Basic Programming Project Report: Hyperspace Hazard

Gilles Jacobs and Vincent Vercruyssen

1. Setting and how to play

a. Story/setting

The Alpha Centauri Conglomerate has received word from a rebel uprising in their mining territories. Suppress this rebellion and DESTROY the rebel MOTHERSHIP in order to render their fleet inoperable. Before you reach the mothership you will have to face many dangers, such as enemy ships and the hostile environment of deep space. Do not even think about taking on the mothership before you have sufficient offensive and defensive capabilities. You can upgrade your shields and weapon systems by fighting and mining your way through the galaxy. Good luck, captain!

b. How to play:

NAVIGATION: Jump to nearby sectors by clicking on the tiles next to your ship. Only the tiles that directly surround your ship are reachable! Your radar can only scan a set radius of sectors around your ship. Most of the map is invisible!

HP: Your ship integrity goes down by taking damage in fights. You die a fiery death when your hitpoints reach zero. Prevent taking damage by upgrading your shields' DEFENSE and having them absorb enemy damage!

FUEL: When navigating the galaxy you will have to make sure you do not run out of fuel. Every jump to a nearby sector consumes some fuel. Additionally, navigating in the neighborhood of BLACK HOLES lowers your fuel level.

OFFENSE: Fight enemy ships by clicking on them when you are near. You can do more damage with increased OFFENSE, you take less damage with increased DEFENSE. BEWARE! Some enemy types are stronger than others!

DEFENSE: Your shields protect you from enemy attacks. They get damaged by radiation from a STAR or by navigating near ASTEROID FIELDS.

EXPERIENCE: Increase your OFFENSE and DEFENSE by upgrading your weapons systems and shields by using experience earned by mining.

MINING: You can mine PLANETS, MOONS and ASTEROIDS by clicking on them when you are near. These have different yields each. You can choose to mine them for FUEL or for EXPERIENCE.

2. Class descriptions

In this section we will give an overview of all classes used in the project. We will briefly describe their functionality without going into too much detail. We designed our project according to the MVC design pattern. The packages View, Controller and Logic correspond to this paradigm and the functionality of classes within these packages adhere to the MVC design as rigidly as possible.

Controller package:

Bootstrap:
This class contains the main method and only initiates the GameController. It is thus the primary class that launches the game.

GameController
GameController is the main controller that coordinates the different GUI elements and the various logic classes that are connected to these GUI elements. It initializes the necessary listeners (StartLogic, GameLogic, and EndGameLogic) and two main GUI elements (StartScreen and GameScreen, both JFrames). It contains methods that create and switch out instantiations of the different GUI classes. It

transfers the necessary information set in the StartScreen to the GameLogic controller class where it can be used to initialize the game. It also contains methods that can be called to update or reset the stats and the gameboard and return to the start screen. The controller thus makes sure that all the parts of the game work seamlessly together.

GameLogic
GameLogic is the controller class that is responsible for coordinating the gameboard GUI and the game logic. It receives StartScreen settings information (passed on by the controller) and uses this to instantiate all the necessary objects of classes in the logic such as: the player object (Player class), the galaxy object (which is an abstract 2D-array representation of the gameboard realized by the Galaxy class), asteroids, enemy ships... In short, it instantiates all the objects that are necessary to play the game. It contains the method CheckGame which continously checks for win or lose-states. The gameOver method and gameWon methods are called when the player wins or loses, opens an instance of the EndGameScreen GUI class and then goes back to the StartScreen by means of the showStart method (which calls upon the controller to switch between screens). GameLogic also implements the ActionListener that listens to the actual game and provides the necessary steps to be taken after the player has performed an action. After each action it updates the gameboard, score and statspanel (if necessary) with the methods provided.

EndGameLogic
EndGameLogic implements the ActionListener interface and is dedicated to exiting the game and opening the startmenu and the highscores window.

StartLogic
StartLogic implements the Action- and ChangeListener interface for the instance from StartScreen (created in the controller). It coordinates the selected difficulty and skillset and passes this data to the GameController.

