

**CS 319**

**Object-Oriented Software Engineering**

**Fall 2015**

**Void Ablaze**

Section #3 Group #5

Mustafa Yıldız

Aras Herper

Murateren İlgar

Supervisor: Can Alkan

28.10.2015

**Table of Contents**

1. Introduction.....................................................................................................4

2. Requirement Analysis.......................... ...........................................................4

2.1. Overview.......................... .............................................................4

2.2. Functional Requirements................................ ..............................12

2.3. Non-Functional Requirements.......................................................13

2.4. Constraints..................................................................................... 14

2.5. Scenarios.......................... .............................. ...............................14

2.6. Use Case Models............................................................................19

2.7. User Interface................................................................................. 19

3. Analysis........................................................................................................... 23

3.1. Object Model...................................................................................23

3.1.1. Domain Lexicon.............................. .............................23

3.1.2. Class Diagram(s) ...........................................................24

3.2. Dynamic Models............................................................................. 25

3.2.1. State Chart......................................................................25

3.2.2. Sequence Diagram...........................................................26

4. Design

4.1 Design Goals……………………………………………………… 28

4.2 Sub-System Decomposition...............................................................29

4.3 Architectural Patterns…………………………………………… 30

4.4 Hardware/Software Mapping……………………………………… 31

4.5 Addressing Key Concerns..................................................................31

4.5.1 Persistent Data Management........................................... 31

4.5.2 Access Control and Security ........................................... 31

4.5.3 Global Software Control .................................................31

4.5.4 Boundary Conditions.......................................................32

4.1 Design Goals……………………………………………27

5. Object Design

5.1 Pattern Applications.......................................................................33

5.2 Class Interfaces…………………………………………… ..........35

5.3 Specifying Contracts……………………………………..........… 38

5. Conclusion....................................................................................................... 43

1. **Introduction**

Entertainment is trending sector and gaming is one of biggest part of it. As the technology advances, games are improved so much. However, when thinking the philosophy of games and how they entertain people; it might be seen that the key point is the critical design and good implementation of the game.

Void Ablaze is 2D, vertical multiplayer online game which offers simplicity and competition and repetitiveness of fun. Based on player versus player (PvP) elements, each player controls a battleship having the intention of destroying other players’ ship.

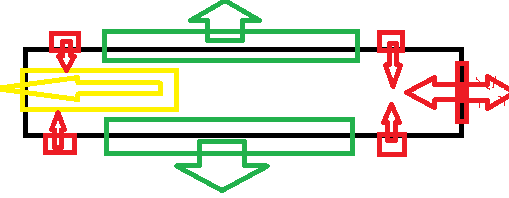
Void Ablaze is an object-oriented game project and the following chapters of this report contain brief investigation, requirements, and models of this game.

1. **Requirement Analysis**
   1. **Overview**

To grasp the background idea and design of the game, following details will help the reader to visualize the game better. In order to do this, details are constituted in five parts that are not necessarily distinct.

1. Ship Structure

Since battleships are the main part of this game, description of them would be a fitting beginning. The game will have only one type of ship (since balancing etc. would be hard otherwise, that’s being said code will allow quick implementation of another design for future uses), this ship has a rectangular shape, with 5 propellers, one located at the back, others located at the two opposing sides of the ship, those side propellers are used for maneuvering and opposing propellers which are closet to each other cannot be active at the same time. The ships armaments contains a massive hyper velocity kinetic energy weapon located at the front, and two lines of broadside kinetic weapons with lower velocity located at the opposing sides; none of these two types of weapon have ability to self-maneuvering or aiming. Figure 1 is a rough visual representation of it. Red rectangles are propellers and red arrows show directions that those propellers propel towards. Green rectangles represents broadside weapon, green arrows represent their firing directions. Yellow rectangle is the hyper velocity weapon, and yellow arrow represents its firing direction. Black Rectangle is the main frame of the battleship.



*Figure 1 Basic visual representation of the ship*

These ships will have a certain amount of mass, ammunition capacity and fuel capacity. Those ships also carries certain number of fighter crafts, and material extractors which will carry material which can be turned into fuel and ammunition to refill empty reserves; conversion is being done in the ships part names as “material converter”. Ship also has a shield against damage to protect it from the harm, by receiving the damage instead of the hull. All those tools also consume energy thus it will have an engine, with a defined capacity, to convert fuel into energy to power weapons, shields, and material converter. For example it is possible to divert entire energy output to shields only.

As a result, the following elements will be part of this ship:

* Hull, mainframe: defining hit-box in the implementation.
* Main propeller (located at the back): greatest source of acceleration can propel towards both back and front (so it has exhaust facing towards both sides).
* Side propeller, starboard(right), front: Used to simply propel the ship towards port(left) when used together with the propeller located at the starboard back; if used alone ship gains angular momentum towards ports side in addition. If used together with the port back propeller, ship only gains angular momentum.
* Side propeller, rest: Same logic only directions differ.
* Broadside kinetic weapons: Have low damage and low energy consumption. Ships velocity affects their velocity, hence their damage.
* Forward hyper velocity weapon: Has high damage and high energy consumption. Ships velocity affects their damage, propels the ship towards back briefly.
* Energy Shield: Receives the damage instead of the hull. It is basically “hit points” of the ship.
* Material converter: Converts raw materials into fuel, fighters/extractors and ammunition. Adjustable.
* Engine: Transforms fuel into energy for energy shied, weapons and material  
  converter, ratio per those tools and total activity can be adjusted.
* Fuel Reserves: Basically a currency, material extractors carry raw material to ship, which will be converted into fuel, powers propellers.
* Ammunition Reserves: Same with fuel reserves, only it powers weapons instead of propellers.
* Fighter crafts: Swarm of small spaceships, released by the ship; can attack to enemy material extractors and main battleship, does not deal any damage to battleship however cripples its energy shield regeneration speed, slowly die while doing so. Have an operating range which is defined by the distance from the ship.
* Material extractors: Gathers raw material from environmental sources of materials, carries them to the main ship, have an operating range similar to fighter crafts.

For a clearer functional idea over currencies:

1. Raw Material:

-Used By: Material converter.  
- Produced By: Material extractor.

1. Fuel:

-Used By: Propellers, Engine.  
- Produced By: Material converter.

1. Ammunition:

-Used By: Weapons.  
 -Produced By: Material converter.

1. Energy:

-Used By: Weapons, shield, material converter.  
 -Produced By: Engine.

1. Shield power:

-Used By: Damage calculation.  
 -Produced By: Shield.

1. Fighter amount:

-Used By: Fighter releasing function.  
 -Produced By: Material converter, returning squadrons.

1. Environmental Structures

There will be colloidal spherical objects that will interfere with user’s battleships and projectiles thrown by them do damage to ships if collision occurs. Some of those structures will also contain raw materials that can be collected by material extractors.

