2102

Space Shooter

Detailed Design

COP 4331: Processes for Object Oriented Software Development
Fall – 2012

UCF
Team 2
10/5/2102

Table of Contents

Prefatory Information

Modification history:

Version	Date	Who	Comment
v0.5	10/7/2012	Thaddeus Latsa	Started Documentation
v0.8	10/8/2012	Thaddeus Latsa	Finished Documentation and Requirement Tracing
v1.0	10/8/12	Thaddeus Latsa	Finished Initial Draft
v1.1	10/8/2012	Joshua Thames	Some formatting
V1.2	10/9/2012	Thaddeus Latsa	Minor Revisions

Team Name:

Team 2

Team Members:

Name	Email address
Andre Meireles	andre.meireles@knights.ucf.edu
Alex Banke	banke@knights.ucf.edu
Christopher Margol	margol_chris@knights.ucf.edu
Chris Lin	christophercklin@gmail.com
Joshua Thames	jthames88@knights.ucf.edu
Thaddeus Latsa	tlatsa@knights.ucf.edu

Design Issues:

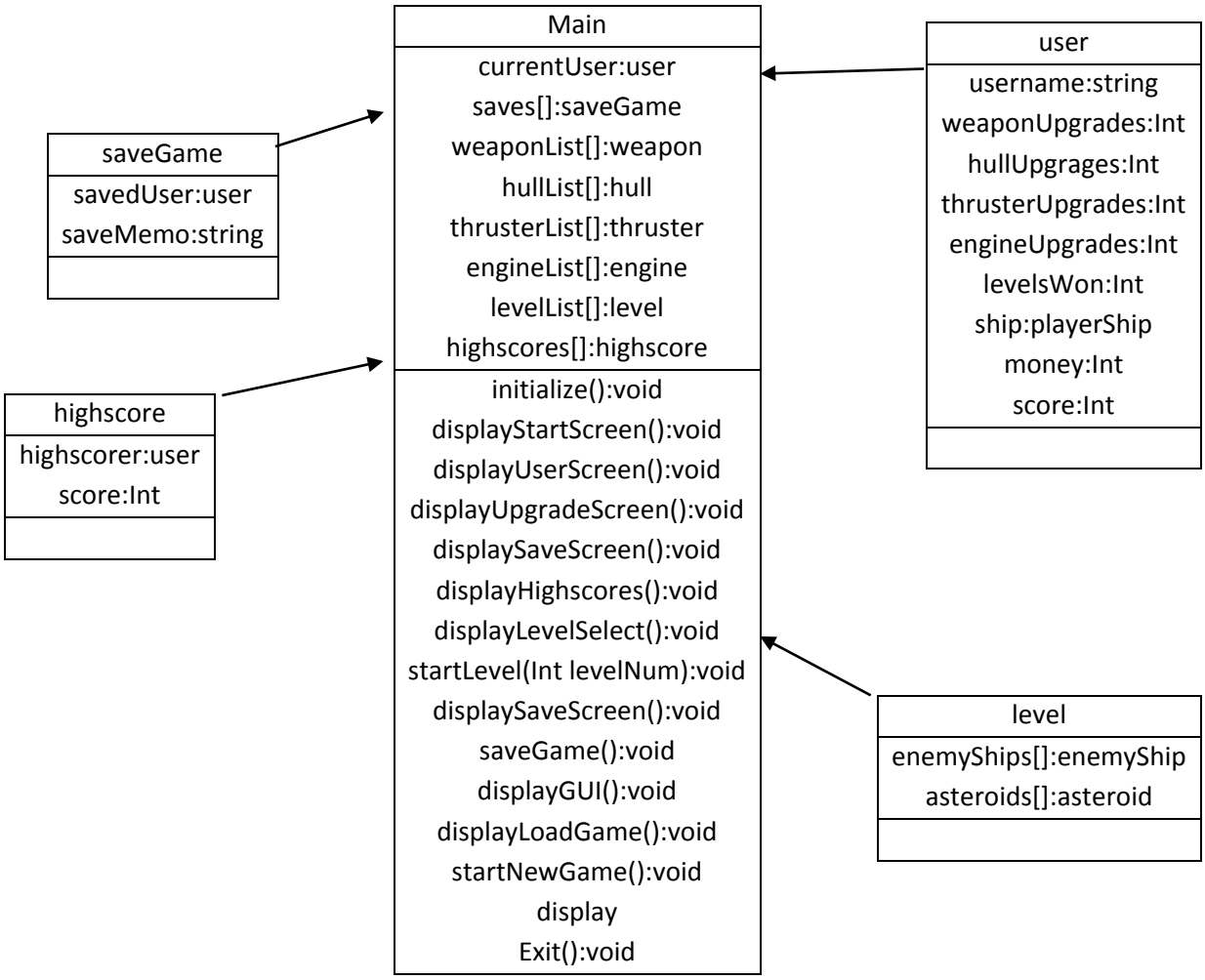
- Reusability
    - This will be a one-time project in which we will not develop further after this class project
    - However, the components in the system will be independent enough so that we can perhaps use the modularity of the system for personal research in the future
- Maintainability
    - Maintain high score tracker within local entity on hard disk
        - This is done to reduce complexity as well as
    - Must maintain the project website for downloading software and user notes
    - Once system is completed and before the final package is sent, we will still continue to check for errors and update system appropriately
- Performance
    - Our goal is to maintain 24 frames per second even on low end computers
        - Our goal is for everyone to have a good gaming experience.
- Portability
    - Send the .jar or .exe to grade
        - This is done so it is easy for any user to use the product.
        - Risk: Program in compiled form has errors
    - We would like any computer with Java to be able to run this program
        - This is done to increase the consumer base.
    - As of right now, we will not plan for Android or Mac support unless we have an abundance of time
- Prototypes
    - We will be using the incremental phased development approach so each new functionality or class will be a new demo/prototype
        - This allows us to work on each functionality individually, which allows for easier testing and gives us a good idea of progress.
- Technical Difficulties
    - Getting graphics/sound and implementing them in the system
        - Thad and Chris Lin are responsible for graphics and sound
        - They will possibly consult existing Java libraries for graphics and sound
        - For any additional materials beyond their capabilities, Thad and Chris will consult UCF digital media design students
    - Designing an appropriate physics engine to conform to the system requirements
        - We will search for existing Java libraries to simulate our required physics and modify them for our specific needs