WindowHandler:
This class implements the WindowListener interface and adds functionality for closing, activating and opening windows. It is called upon multiple times throughout the entire game by the JFrames and a couple of JDialogs.

Logic package:

Asteroid
This class lends the user the capability to make asteroids and handle them. It contains setters and getters for setting the amount of experience and fuel that can be mined from the asteroid by the player, it also contains methods that allow the player to mine the asteroid.

AsteroidField
The AsteroidField class defines the making and usage of asteroid fields, they can do damage to the shields of the player and the amount of damage can be chosen beforehand.

BlackHole
The BlackHole class delivers objects in the form of black holes. They reduce the fuel of the player when he/she comes to near.

Changeable
The interface Changeable defines a number of methods that have to be implemented by the player class. In this version of the game, only the player class implements changeable, however, if one wishes to expand the game, the EnemyShip class could also implement it.

EnemyShip
The EnemyShip class makes enemy ships that have a certain amount of life, fuel and experience that can be mined by the player once they are defeated. The methods allow it to deal and receive damage as well as be mined after defeat. Higher up in the hierarchy (GameLogic) three instances of this class are made to distinguish between ships of different difficulty.

Fightable
This interface specifies two methods that ought to be implemented by every class that goes into a fight, namely: the Player, EnemyShip, and MotherShip class. The interface specifically demands that objects of a certain class can do damage and receive damage.

Galaxy
Before the game can start, the entire game environment with train tiles has to be build as an abstract non-GUI bound representation. This is done using the Galaxy class. It contains methods to build a galaxy from scratch (every galaxy is filled with galactic objects, see infra) by placing instances of GalacticObjects in a 2D-array; for the moment, every new galaxy is constructed in a random fashion. However, this can easily be adapted if one would want to introduce specific relations between various galactic objects (e.g. moons only occur in the vicinity of planets...). The Galaxy class also allows for updating all the objects it contains later in the game.

GalacticObject
The GalacticObject class contains all the building blocks for building a galaxy with all the terrain it contains (i.e. planets, moons, stars, ships, enemies, black holes, etc.). Every object itself has a few properties, such as visibility, being marked (when a moon is mined, it gets marked so it cannot be mined again). All these are easily adapted using the prescribed methods. It is also here that the requirement of randomness of opponent is met, but in a slightly different way than suggested: our enemy ships can be three different levels with different damage and hitpoints stats associated with each level. The amount of opponents for the three different levels that are generated depend on a random integer.

HighScores
The HighScores class allows for handling everything that has to do with storing and retrieving high scores. It contains methods to write info to a textfile in a specific format up to a certain amount, read and sort the high scores, replace current high scores with new ones, read the entire file and pass on the output to the HighScoreScreen class.

Lowersdef
The Lowersdef interface forces the class that implements it to implement the methods it contains. It specifies that an object from that specific class has the ability to lower the defenses of another object.

Mineable
This interface contains the methods that specify whether and how a certain object can be mined for fuel and/or experience.

Moon
This class lends the user the capability to make moons and handle them. It contains setters and getters for setting the amount of experience and fuel that can be mined from the moon by the player, it also contains methods that allow the player to actually mine the moon.

MotherShip
This class implements the Fightable interface and contains the getters and setters for its stats. It also contains methods for doing damage to the player and taking damage from the player. When the mothership is defeated, the game is won.

Planet
This class implements the Mineable interface and contains method, getters and setters for the amount of fuel or experience it can give to the player.

Player
This class defines all the properties of the player and contains methods for instantiating the player with base stats depending on the chosen difficulty setting and the particular division of skill points chosen in the StartScreen. It also contains methods for increasing and decreasing the stats (offense, defense, fuel) as well as setting and getting the coordinates and for the name of the player and his ship.

ScoreComparator
Implements the Comparator<Integer> interface which is a function to order sets. This class is used in the HighScores class to sort an array of highscore doubles.