1. Game Space

Game plane will be a Euclidean plane. All objects other than fighters and material extractors will act according to some pseudo-physical rules. For example environmental structures will not move. So any calculation of collusion will contain at least one of the battleships. All collision is non-elastic. Space will contain following object types:

1) Battleships:

Each battleship will be a rectangle in calculations and will contain:

1. Velocity as a 2D vector.
2. Mass.
3. Center of mass.
4. Position of 4 vertices in the Euclidean plane.
5. Acceleration.
6. Angular Velocity.
7. Angular Acceleration.
8. Projectiles:

Each projectile will be a point in calculations and will contain:

1. Velocity as a 2D vector.
2. Mass.
3. Position.
4. Environmental Structures:

Each structure will be a sphere in calculations and will contain:

1. Position.
2. Radius.
3. Fighter Squadrons and Material extractors:

Each one will be a point in calculations, will not be a part of collision calculations and will contain:

1. Position
2. Velocity
3. Target Position
4. Interactions between Objects in the Space

Propellers are deciding the ships acceleration and angular acceleration. Hyper velocity weapon changes ships velocity briefly when used, it is assumed to be instantaneous. If rival ship collides with the shell, ships physical properties (velocity, angular velocity) changes briefly. All projectiles cease to exist after any collision. If a Battleship collides with an environmental object, it loses all velocity; object will not move. Broadside cannons does not interfere with the ships physical properties when used. It also won’t change any physical properties of the rival ship.

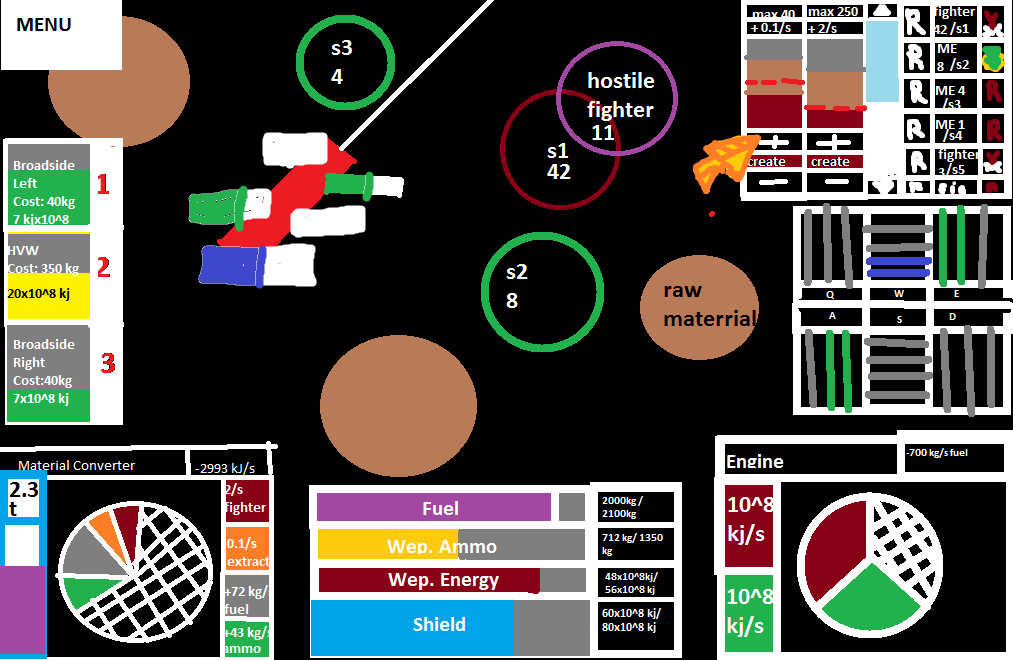
1. Controls and in-game User Interface

This section is included in ‘overview’, for the sake of clarity of the project. However a more detailed part is included at the report.

The game UI consists of 8 elements; those are:

1. Game Pane: Covers entire screen below all other elements, GUI representation of in-game objects is drawn here.
2. Menu Key: Located at top, left corner; allows user to gain access to some features.
3. Weapons Tab: Located at the left edge; contains 3 different elements for 3 different weapon systems contained in the battleship. Those distinct elements contain information about the cost of the firing process and weapons cooling down state. Also shows bound keys to activate those weapons.
4. Material Converter Tab: Located at the bottom left corner, contains GUI representation of; raw material storage, energy cost of the converter, rate of idleness of the converter, rates of all output types (fuel etc.) and actual amount of these outputs. Players can increase and decrease those rates by simply clicking on right and left mouse buttons on the output information panels. Idleness rate adjusted by those changes.
5. Important Information Tab: Located at the bottom edge, contains GUI representation of; shield power, weapon ammunition amount, weapon energy amount and fuel amount also contains numerical representation of those.
6. Engine Tab: Located at the bottom right corner, contains GUI representation of; engines fuel consumption, rate of idleness of the engine, rates of all output types (shield power etc.), their numerical amount. Controls of those are analogous to Material Converter Tab.
7. Propellers Tab: Located at the right edge, contains GUI representation of propeller powers, also contains bound keys to control propeller output amount. Note that, based on Image 2, ‘W’ and ‘S’ have opposing functionalities; ‘S’ increases the power of top edge propeller( main propeller pulls the ship), while reduces the power of bottom edge propeller( main propeller pushes the ship ), opposing is true for ‘W’; so none of those two functionalities can work at the same time. Same is true for top corner propellers and ‘Q’, ‘E’ keys; and for bottom corner propellers with ‘A’,’D’ keys.
8. Squadrons Tab: Located at the top right corner, contains GUI representation of certain details about fighter or material extractor squadrons. Those are;
   1. Maximum number of fighter/material extractor amount that can be maintained.
   2. Increase in this amount per second.
   3. A bar symbolizing amount of crafts that will be included in the created squadron. This bar can be controlled by ‘+’ and ‘-’ keys located at their side. With the button ‘create’ user will create a squadron consisting of that number of crafts.
   4. A list of active squadrons located at the right hand side. Individual elements in this list contains; what they are doing, their ID (generated when they are created), and craft amount they contain. Those elements also have 2 control functions bonded on them; user can click on ‘R’ button with their mouse tor recall that squadron back to ship, hence re adding them to unused crafts pool, and user can click on the panel containing their ID to select a craft, to give another order.

For a rough visual representation refer to Figure 2, note that numbers are completely abstract.



*Figure 2 Basic visual representation of the UI*

* 1. **Functional Requirements**

**-** Ships will move according to their velocity, which is decided by acceleration enforced by propellers whose output depends on the input coming from their respective players.

**-** Ships will fire projectiles according to input coming from their respective players.

- Players will be able to modify output of ships components that converts raw materials to resources like; small craft, ammunition, energy, fuel, shield power.

- Players will be able to administrate numerical strength of each small craft squadron; their orders, and total amount of all separate squadrons.

- Game would be able to couple players online to begin the game.

- Game will end when a players ships shield power reduces to zero.

* 1. **Non-Functional Requirements**
* Main Menu:
  + Play
    - Goes to Connection Establishment Panel
  + Settings
    - Goes to Settings Panel
  + Exit
    - Leaves the game
* Connection Establishment Panel:

Waits until server finds a rival, and responds to user’s machine. Has a cancel capability to return back to main menu.