    - Getting Git and Github to work as expected
        - Keep trying to figure out the bugs
        - Consult tech support
- Trade-offs in design architecture
    - No Android or Mac support because of time and complexity
    - Simple ship upgrade methods as opposed to a more detailed and graphic intensive alternatives. We would like to use detailed graphics to represent different ships and weapon options, however, because of time demands and possible conflicts with collision detection, we will opt for simple ship model representations. If time and expertise allows, we will use higher quality models, sounds, etc.
    - We are going to use pre developed Java libraries and modify them to our needs. We would all like the experience in developing our own.
- Rationale for this architecture
    - We are going to use the incremental phased approach which will ensure that we have one functional/working system at all times, improve that system with new functionality, insuring that new functionality will be working before we move to the next phase.
    - There is a limited amount of time, limited experience, and many things that we desire to implement. Our highest priority is getting the minimum requirements done as quickly and efficiently as possible. We wish to focus most of our energy on the design process, therefore, it is counterproductive implement many features outside of the minimum requirements.
- Risks with this solution
    - Using too much of another developers code could result in lazy programming techniques, poor code documentation, or plagiarism.
    - We could get hung up on developing one facet of the program and not move to the next in time to complete the assignment.

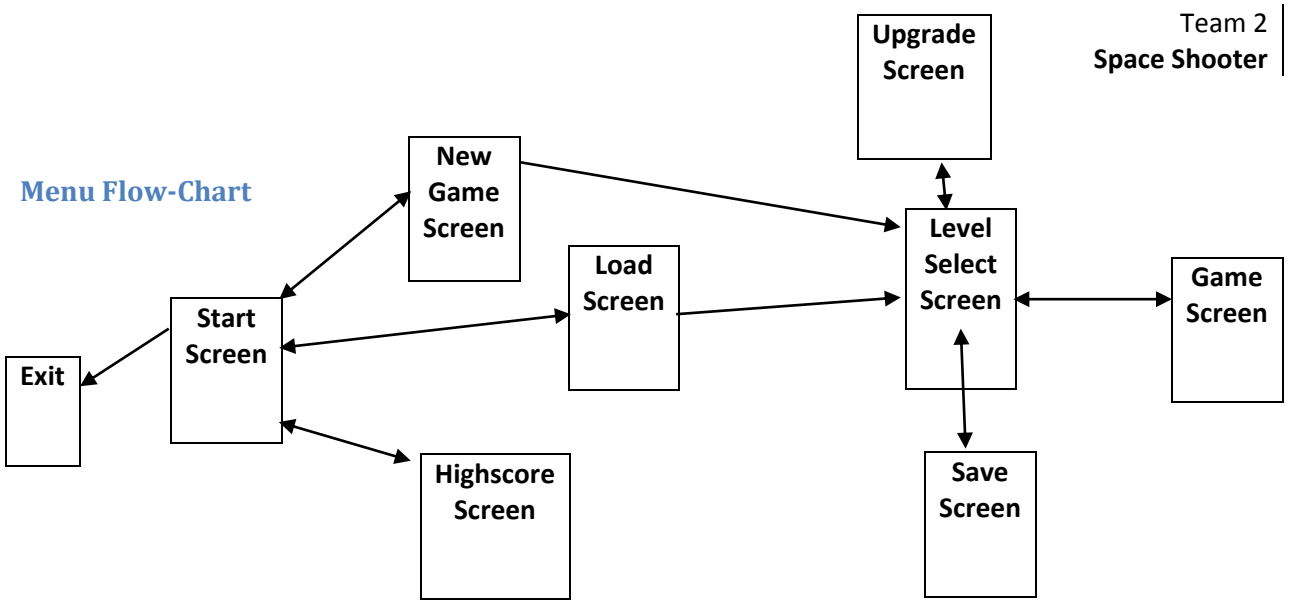
Detailed Design Information

The program will utilize a multitude of classes to accomplish its goals. The classes are as follows; Main, user, level, saveGame, highscore, object, playerShip, bullet, asteroid, enemyShip, upgrade, weapon, hull, thruster, and engine. Below are class diagrams to illustrate how the classes are connected, and documentation for each class.