Slowable
Slowable implements an interface with the method reducesFuel; that is implemented the BlackHole class.

Star
The Star class implements the Lowersdef interface and contains the constructor for the Star class which can do damage to the player and lowers defense.

View package:

EndGameScreen
This View class makes the JDialog that pops up when the game is won or lost and the player has to enter his name and ship's name for saving in the highscores. Exceptionally, we have not separated controller and view in this case. As one can see, two new private classes were created to implement the CaretListener and the ActionListener. This is done because the CaretListener has to listen to two textfields, which gets quite messy to implement when trying to separate logic and view according to the MVC. For simplicity's sake, the ActionListener was implemented as a *nested* class as well.

Gamescreen
GameScreen is the class that draws the main game JFrame containing a panel for the gameboard, a panel for the stats and a panel for the menu buttons in a BorderLayout layoutmanager. The method makeGameBoard for drawing the game environment defines how the board is made and uses an array of JButtons in a gridlayout, instantiating a first version of the board. The method updateGameBoard draws the gameboard when the controller calls it; it is with this method that the board is initialized and later adapted when necessary. It is backed by an instance of the Galaxy 2D-array (matrix) of terrain objects from the game logic as an abstract representation of the game world. ImageIcons of corresponding terrain types are placed on JButtons with the same coordinates in the grid. The updateStatsPanel method is used for displaying the updated stats at any moment during the game.

GenericButton
This class extends JButton and contains a constructor for all menu buttons used in the project.

GenericPopUp
This class extends JDialog and defines how all pop-ups in the project will behave.

HighScoreScreen
This displays the JDialog containing the highscores. The highscores that the player sees are constructed with the methods from the HighScore class in the logic.

StartScreen
The StartScreen contains all the necessary elements for constructing the start menu and displays the JFrame of the start menu containing panels in GridBagLayout. We used the GridBagLayout manager due to its versatility and flexibility in arranging the panels by using GridBagConstraints.

3. Class diagram

In the appendix you find a class diagram that models the whole game and shows the relationships of inheritance, interfaces and method calls between classes. It is generated from the source code using the ObjectAid Eclipse plug-in which generates UML class diagrams[1]. The diagram included in this

[1] based on the OMG's UML 2.0 specification (see http://www.omg.org/uml/).

document is set to be viewed at A3 paper size, but we suggest to view it as the .svg file which we included in to .zip attachment (.svg files can be rendered in a browser).