* Settings Panel:

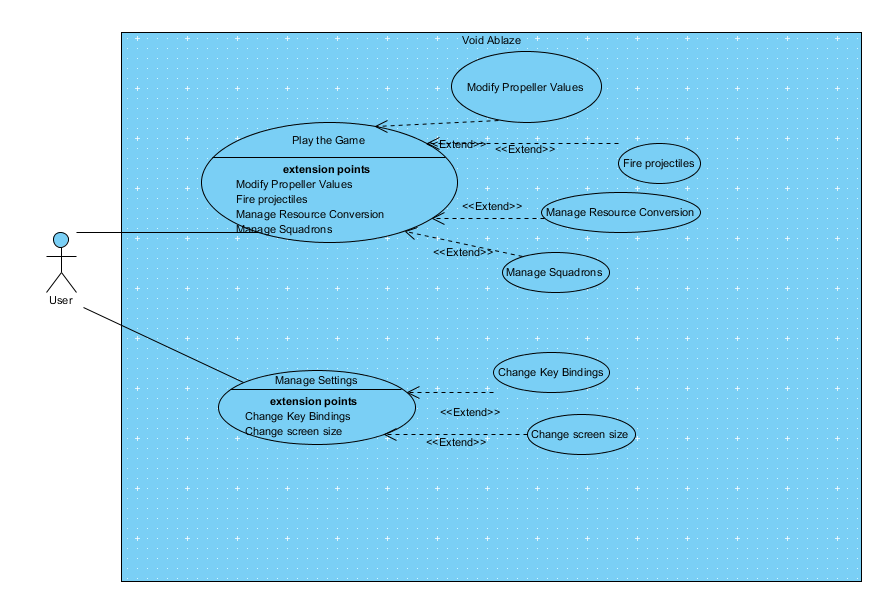
Screen size will be adjustable. After a change re-executing the executable is necessary. Key bindings will be adjustable

* In-game Menu:

Only has exit and cancel capabilities, cancel returns back to in-game screen, game will be running while user spends his/her time in this menu.

* Extendibility: New ships, new projectiles and new environmental objects might be added, so the system will support so. Overall design might include extra amount of players, to add those system better be basic for reimplementation.
* Low delay between a player and the server: Server should be able to resolve a request and respond it back to player fast enough, so that user experience would not be affected.
* When a player loses connection to an ongoing game, or leaves the game; other player will be informed via an GUI message, and returns back to main menu.
  1. **Constraints**
* -Resources of the machine which the server resides in: Since there is no data on GUI interface in the server, RAM usage wouldn’t be a concern compared to CPU usage, dangerous amount of RAM usage will be faced only if there are high number of ongoing games; which will show himself before as latency caused by CPU’s, ideally it would be best to run a thread on each physical core, talking about quad core CPUs that makes four threads, but since we assume primary core will be used by some other stuff, and main thread, maximum number of games, at default, would be 3.
* -Development cost does not exist.
* -System will be implemented in Java.
  1. **Scenarios**

1. For server provider:
   * 1. A runs the server executable.
     2. Executable queries A; “Maximum number of games this server will run:” through the command shell.
     3. Inputs an integer using the keyboard.
     4. Server prints information containing amount of running game instances.
     5. Whenever A closes the shell, game server closes.
2. For user A:
   1. Opening the game
      1. A runs the game executable.
      2. Game GUI comes up. (Rest will continue from here)
   2. Closing the game
      1. A clicks on the ‘Exit’ button
      2. GUI closes, program stops.
   3. Changing the game screen size
      1. A clicks on the ‘Settings’ button.
      2. ‘Settings’ panel covers the game window
      3. A modifies values in the text field next to ‘Screen Dimensions X:’, ‘Screen Dimensions Y:’
      4. A exits from the ‘Settings’ panel by using the ‘Back’ button.
      5. A clicks on the ‘Exit’ button
      6. GUI closes, program stops.
      7. A runs the game executable.
      8. Game GUI comes up with the new dimensions.
   4. Changing the key bindings
      1. A clicks on the ‘Settings’ button.
      2. ‘Settings’ panel covers the game window
      3. A modifies values in the text field s under the ‘Key Binding List’, for each of those binding text fields there will be information about its functionality next to it.
      4. A exits from the ‘Settings’ panel by using the ‘Back’ button.
      5. A clicks on the ‘Exit’ button
      6. GUI closes, program stops.
      7. A runs the game executable.
      8. Game GUI comes up with the new dimensions.
   5. Initiating a game
      1. A click on the ‘Play’ button
      2. ‘Connection Establishment’ panel covers the GUI.
      3. User client program connects to the server.
      4. Server program receives the IP of the client.
      5. Server program waits if total number of games played at the time is under the threshold and prepares a port for the client, otherwise denies the connection and messages the denial to client.
         1. If denial occurs.
            1. “Connection Establishment” panel informs the user, and asks him to try again later.
         2. Else
            1. A waits at the “Connection Establishment” panel, till further information from the server. A can return back to menu by using the button ‘Cancel’. (Both server and the user checks each other in certain intervals.)
            2. Another user gets connected to the server with the similar process
            3. Both A and other users GUI gets covered up by the Game GUI
   6. Playing the game
      1. A pushes the button bound to forward propeller.
      2. A’s ships forward propellers output will increase over time as long as the button remains pushed.
      3. Ships velocity is modified by acceleration induced by forward propeller.
      4. Ship moves by its velocity.
      5. A uses the button bound to left broadside cannons.
      6. Five projectiles appear from the ships left, directions perpendicular to ships left side.
      7. Projectiles travel with their initial velocity, until they hit rivals ship.
      8. Other ship loses some of its shield power.
      9. A uses create a material extractor squadron button located in squadrons tab after adjusting its numerical strength.
      10. Reserved m.e. count will be reduced. A new squadron will be added to squadrons tab.
      11. A clicks on newly created squadron, and clicks on an environmental object.
      12. Material extractor squadrons will start to carry raw material from the object to ship.
      13. A calls squadron back to ship by using recall button located in the squadrons tab next to squadrons ID.
      14. Squadron will return back to ship, and upon return it will be disposed of; reserved material extractor number will increase as much as the numerical strength of the squadron.
      15. A increases material converters fuel output by left clicking on the fuel tab located at the material converter tab.
      16. Fuel created per unit time will increase.
   7. Ending the game.
      1. Through game mechanics
         1. If any of users battleships shield power reduces to 0 at the server, ‘End- Game GUI’ covers the screen, informs the player whether he lost or won, connection between server and clients closes.
         2. User clicks on the ‘Exit’ button in this panel.
         3. Player returns back to main menu.
      2. Through menu
         1. User uses ‘Menu’ button located at the Game GUI.
         2. In-game menu comes up.
         3. User clicks on the button ‘Exit’.
         4. ‘End- Game You Quit GUI’ covers the screen, connection between server and clients closes.
         5. User clicks on the ‘Exit’ button in this panel.
         6. Player returns back to main menu.
      3. Other player quits
         1. Rival Quits using the previous scenario.
         2. ‘End- Game Your Rival Quits GUI’ covers the screen, connection between server and clients closes.
         3. User clicks on the ‘Exit’ button in this panel.
         4. Player returns back to main menu.
      4. Server does not respond
         1. ‘End- Game is not responding GUI’ covers the screen, connection between server and clients closes.
         2. User clicks on the ‘Exit’ button in this panel.
         3. Player returns back to main menu.
   8. **Use Case Models**