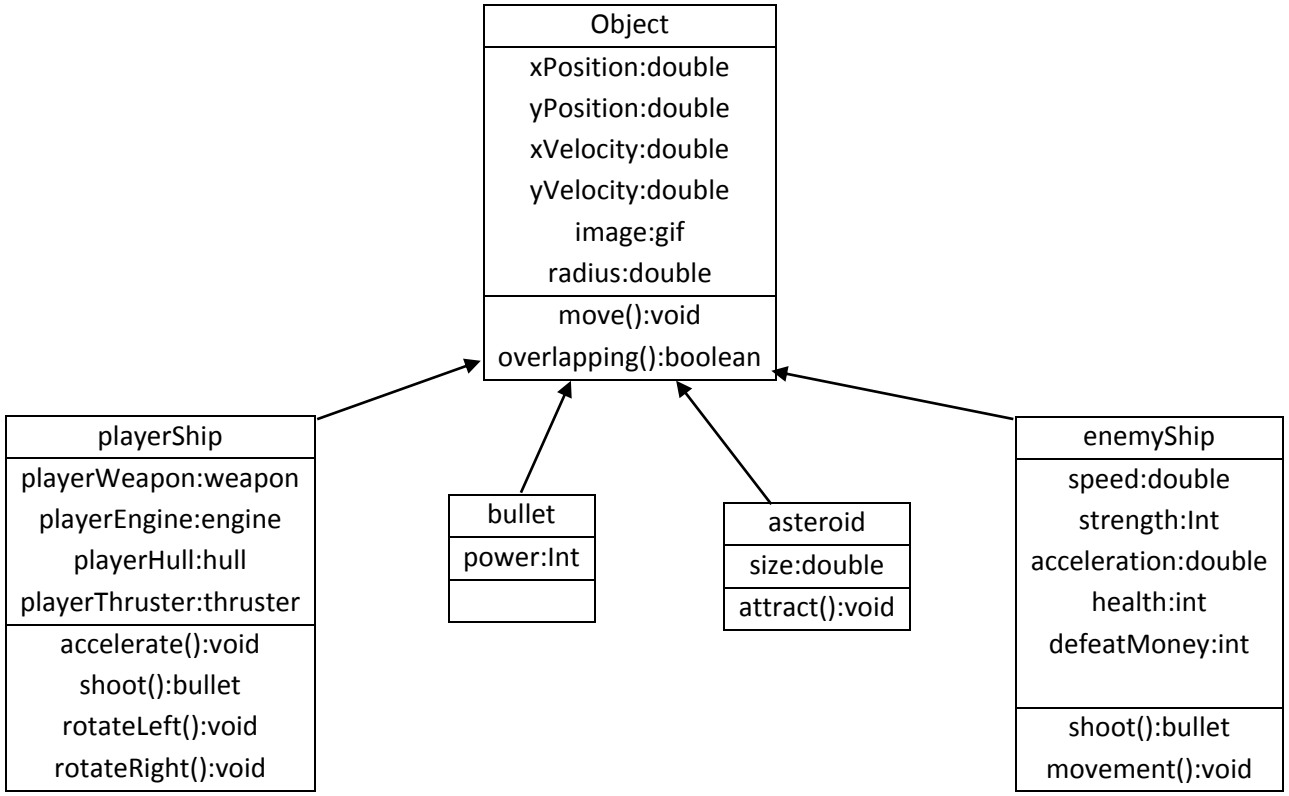
Class Diagram of Main Class



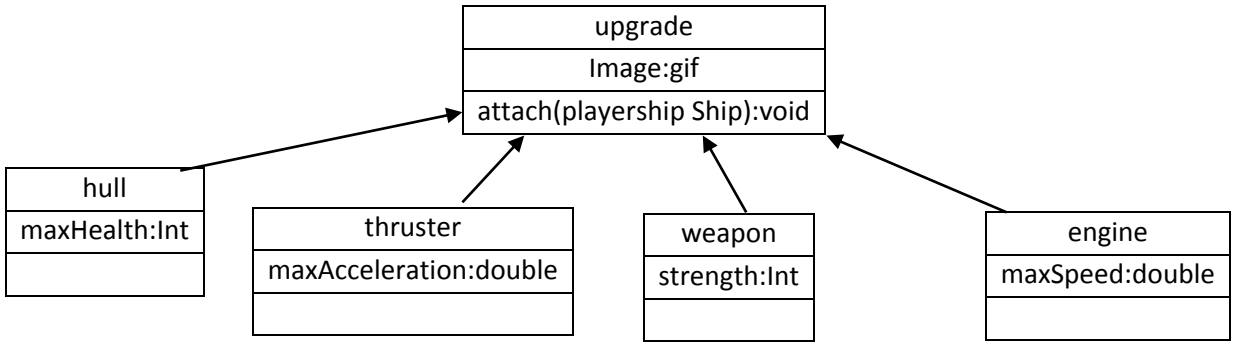
Menu Flow-Chart



Class Diagram of Object Classes



Class Diagram of Upgrade Classes



Class Documentation

Main

- Fields
  - highscores[]:highscore
    - An array of all the highscores on the current system.
  - currentUser:user
    - The current User loaded into the game
  - saves[]:saveGame
    - An array of all of the saved games.
  - weaponList[]:weapon
    - A list of all the weapons in the game
  - hullList[]:hull
    - A list of all the hulls in the game
  - thrusterList[]:thruster
    - A list of all the thrusters in the game
  - engineList[]:engine
    - A list of all the engines in the game
  - levelList[]:level
    - A list of all the levels in the game