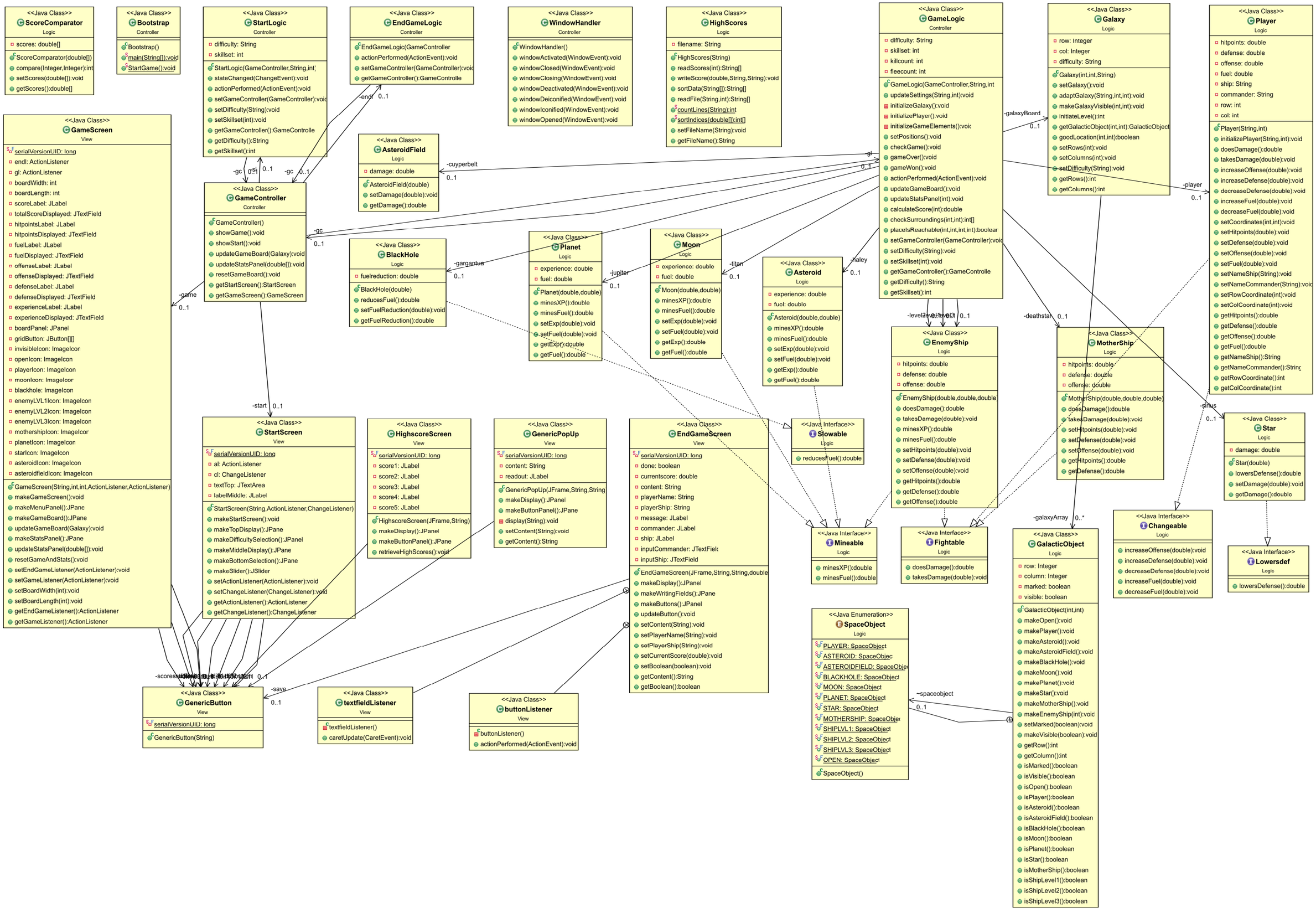
4. Strengths and weaknesses

Weaknesses:

- It would be rather difficult to implement individuality of the enemy ships so every single ship has its own amount of HP which does not reset when the player forfeits an attack on the ship prematurely. Implementing this would require to make separate objects of the EnemyShip class for each ship on the board, now there is one generic object for every ship of a certain difficulty level.
- Adding new listeners and passing on information between the screens is quite complex, all of this has to be done through the GameController and the right order of instantiating everything is very important and maintaining an overview is hard.
- Generally, adding new screens (GUI elements) and their functionality can become tricky, especially if information has to be passed on between various parts of the game, because everything is separated.

Strengths:

- The MVC model was adhered to where plausible. This lends great flexibility, maintainability and expandability to the code (this is also echoed in some of the following points).
- Due to the (quite) strict MVC implementation, it is very easy to adapt the various screens without affecting any other part of the game.
- The strictly separated GameLogic, StartLogic, and EndGameLogic allow for an easy update of the functionality of the game. For instance, in GameLogic, the complexity of the game can be increased a lot without having to adapt any other classes (e.g. stars also do damage to the player in terms of hit points, enemy ships can be mined for fuel/experience after they have been defeated...).
- Because the entire galaxy is created as a whole every time a new game is started, it is very easy to adapt the method with which it is created. For the moment, objects are placed randomly on the board. However, every possible relationship between objects can be implemented and manipulated easily (e.g. moons occur only in the vicinity of planets, other astronomical realities...).
- We added game mechanics for hitpoints and fighting: something which made the game much more interesting and engaging to play than the initially required game.
- Once again, due to MVC it is quite easy to adapt various functionalities without too much trouble. For instance, the HighScore class can be manipulated to allow for more scores being stored, less scores being returned to the player, another format of storing...
- The current code can quite easily be extended to include save game functionality. Save games that save the an ongoing game's state with enemy positions, instantiations and player stats are quite easy to implement, because all the important objects for the game are instantiated in one class (GameLogic) from where they could be exported to some save file and later reused to jump start the game again.