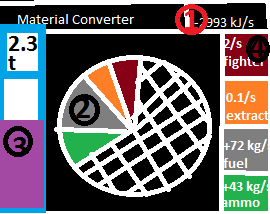
*Figure 3 Use Case Diagram of the Game for an user(Visual Paradigm Professional Edition Evaluation Copy)*

* 1. **User Interface**
* Game Pane: Covers entire screen below all other elements, GUI representation of in-game objects is drawn here.
* Menu Button: Located at top, left corner; allows user to gain access to some features.
* Weapons Tab: Located at the left edge; contains 3 different elements for 3 different weapon systems contained in the battleship. Those distinct elements contain information about the cost of the firing process and weapons cooling down state. Also shows bound keys to activate those weapons. This cooling down state is represented with a rather fat bar located at the background of a weapon information panel.



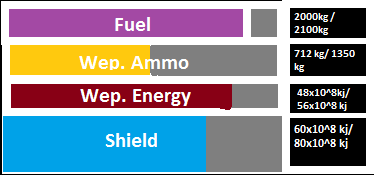
*Figure 4 Weapons Tab*

* Material Converter Tab: Located at the bottom left corner, contains GUI representation of; raw material storage, energy cost of the converter, rate of idleness of the converter, rates of all output types (fuel etc.) and actual amount of these outputs. Players can increase and decrease those rates by simply clicking on right and left mouse buttons on the output information panels. Idleness rate adjusted by those changes.



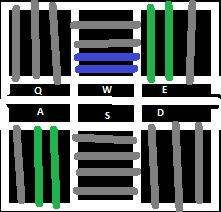
*Figure 5 Material Converter Tab*

* Important Information Tab: Located at the bottom edge, contains GUI representation of; shield power, weapon ammunition amount, weapon energy amount and fuel amount also contains numerical representation of those. Those are represented with bars.



*Figure 6 Important information tabs*

* Engine Tab: Located at the bottom right corner, contains GUI representation of; engines fuel consumption, rate of idleness of the engine, rates of all output types (shield power etc.), their numerical amount. Control of those is analogous to Material Converter Tab.
* Propellers Tab: Located at the right edge, contains GUI representation of propeller powers, also contains bound keys to control propeller output amount. Note that, based on Image 2, ‘W’ and ‘S’ have opposing functionalities; ‘S’ increases the power of top edge propeller( main propeller pulls the ship), while reduces the power of bottom edge propeller( main propeller pushes the ship ), opposing is true for ‘W’; so none of those two functionalities can work at the same time. Same is true for top corner propellers and ‘Q’, ‘E’ keys; and for bottom corner propellers with ‘A’,’D’ keys. With those changes in propeller outputs GUI interface located at the top and bottom changes its state. For example at Image 6 the battleship accelerates backwards with %50 of the output of the main propeller and has angular momentum because of outputs of right-front and left-back propellers, both at %66 output.



*Figure 7 Propellers tab*

Squadrons Tab: Located at the top right corner, contains GUI representation of certain details about fighter or material extractor squadrons. Those are;

* + - * Maximum number of fighter/material extractor amount that can be maintained.
      * Increase in this amount per second.
      * A bar symbolizing amount of crafts that will be included in the squadron to be created. This bar can be controlled by ‘+’ and ‘-’ keys located at their side. With the button ‘create’ user will create a squadron consisting of that number of crafts.
      * A list of active squadrons located at the right hand side. Individual elements in this list contains; what they are doing, their ID (generated when they are created), and craft amount they contain. Those elements also have 2 control functions bonded on them; user can click on ‘R’ button with their mouse tor recall that squadron back to ship, hence re adding them to unused crafts pool, and user can click on the panel containing their ID to select a craft, to give another order.



*Figure 8 Squadrons Tab*

*Explanation of Figure:*

*1: Bar showing how much crafts will remain after the creation of a squadron with currently selected amount.*

*2: Increase in material extractor amount per second.*

*3: User is currently modifying selection amount for material extractors.*

*4: Create button.*

*5: Recall button for 1st squadron.*

*6: Selection button and Information panel for the second squadron.*

*7: Action being performed by second squadron.*

1. **Analysis**
   1. **Object Model**
      1. **Domain Lexicon**

**Battleship:** Main character, player uses this.

**Engine:** The instrument which converts fuel into energy.

**Material converter:** It converts raw material for the fuel, fighters and ammunition.

**Raw material:** Input of material converter.

**Fuel:** Used to generate energy, also consumed by propellers.

**Energy:**  Powers shield and weapons.

**Hyper-velocity weapon:** Forward pointing powerful weapon.

**Broadside kinetic weapons:** Weapons located at both left and right sides of the ship, weaker compared to hyper-velocity weapon.

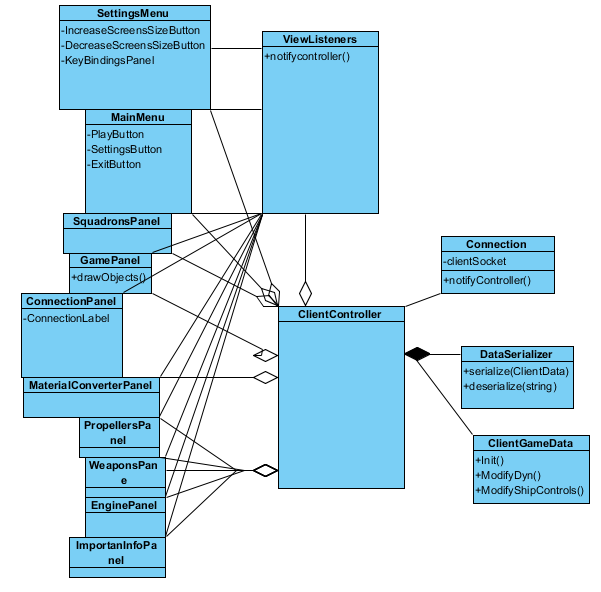
**Ammunition:** Weapons use these.