- Methods
    - initialize():void
        - Checks the hardware to see if it can handle the work. If not, displays a message warning the user about this. Loads information from a file saved on the computer pertaining to the saved games, users, and highscores. If such a file cannot be found, ask the user to create new save information.
    - displayStartScreen():void
        - Display the menu screen. Clickable options available will be New Game, LoadGame, Highscores, and Exit, which will trigger methods startNewGame(), displayLoadScreen(), displayHighscores(), and Exit() respectively.
    - displayUpgradeScreen():void
        - Display the upgrade screen. A list of upgrades available will be shown. The user will be able to purchase upgrades for their ship. A back button will be shown which will trigger the displayLevelSelect() method.
    - displaySaveScreen():void
        - Display the save screen. 3 slots will be shown. The user's data will be saved into the slot that they select. A back button will be shown which will trigger the displayLevelSelect() method.
    - displayHighscores():void
        - Display the highscore screen. A list of all the highscores will be shown. A back button will be shown which will trigger the displayStartScreen() method.
    - displayLevelSelect():void
        - Display the level select screen, which will act as the user hub. A list of levels will be displayed, clicking on them will trigger the startLevel() method. Other buttons include the upgrade button, the save button, and the log out button, which will trigger the displayUpgradeScreen() method, the displaySaveScreen() method, and the displayStartScreen() method respectively.
    - startLevel(Int levelNum):void
        - Starts the level using the levelNum integer as index in the levelList[] field to locate the level from the list of levels. The displayGUI() method is triggered and continues until the player is victorious or defeated, at which point the displayLevelSelect() method is triggered.
    - saveGame():void
        - Creates a save in the chosen index of saves[] with the user information found in the field currentUser. The program then creates or edits a text file that contains all the information found in the saves[] field that will be used in the initialize() method.

- displayGUI():void
  - Displays the game Gui until the player is killed, all enemy targets are destroyed, or the player exits voluntarily. The gui will include a healthbar, a score counter, and all the graphics.
- displayLoadGame():void
  - Display the load game screen. A list of saved games will be shown. Clicking on them will change the current user to the one found in the saved game. When one is selected, the displayLevelSelect() method is triggered. There is also a back button which triggers the displayStartScreen() method.
- startNewGame():void
  - Prompts the user to enter a new username. The user may select a back button, which triggers the displayStartScreen() method. A new user is created using the username given if one is provided. The current user is switched to this user. A tutorial on how to play the game will follow. The displayLevelSelect() method is triggered.
- Exit()
  - The system saves all the information found in the saves[] field, and highscores[] field into an external file for future retrieval. The game exits.

User

- Constructor
  - user(string name)
    - creates a user with all fields at 0 with the given username
- Fields
  - username:string
    - The user's name
  - weaponUpgrades:Int
    - The number of weapon upgrades the user has purchased.
  - hullUpgrages:Int
    - The number of hull upgrades the user has purchased.
  - thrusterUpgrades:Int
    - The number of thruster upgrades the user has purchased.
  - engineUpgrades:Int
    - The number of engine upgrades the user has purchased.
  - ship:playerShip
    - The user's current ship, for easy access for loading.
  - money:Int
    - The amount of money the user currently has
  - score:Int

        - The score the user currently has.

Level

- Constructor
    - level()
        - Creates a blank level
- Fields
    - enemyShips[]:enemyShip
        - A list of all enemy ships and pertinent information in the level.
    - asteroids[]:asteroid
        - A list of all asteroids and pertinent information in the level

SaveGame

- Constructor
    - saveGame(string Memo)
        - Creates a save with the current user information with a memo to help the user remember the save.
- Fields
    - savedUser:user
        - The user of the save file
    - saveMemo:string
        - A string for the user's purpose

Highscore

- Constructor
    - highscore(user User, int score)
        - Creates a highscore of score pertaining to the given user.
- Fields
    - highscorer:user
        - The owner of the highscore
    - score:int
        - The score

Object

- Constructor

    - object(double x, double y, double xV, double yV, gif img, double r)
        - Creates a new object at x, y with velocity xV and yV, with image img and radius r
- Fields
    - xPosition:double
        - The xPosition of the object
    - yPosition:double
        - The yPosition of the object
    - xVelocity:double
        - The xVelocity of the object
    - yVelocity:double
        - The yVelocity of the object
    - image:gif
        - The visual image of the object
    - radius:double
        - The radius of the object
- Methods
    - move():void
        - Using the x and y velocity fields of the object, calculates the next x and y position of the object.
    - overlap(object obj):boolean
        - Using the radius, determines if two objects are overlapping or not.