<<Java Class>>
ScoreComparator
Logic
- scores: double[]
ScoreComparator(double[])
compare(Integer,Integer):int
setScores(double[]):void
getScores():double[]
<<Java Class>>
Bootstrap
Controller
Bootstrap()
main(String[]):void
StartGame():void
<<Java Class>>
StartLogic
Controller
- difficulty: String
- skillset: int
StartLogic(GameController,String,int)
stateChanged(ChangeEvent):void
actionPerformed(ActionEvent):void
setGameController(GameController):void
setDifficulty(String):void
setSkillset(int):void
getGameController():GameControlle
getDifficulty():String
getSkillset():int
<<Java Class>>
EndGameLogic
Controller
EndGameLogic(GameController
actionPerformed(ActionEvent):void
setGameController(GameController):void
getGameController():GameControlle
<<Java Class>>
WindowHandler
Controller
WindowHandler()
windowActivated(WindowEvent):void
windowClosed(WindowEvent):void
windowClosing(WindowEvent):void
windowDeactivated(WindowEvent):void
windowDeiconified(WindowEvent):void
windowIconified(WindowEvent):void
windowOpened(WindowEvent):void
<<Java Class>>
HighScores
Logic
- filename: String
HighScores(String)
readScores(int):String[]
writeScore(double,String,String):void
sortData(String[]):String[]
readFile(String,int):String[]
countLines(String):int
sortIndices(double[]):int[]
setFileName(String):void
getFileName():String
<<Java Class>>
GameLogic
Controller
- difficulty: String
- skillset: int
- killcount: int
- fleecount: int
GameLogic(GameController,String,int
updateSettings(String,int):void
initializeGalaxy():void
initiatePlayer():void
initializeGameElements():voic
setPositions():void
checkGame():void
gameOver():void
gameWon():void
actionPerformed(ActionEvent):void
updateGameBoard():void
updateStatsPanel(int):void
calculateScore(int):double
checkSurroundings(int,int):int[]
placeIsReachable(int,int,int,int):boolear
setGameController(GameController):void
setDifficulty(String):void
setSkillset(int):void
getGameController():GameControlle
getDifficulty():String
getSkillset():int
<<Java Class>>
Galaxy
Logic
- row: Integer
- col: Integer
- difficulty: String
Galaxy(int,int,String)
setGalaxy():void
adaptGalaxy(String,int,int):void
makeGalaxyVisible(int,int):void
initiateLevel():int
getGalacticObject(int,int):GalacticObject
goodLocation(int,int):boolean
setRows(int):void
setColumns(int):void
setDifficulty(String):void
getRows():int
getColumns():int
<<Java Class>>
Player
Logic
- hitpoints: double
- defense: double
- offense: double
- fuel: double
- ship: String
- commander: String
- row: int
- col: int
Player(String,int)
initializePlayer(String,int):void
doesDamage():double
takesDamage(double):void
increaseOffense(double):void
increaseDefense(double):void
decreaseDefense(double):void
increaseFuel(double):void
decreaseFuel(double):void
setCoordinates(int,int):void
setHitpoints(double):void
setDefense(double):void
setOffense(double):void
setFuel(double):void
setNameShip(String):void
setNameCommander(String):void
setRowCoordinate(int):void
setColCoordinate(int):void
getHitpoints():double
getDefense():double
getOffense():double
getFuel():double
getNameShip():String
getNameCommander():String
getRowCoordinate():int
getColCoordinate():int
<<Java Class>>
GameScreen
View
- serialVersionUID: long