**Shield Power:** Source of hit points, indicator showing whether you are still living or not.

**Propellers:** Source of acceleration, consumes fuel.

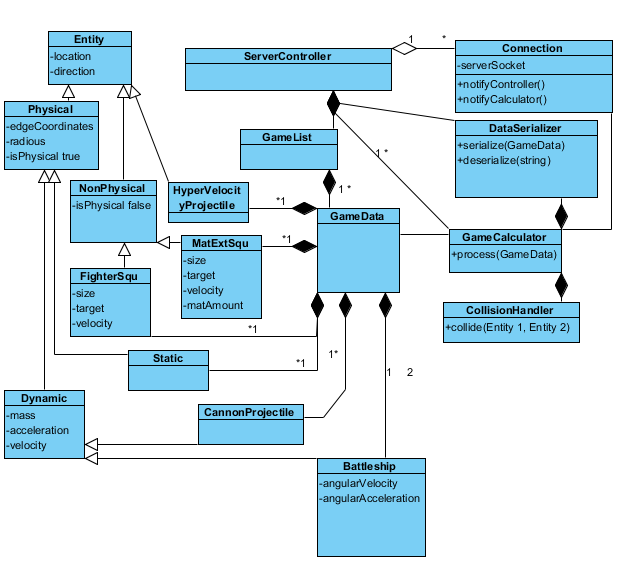
**Environmental Structures:** Static structures contained in the game space.

* + 1. **Class Diagrams**

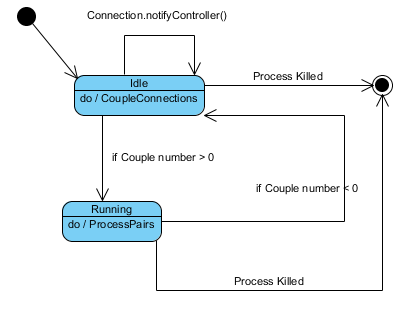


*Figure 9 Client Sided program*

*Figure 10 Server Sided Program*

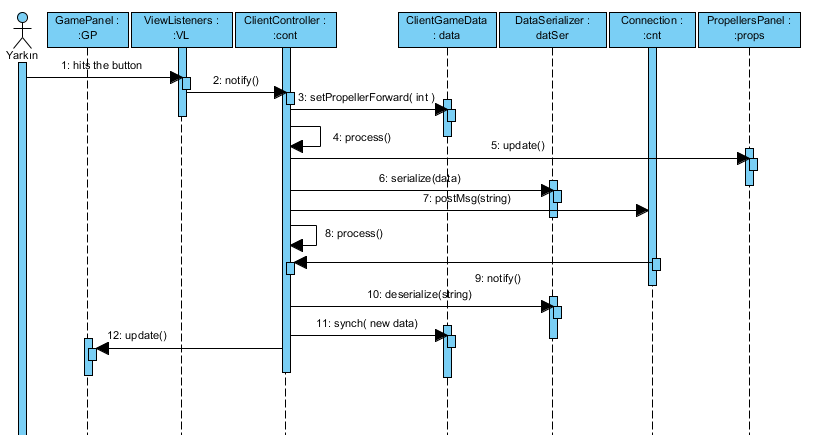


* 1. **Dynamic Model**
     1. **State Chart**



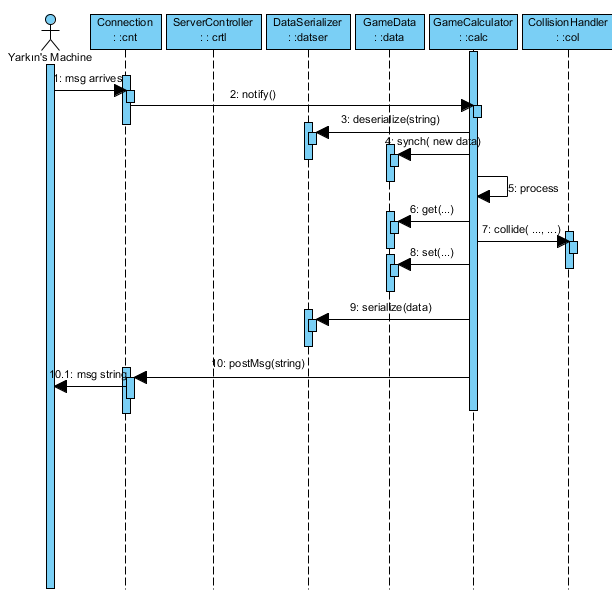
*Figure 11 State Diagram of the Server Controller*

* + 1. **Sequence Diagram**
       1. Scenario 2: Yarkın is playing the game, and currently watches the in-game UI. He uses the key bound to main propeller forward acceleration function.



*Figure 13 Scenario 1*

* + - 1. Scenario 4: The server receives Yarkın’s machines request and processes it, posts its own request back to client.



*Figure 15 Scenario 4*

**4. Design**

### 4.1. Design Goals

* **Time efficiency**

Especially server sided program possibly spending quite a lot of processing power to both, sustain connection to client programs and handle each instance of a single game. Client sided users should not feel any significant battle-neck. The response time of our system is important. The system should react quickly to a given input.

* **Creativity Promotion**

Game mechanics should be challenging enough to reward creativity by giving creative users upper hand over his/her rival.

* **Robustness**

The Game needs to be bug free, and connection should not drop randomly to give maximum amount of satisfaction to users. In case of loses in connection, the game will wait for reconnection and may drop the players when response delayed for a certain amount of time specified.

* **Ease to Use**

One of the fundamental goals of our system is to make it easy to use for the players. A player should easily establish a connection with another player or set up a game server for incoming connections, and should not be confused during using the system.

* **Modifiability**

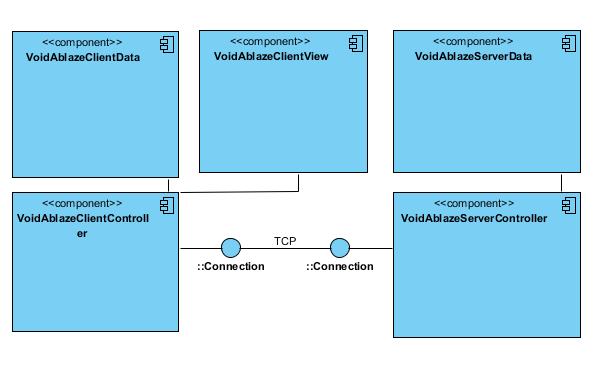
In order to make our system modifiable our system components have been observed to diminish the effects of alterations. We have thought how to make our system easily understandable and manageable to make our system more modifiable.

### Trade-offs

To satisfy those criteria meaningfully more amount of time needed to be sacrificed at the development stage.

### 4.2. Subsystem Decomposition

As clear by now the system will be made from 2 main components separated into two machines. One of these is the server, while other is the client. Server will handle most of the calculation required to run the game, and knows the state of the game; client only processes data related to users view. Both server and client processes stream of data coming from web and listen to it, and both produce and post their own messages back to other.



*Figure 16: Composition Diagram*

**4.2.1 Client Game Data Subsystem:**

Client program will store necessary data only to draw view elements related to that client. This will be stored in an instance of a class, thus termination of the program will also erase any data related to this, since it holds no value anymore; also re-initialization of the game will also wipe it since another games data won’t be related to others.

**4.2.2 Client View Subsystem:**

A group of Java UI classes will be implemented using java graphic libraries; used to draw and receive inputs for both menu components and in-game components.

**4.2.2 Client Program Main Subsystem Component:**

Controls sustenance of TCP connection and modifies data subsystem for protection of synchronization.