PlayerShip

class playerShip extends object

- Constructor
    - playership(double x, double y, double xV, double yV, double r, weapon w, engine e, thruster t, hull h)
        - Creates a player ship in designated location with designated equipment
- Fields
    - playerWeapon:weapon
        - The equipped weapon
    - playerEngine:engine
        - The equipped engine
    - playerThruster:thruster
        - The equipped thruster
    - playerHull:hull
        - The equipped hull
- Methods

- accelerate():void
  - Increases the x and y velocity based on the max acceleration
- shoot():bullet
  - Creates a bullet at the weapons position of the ship, using the equipped weapon to determine the power of the shot.
- turnLeft():void
  - Changes the way the ship is facing counterclockwise
- turnRight():void
  - Changes the way the ship is facing clockwise

Bullet

class bullet extends object

- Field
  - power:Int
    - How much damage the bullet will do when it collides with something
- Constructor
  - Bullet(double x, double y, double xV, double yV, gif img, double r, int power)
    - Creates a bullet at designated location with designated power

Asteroid

Class asteroid extends object

- Methods
  - attract(object obj):void
    - The asteroid attracts all objects in the level with a force proportional to its size and inversely proportional to the distance between them.

EnemyShip

class enemyShip extends object

- Fields
  - speed:double
    - The speed of the ship
  - strength:int
    - The power of the ships bullets

    - acceleration:double
        - The acceleration of the ship
    - health:int
        - The max health of the ship
    - defeatMoney:int
        - The amount of money earned when the ship is destroyed
- Methods
    - shoot():bullet
        - The enemy ship creates a bullet at the location where its guns are located.
    - movement():void
        - Contains a thread that details the enemy ships movement and behavior.

Upgrade

- Fields
    - image:gif
        - The image of the upgrade
- Methods
    - attach(playership Ship):void
        - Exchanges this upgrade with the one currently on the ship

Weapon

class weapon extends upgrade

- Fields
    - power:int
        - How much damage the bullet deals.

Hull

class hull extends upgrade

- Fields
    - maxHealth:int
        - How much health the hull provides.

Thruster

class thruster extends upgrade

- Fields

- maxAcceleration:double
  - How much acceleration the thrusters provides.

Engine

class engine extends upgrade

- Fields
  - maxSpeed:double
    - How much speed the engine provides.

Trace of Requirements to Design:

3.1 Functional Requirements:

- F1 - GUI
  - The Main class has a gameGUI method which is used to show the game gui.
- F2- Menu
  - The Main class has a multitude of menu methods which are intended to provide a multitude of different functions.
- F3- Gameplay
  - The gameplay basics are found in the object class and the player ship class
- F4-Website
- F5-High Score Database
  - The High Score Database will be on the user's systems, as described in the highscore field in the Main class

3.2 Interface Requirements:

- I1- Menu
  - The Main class has a multitude of menu methods which are intended to provide a multitude of different functions
- I2- High Scores
  - High Scores are implemented as described in the display Highscore Method in the Main Class

3.3 Physical Environment Requirements:

- P1-Hardware

- The initialize method in the main class checks if the hardware is sufficient for this program.

3.4 Users and Human Factors Requirements:

- UH-1 User must be able to operate a keyboard.
  - The ship movements are activated by keyboard

3.5 Documentation Requirements:

- D-1 Code Documentation
  - Code Documentation is found in the previous section

3.6 Data Requirements:

- D-1 Gravity
  - The gravity functionality is found in the asteroid class
- D-2 Scoring
  - The score is found in the user class

3.7 Resource Requirements:

- R-1 Java
  - The code documentation is in a Java format