- endl: ActionListener
- gl: ActionListener
- boardWidth: int
- boardLength: int
- scoreLabel: JLabel
- totalScoreDisplayed: JTextField
- hitpointsLabel: JLabel
- hitpointsDisplayed: JTextField
- fuelLabel: JLabel
- fuelDisplayed: JTextField
- offenseLabel: JLabel
- offenseDisplayed: JTextField
- defenseLabel: JLabel
- defenseDisplayed: JTextField
- experienceLabel: JLabel
- experienceDisplayed: JTextField
- boardPanel: JPanel
- gridButton: JButton[][]
- invisibleIcon: ImageIcon
- openIcon: ImageIcon
- playerIcon: ImageIcon
- moonIcon: ImageIcon
- blackhole: ImageIcon
- enemyLVL1Icon: ImageIcon
- enemyLVL2Icon: ImageIcon
- enemyLVL3Icon: ImageIcon
- mothershipIcon: ImageIcon
- planetIcon: ImageIcon
- starIcon: ImageIcon
- asteroidIcon: ImageIcon
- asteroidfieldIcon: ImageIcon
GameScreen(String,int,int,ActionListener,ActionListener)
makeGameScreen():void
makeMenuPanel():JPane
makeGameBoard():JPane
updateGameBoard(Galaxy):void
makeStatsPanel():JPane
updateStatsPanel(double[]):void
resetGameAndStats():void
setEndGameListener(ActionListener):void
setGameListener(ActionListener):void
setBoardWidth(int):void
setBoardLength(int):void
getEndGameListener():ActionListener
getGameListener():ActionListener
<<Java Class>>
GameController
Controller
GameController()
showGame():void
showStart():void
updateGameBoard(Galaxy):void
updateStatsPanel(double[]):void
resetGameBoard():void
getStartScreen():StartScreen
getGameScreen():GameScreen
<<Java Class>>
AsteroidField
Logic
- damage: double
AsteroidField(double)
setDamage(double):void
getDamage():double
<<Java Class>>
BlackHole
Logic
- fuelreduction: double
BlackHole(double)
reducesFuel():double
setFuelReduction(double):void
getFuelReduction():double
<<Java Class>>
Planet
Logic
- experience: double
- fuel: double
Planet(double,double)
minesXP():double
minesFuel():double
setExp(double):void
setFuel(double):void
getExp():double
getFuel():double
<<Java Class>>
Moon
Logic
- experience: double
- fuel: double
Moon(double,double)
minesXP():double
minesFuel():double
setExp(double):void
setFuel(double):void
getExp():double
getFuel():double
<<Java Class>>
Asteroid
Logic
- experience: double
- fuel: double
Asteroid(double,double)
minesXP():double
minesFuel():double
setExp(double):void
setFuel(double):void
getExp():double
getFuel():double
<<Java Class>>
EnemyShip
Logic
- hitpoints: double
- defense: double
- offense: double
EnemyShip(double,double,double)
doesDamage():double
takesDamage(double):void
minesXP():double
minesFuel():double
setHitpoints(double):void
setDefense(double):void
setOffense(double):void
getHitpoints():double
getDefense():double
getOffense():double
<<Java Class>>
MotherShip
Logic
- hitpoints: double
- defense: double
- offense: double
MotherShip(double,double,double)
doesDamage():double
takesDamage(double):void
setHitpoints(double):void
setDefense(double):void
setOffense(double):void
getHitpoints():double
getDefense():double
<<Java Class>>
Star
Logic
- damage: double
Star(double)
lowersDefense():double
setDamage(double):void
getDamage():double
<<Java Interface>>
Slowable
Logic
reducesFuel():double
<<Java Interface>>
Mineable
Logic
minesXP():double
minesFuel():double
<<Java Interface>>
Fightable
Logic
doesDamage():double
takesDamage(double):void