**4.2.3 Server Program Main Subsystem Component:**

Controls sustenance of TCP connections, processes games, changes Game Data Subsystem according to post-processed state, changes data according to input strings coming from clients.

**4.2.4 Game Data Subsystem Component:**

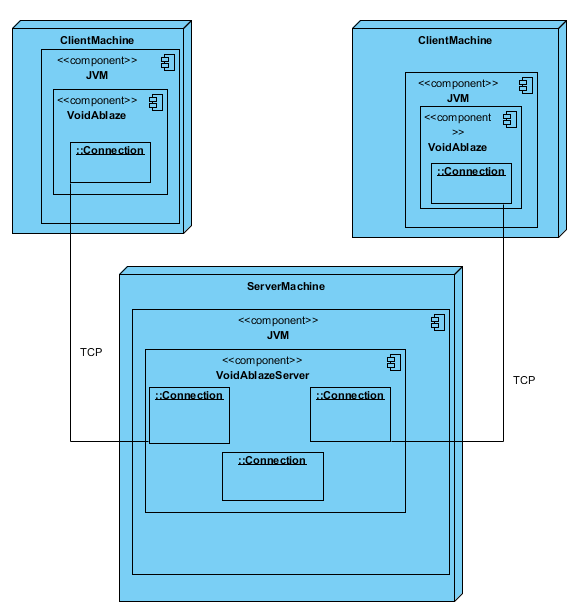
Holds state of each game, by storing them in separate data classes, which stores any components in those games as distinct objects.

### 4.3. Architectural Patterns

System will be monolithic, both client and server will have a central controller for data management and all details related to the main game; however client program will have its certain parts following layered and MVC design pattern to make a better use of java libraries related to basic UI management.

### 4.4. Hardware/Software Mapping

JVM is the hardware interface for client and server machines, so Java is needed to play the game, and launch the server. As implied in previous sections mouse is only used in the menus, and some other components like, fighter management, resource management etc. Rest of the battleships control will be bound to keyboard. Deployment Diagram is given below.



*Figure 17: Deployment Diagram*

### 4.5. Addressing Key Concerns

### 4.5.1. Addressing Persistent Data Management

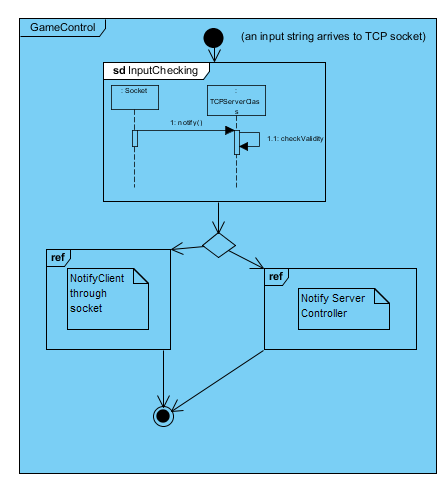
Only IP of the server will be stored in a resource file of the client- side program. Because there is no reward system or any kind of persistent identity; TCP/ IP would be sole interface for the connection; servers public IP will be provided with the client program. Other than settings (key bindings, screen size etc.) for client program will be stored in a file on the machine shared by program and file both.

### 4.5.2. Access Control and Security

Client-Side and Server-Side programs are explicitly separate; no server side functionality coming from the outside of a server is possible. Authority is preserved on the server machine solely because of architectural design limitations, thus no extra subprogram or program is necessary. And since users will not be needed to share any personal information, any security check is unnecessary too. Only danger is the fact that clients and servers share their IPs with each other.

### 4.5.3. Global Software Control

Server-Side program will check and control IPs, legality of inputs coming from client, and the state of all game instances. Input coming from clients will be checked by class implemented to handle TCP connection, main controller will handle timely process of the each individual game, unrelated to validity of the input, and continues to post it via class implemented to handle TCP.



*Figure 18: Server Sided input control interaction overview diagram*

### 4.5.4. Boundary Conditions

* **Program Initialization Boundary Condition**

.jar file will trigger initialization

* **Game Initialization Boundary Condition**

Server Program responds to user’s client sided program while users program is in the waiting for connection state.

* **Game Ending Boundary Condition**

At least a user was leaved, connection interrupted for a long time, or user has clicked on the Quit Game button in the In-Game Menu.

.

* **Program Termination Boundary Condition**

Regular: User clicked on Quit Game in the main menu.

Irregular: Program crashed or user killed the process.

1. **Object Design**

### 5.1 Pattern Applications

**5.1.1 Monolithic Design Pattern**

**Client Side:** Program has only one controller ‘ClientController’ initiating, regulating and binding components at the clients side. This was more or less necessary since it would otherwise be rather hard to regulate input coming from the server, decide on which data on the clients side should be synchronized and when.

**Server Side:** Similar to Client side there is one self existing controller on the top of entire server sided code, ‘ServerController’; however for each individual game ‘ServerController’ oversees initiation of ‘Game Calculator’, which takes on the role of regulating TCP transmission via a ‘Connection’ object. It also modifies ‘GameData’ granted to it by ‘ServerController’; so ‘Server Controller’ via ‘GameCalculator’ creates monolithic base for server sided program.

Monolithic design is occurred here to eliminate the need of constantly checking and synchronizing between sub components, especially data modified by TCP connection; and specific cases can more easily be resolved since only authorized class responsible to change certain data is specific and known.

**5.1.2 MVC Design Pattern**

**Client Side:** Model, view, and control are strictly separated int the client implementation. Model is composed of ‘GameData’ class and certain number of data holding classes, again serving the ‘GameData’. View is composed of Panels, that doesn’t change Models, only reads, and for control which mainly contains the class ‘ClientController’ and certain other classes serving it, initializes and binds those components to each other; it even assigns Listeners to Views.

