down
    - Asteroids are drawn, and deleted if they are drawn off the window
    - Ammo refills are printed across the board and deleted if they move off the window
    - Bullets are printed, and deleted if they move off right edge of window
    - Boss is drawn when he comes into range
    - Boss’ missiles are drawn and deleted if drawn off the window
    - Player life, ammunition, and pause status are displayed
  + Various collisions()
    - Create rectangle containing first object’s size
    - Create second rectangle containing second object’s size
    - Check for rect1 intersecting rect2
    - Delete collision items, and properly call hit() functions

Test Cases:

* Console input is tested for:
  + Various single input text strings
  + Usernames including integers
  + Proper input, and initiation of game after
* Game is tested by:
  + Running player into many asteroids, and checking for proper “collision”
  + Running player over numerous ammo cases to check whether ammo is increased randomly
  + Bullets are fired at asteroids to check for deletion
  + Boss is fired at to check for proper collision, and boss is killed many times to find errors in handling boss life.
  + Player is ran into boss missiles numerous times
  + Player properly loses game if:
    - Dies
    - Reaches end without killing boss
  + Check for conditions leading to win
    - Kill boss – works
    - Reach end – works
  + Check scoring
    - Further distance = higher score
    - More bullets = higher score
  + Check for player bounding, whether you can move off the border.
  + Check for any accidental cases leading to runtime error
  + Check player can pause
  + Check pause state leaves all variables in their proper places
  + Check overall look and feel