<<Java Interface>>
Changeable
Logic
increaseOffense(double):void
increaseDefense(double):void
decreaseDefense(double):void
increaseFuel(double):void
decreaseFuel(double):void
<<Java Interface>>
Lowersdef
Logic
lowersDefense():double
<<Java Class>>
StartScreen
View
- serialVersionUID: long
- al: ActionListener
- cl: ChangeListener
- textTop: JTextArea
- labelMiddle: JLabel
StartScreen(String,ActionListener,ChangeListener)
makeStartScreen():void
makeTopDisplay():JPane
makeDifficultySelection():JPanel
makeMiddleDisplay():JPane
makeBottomSelection():JPane
makeSlider():JSlider
setActionListener(ActionListener):void
setChangeListener(ChangeListener):void
getActionListener():ActionListener
getChangeListener():ChangeListener
<<Java Class>>
HighscoreScreen
View
- serialVersionUID: long
- score1: JLabel
- score2: JLabel
- score3: JLabel
- score4: JLabel
- score5: JLabel
HighscoreScreen(JFrame,String)
makeDisplay():JPanel
makeButtonPanel():JPane
retrieveHighScores():void
<<Java Class>>
GenericPopUp
View
- serialVersionUID: long
- content: String
- readout: JLabel
GenericPopUp(JFrame,String,String
makeDisplay():JPanel
makeButtonPanel():JPane
display(String):void
setContent(String):void
getContent():String
<<Java Class>>
EndGameScreen
View
- serialVersionUID: long
- done: boolean
- currentscore: double
- content: String
- playerName: String
- playerShip: String
- message: JLabel
- commander: JLabel
- ship: JLabel
- inputCommander: JTextFiel
- inputShip: JTextField
EndGameScreen(JFrame,String,String,double
makeDisplay():JPanel
makeWritingFields():JPane
makeButtons():JPanel
updateButton():void
setContent(String):void
setPlayerName(String):void
setPlayerShip(String):void
setCurrentScore(double):void
setBoolean(boolean):void
getContent():String
getBoolean():boolean
<<Java Class>>
GenericButton
View
- serialVersionUID: long
GenericButton(String)
<<Java Class>>
textfieldListener
View
textfieldListener()
caretUpdate(CaretEvent):void
<<Java Class>>
buttonListener
View
buttonListener()
actionPerformed(ActionEvent):void
<<Java Enumeration>>
SpaceObject
Logic
PLAYER: SpaceObject
ASTEROID: SpaceObjec
ASTEROIDFIELD: SpaceObje
BLACKHOLE: SpaceObject
MOON: SpaceObject
PLANET: SpaceObject
STAR: SpaceObject
MOTHERSHIP: SpaceObje
SHIPLVL1: SpaceObject
SHIPLVL2: SpaceObject
SHIPLVL3: SpaceObject
OPEN: SpaceObject
SpaceObject()
<<Java Class>>
GalacticObject
Logic
- row: Integer
- column: Integer
- marked: boolean
- visible: boolean
GalacticObject(int,int)
makeOpen():void
makePlayer():void
makeAsteroid():void
makeAsteroidField():void
makeBlackHole():void
makeMoon():void
makePlanet():void
makeStar():void
makeMotherShip():void
makeEnemyShip(int):voic
setMarked(boolean):void
makeVisible(boolean):void
getRow():int
getColumn():int
isMarked():boolean
isVisible():boolean
isOpen():boolean
isPlayer():boolean
isAsteroid():boolean
isAsteroidField():boolean
isBlackHole():boolean
isMoon():boolean
isPlanet():boolean
isStar():boolean
isMotherShip():boolean
isShipLevel1():boolean
isShipLevel2():boolean
isShipLevel3():boolean
-endl
0..1
-gc
-sl
-cuyperbelt
-gargantua
-jupiter
-titan
-haley
-galaxyBoard
-player
-game
-start
-scores
-save
-level1
-level2
-level3
-deathstar
-galaxyArray
-sirius
-spaceobject
0..*