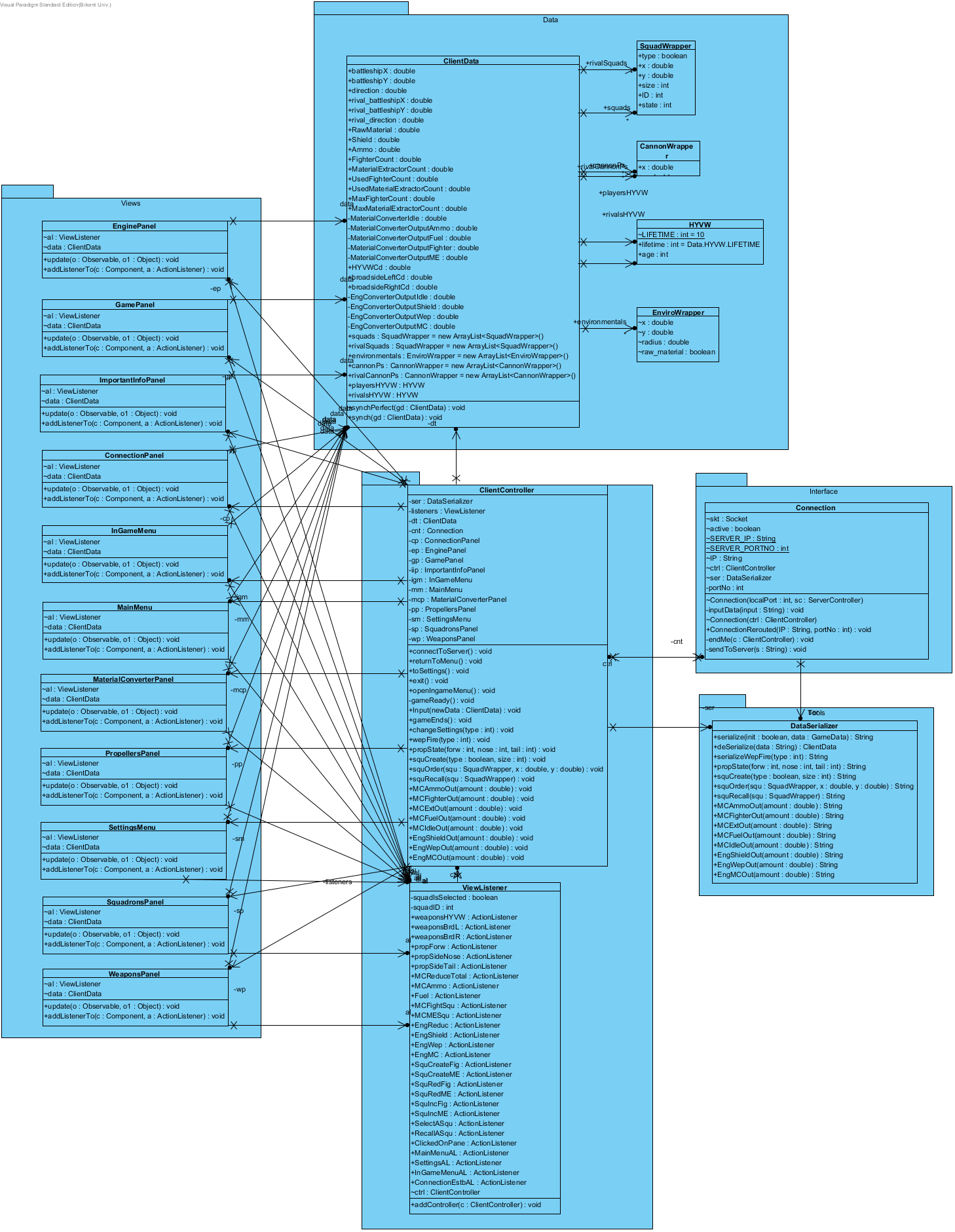
**5.1.3 Object Oriented Design Pattern**

**Client Side:** Other than classes forming the data subsystem of the client system( since most of these have fair share of public variables and very little amount of methods, since they are intended to ease the implementation and readability, they have very little uses to justify being an object) ; classes follow this pattern unavoidably; since it is easier to implemented in Java that way. Communication between ‘ClientController’ and ‘Connection’; communication between views and listeners are primary examples of that. However ‘Serializer’ exists only to hold those methods with an more readable manner, it has very little use as an object, thus its only capability is to have one of its methods called to produce an output briefly.

**Server Side:** Similar to client data classes only exist to ease the implementation and readability, since all interactions between those objects are handled by a ‘GameCalculator’. But other than their package, OOP is easy to spot, since rest of the classes interact with each other using methods. And most objects have their own inner components active despite the procedure of monolithic controller; upon initialization both ‘Connection’ and ‘GameCalculator’ are active and follows their own procedure; interact with each other if necessary.

**5.2 Class Interfaces**

Client side code is described in the figure 19, types of components and visibility of them are described in it.



*Figure 19: Client Detailed Class Diagram*

**View Package:**

Only draws components that makes the view, and contains methods to manage Listeners.

**Data Package:**

Contains ‘ClientData’ class and some other helper classes to hold data.

ClientData: Contains all elements to draw in game elements.

**Controller Package:**

ClientController: Monolithic base of the client system. If it is informed via one of its methods that is related to a players command, it informs ‘Connection’ with a string constructed by ‘DataSerializer’ depending on the command. It does NOT writes changes into data objects directly. Those are changed depending on ‘ClientController’ methods invoked by ‘Connection’; so data models change based on Servers decision; however some parts might be directly written into ‘GameData’ object since that’s why ‘GameData’ has two ‘synch’ methods. It initializes all other sub-components.

ViewListener: Collection of classes extending java.(...)ActionListener class. Controller decides which Listener bound to where by using methods of views.

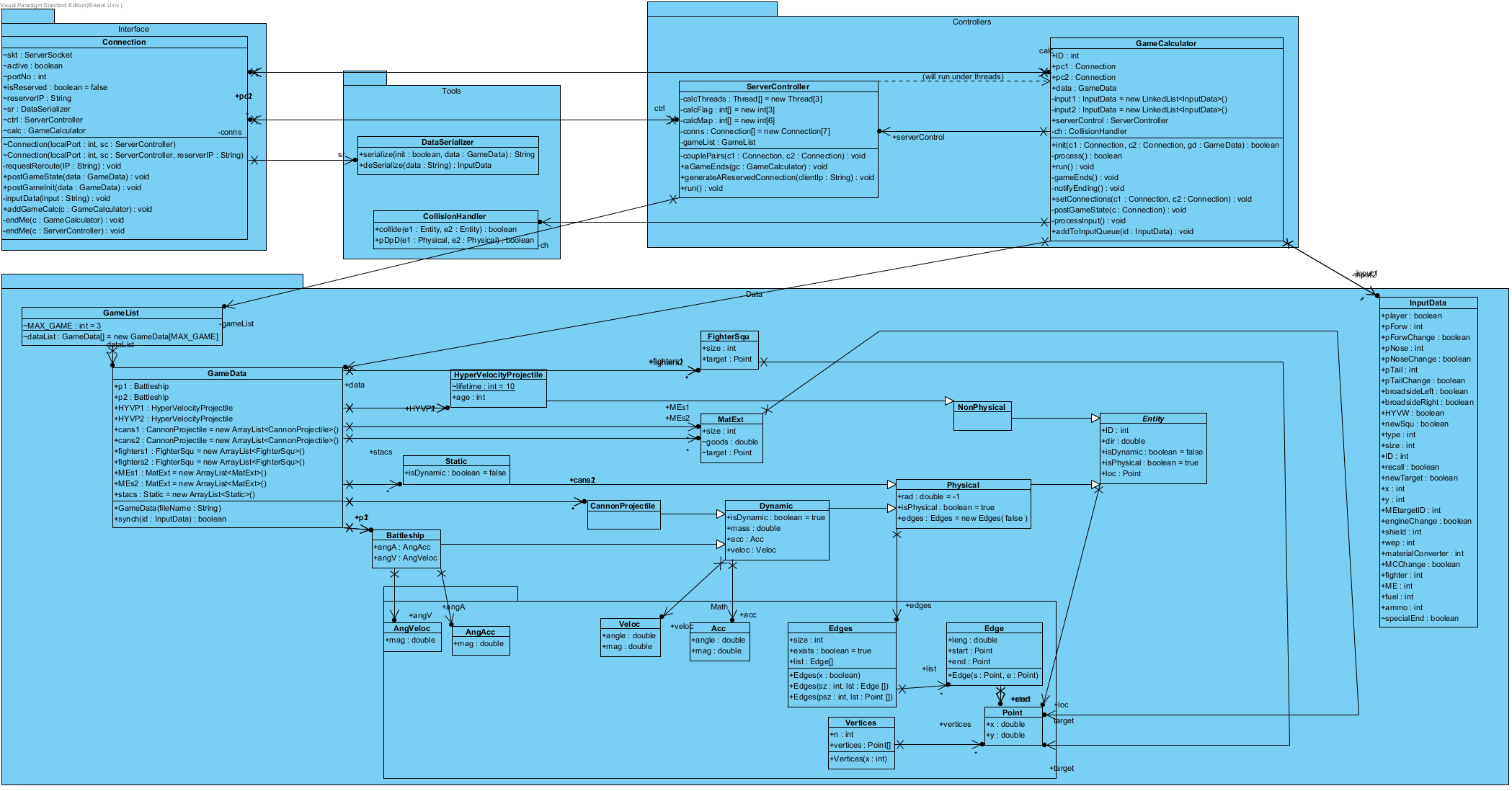
**Interface Package:**

ClientController: A class using java.net’s TCP socket, acts as an interface between server and client.

**Tools Package:**

DataSerializer: Holds few functions, other classes are using.

Client side code is described in the figure 20, types of components and visibility of them are described in it.



*Figure 20: Server’s Detailed UML Class Diagram*

**Data Package:**

***Data.Math Package:*** Holds certain data holding classes which helps higher objects to hold their own data. We will not go into detail since its self explanatory.

GameList: Holds a list of ‘GameData’ objects.

GameData: Holds all in game objects, most of which are extensions of Entity parent class. Constructed by parsing a .txt file for static( environmental ) objects. Can its ‘synch( InputData )’ method be called for synchronization with data coming from clients(via ‘Controller’, via ‘Connection’, via game ‘GameCalculator’).

InputData: Contains all values that can be changed by a client.

**Controllers Package:**

ServerController: Pairs two ‘Connection’ objects wit a ‘GameCalculator’, also instantiates all of them beforehand. Oversees threads that a ‘GameCalculator’ object runs in. Instantiates GameData objects and passes it to instantiated ‘GameCalculator’, oversees purge of used threads and connections.

GameCalculator: Runs in a thread supplied by ‘ServerController’, runs as a game instance from the clients side, after being instantiated by an ‘ServerController’ object, it is free to make use of two ‘Connection’ classes, a ‘GameData’, and a ‘DataSerializer’object supplied to it, runs until a players shield reduces two zero, connection lost or game is forcibly canceled by a player via in-game menu at client side; ‘Connection’ class acts as an interface for those. Holds an input queue, if processing slows down inputs wont be lost then.

**Interface Package:**

Connection: Makes use of java.net’s ‘ServerSocket’, acts as an interface between ‘GameCalculator’ or ‘ServerController’, and Client. One bound to server controller requests from controller to create a new connection instance for that client, if there is a space available sends new data about the Connection to client, so that client can connect to new ‘connection’ object; it also holds reserver clients IP so that it denies any other machine.

**Tools Package:**

DataSerializer: Holds functions, which are being sued by other classes.

CollisionHandler: Holds method to collide two in-game objects, if they can. Only being used by ‘GameCalculator’.

**5.3 Specifying Contracts**

**5.3.1 Server.Interface.Connection:**

**- Connection( int localPort, ServerController sc, String reserverIP )**

Invariant: Object does not exists.

Precondition: Object does not exists.

Postcondition: this.isReserved == true.

**- addGameCalc( GameCalc gc )**

Invariant: this.skt != NULL.

Precondition: this.calc == NULL.

Postcondition: this.calc == gc.

**- endMe( GameCalculator c)**

Invariant: this.skt != NULL

Precondition: this.calc != NULL.

Postcondition: this.calc is informed.

**- endMe( ServerController c)**

Invariant: this.skt != NULL

Precondition: this.ctrl!= NULL.

Postcondition: this.ctrl is informed.

**5.3.2 Server.Controllers.GameCalculator:**

**- run( )**

Invariant: This exists.

Precondition: Thread.currentThread().isAlive() == true.

Postcondition: Thread.currentThread().isAlive() == true.

**- processInput( )**

Invariant: This exists.

Precondition: input1.size > 0 or input2.size>0.

Postcondition: Either input1’s or input2’s size is one lower than precondition state.

**- notifyEnding( )**

Invariant: this.ctrl != NULL

Precondition: gameEnds is invoked()

Postcondition: this.ctrl is informed.

**- gameEnds( )**

Invariant: this.ctrl != NULL

Precondition: data.battleship.p1 == NULL or data.battleship.p2 == NULL.

Postcondition: connections are informed, notifyEnding() is invoked.

**5.3.3 Server.Controllers.GameCalculator:**

**- couplePairs( Connection c1, Connection c2)**

Invariant: This exists.

Precondition: c1.isReserved & c2.isReserved & c1 and c2 are mapped to the same GameCalculator.

Postcondition: GameCalculator::that.connection1 == c1 & GameCalculator::that.connection2 == c2 & c1.calc ==that & c2.calc == that .

**- aGameEnds( GameCalculator gc)**

Invariant: This exists.

Precondition: gc exists.

Postcondition: gc does not exist, and gc.connection1 does not exist, and gc.connection2 does not exist.

**- generateAReservedConnection( String clientIP )**

Invariant: This exists.

Precondition: At least one element in calcFlag<boolean> is false.

Postcondition: That empty element is true.

**- run( )**

Invariant: This exists.

Precondition: Thread.currentThread().isAlive() == true.

Postcondition: Thread.currentThread().isAlive() == true.

**5.3.4 Client.Controllers.ClientConroller:**

**- connectToServer( Connection c1, Connection c2)**

Invariant: This exists.

Precondition: listeners.ConnectionEstabAL.ActionPerformed(Event e) is Invoked.

Postcondition: Connection class was notified.

**- toSettings( GameCalculator gc)**

Invariant: This exists.

Precondition: listeners.SettingsAL.ActionPerformed(Event e) is Invoked.

Postcondition: SettingsMenu.isActive() == true.

**5.3.5 Client.Controllers.ViewListeners:**

**- clickedOnPane.ActionPerformed(Event e)**

Invariant: This exists.

Precondition: InGamePanel.isActive() == true.

Postcondition: ctrl.squOrder(SquadWrapper squ, int x, int y) was invoked.

**6. Conclusion**

So briefly the project is a strategic bullet hell - vertical shooter muliplayer game. In terms of design we have tried to follow MVC and OOP, however since its harder to maintain synchronization, implementation become quite monolithic.

We should have started with implementation earlier, even earlier than analysis report; we lost needless amount of time thinking about it; and designing without implementing proved to be counterproductive.

We used Visual Paradigm for creating UML diagrams .