**Technical solution – Dmitry Veshchikov**

| **Group** | **Model (including data model/structure)** | **Algorithms** |
| --- | --- | --- |
| A | Complex data model in database (eg several interlinked tables)    Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard  Files(s) organised for direct access    Complex scientific/mathematical/robotics/control/business model    Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces  Complex client-server model | Cross-table parameterised SQL  Aggregate SQL functions  User/CASE-generated DDL script  Graph/Tree Traversal  List operations  Linked list maintenance  Stack/Queue Operations  Hashing  Advanced matrix operations  Recursive algorithms  Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty  Mergesort or similarly efficient sort  Dynamic generation of objects based on complex user-defined use of OOP model  Server-side scripting using request and response objects and server-side extensions for a complex client-server model  Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model |
| B | Simple data model in database (eg two or three interlinked tables)  Multi-dimensional arrays  Dictionaries  Records  Text files  File(s) organised for sequential access  Simple scientific/mathematical /robotics/control/business model    Simple OOP model  Simple client-server model | Single table or non-parameterised SQL  Bubble sort  Binary search    Writing and reading from files    Simple user defined algorithms (eg a range of mathematical/statistical calculations)  Generation of objects based on simple OOP model  Server-side scripting using request and response objects and server-side extensions for a simple client-server model  Calling Web service APIs and parsing JSON/XML to service a simple client-server model |
| C | Single-dimensional arrays  Appropriate choice of simple data types  Single table database | Linear search  Simple mathematical calculations (eg average)  Non-SQL table access |

**Group A**

* Queue operations

UIEntryManager.cs

Lines 13, 32, 44, 68

The use of Queue to keep track of active entries, so able to enqueue when made active, and dequeue the entries which have been active for the longest time, and dequeue them. An initial number of entries is instantiated and then are managed via the Manager. Saving memory and garbage collections.

* Complex OOP Model

GameServer - Client.cs

Lines 11 & 24, 12 & 142

Composition of TCP and UDP classes, as well as their definitions within the Client class. Composition of Player class, so able to store player data like position, within a daughter class.

UIEntryManager.cs

Lines 6, 16, 25 inheritance, encapsulation, dynamic generation of objects.

Creation of an abstract class, which 2 other classes (KillFeedUI, InGameChat) inherit from to display UI text entries. Line 16 can be overridden, to make singleton instances of the classes which inherit the abstract class. Line 25 shows a method the only other method which can be accessed by the inherited classes, to make their own text entries. Line 8, EntryTimeToLive is protected so inherited classes can control the time it takes for an entry to disappear. The rest of the implementation, the fields and methods are private, to encapsulate the class.

The add entry method generates objects if none are available or stored, and then manages them via use of a queue.

InGameChat.cs

Lines 14

Polymorphism of the base abstract class.

Lines 38, 42

Use of exposed base abstract class methods.

KillFeedUI.cs

Lines 39

Polymorphism of abstract class to make a static instance.

Line 19

Use of exposed base abstract class method.

NetworkingUI.cs

Lines 12 – 19 aggregation

Holding references to unity gameObjects, so can manipulate and change their activity, and get data from them. Via use of exposed public methods, which can be called by unity.

ScoreboardManager.cs

Lines 14, 23 Aggregation, Dynamic generation of objects

Storing references to ScoreboardEntries which have, via use of dictionary, to manage the self-instantiated scoreboard entries.

GameManager.cs

Lines 36 – 54 Dynamic object generation

Lines 56 – 70 use of dynamic objects, so can subscribe to relevant events.

SoundMusicManager.cs

Lines 12, 33 composition.

EntityWalking.cs

Lines 5 Inheritance from an interface, making of an abstract class.

DamageNumManager.cs

Lines 8, 35 – 38 dynamic object generation, composition

Lines 34 – 53 dynamic object pooling system, that enables for same object to be reused, if available, instead of instantiating a new object.

NonLocalPlayerGun.cs

Lines 6, Inherits a IGun interface.

Lines 9, Lines 62 – 68, 76 – 85 dynamic object generation and management. Pooling system used to add bullets when non are available, and reuse same bullets when available. Reduce memory used, and garbage collection calls when instantiating and destroying many objects.

LocalPlayerGun.cs

Line 4, Inheritence from NonLocalPlayerGun

Lines 8 – 16, Polymorphism, adding functionality of a player being able to shoot, via pressing Fire1 button defined.

Lines 18 – 24, encapsulation of ShootBulletMethod.

MobileLocalPlayerGun.cs

Line 6, inheritence from LocalPlayerGun

Lines 10 – 32 Polymorphism, changing the functionality of how a player shoots, using touch controls.

NonLocalPlayerHealth.cs

Line 5, inheritance from IHealth interface.

NonLocalPlayerAnimations.cs

Line 5, inheritance from IAnimations.

Line 9, aggregation of IWalkingPlayer interface.

Line 54, 58, 72 use of IWalkingPlayer interface

LocalPlayerAnimations.cs

Line 4, inheritance of NonLocalPlayerAnimations.

Line 8 – 19, polymorphism.

NonLocalPlayerMovement.cs

Line 5, inheritance of EntityWalking and IWalkingPlayer.

Line 25 – 34, 50 – 58. Polymorphism

LocalPlayerMovement.cs

Line 5, inheritance of NonLocalPlayerMovement and IWalkingLocalPlayer

Line 55 – 63, 75 – 85, 164 – 193, Polymorphism

MobileLocalPlayerMovement.cs

Line 5, inheritance of LocalPlayerMovement

Line 16 – 71, polymorphism

Line 7, aggregation of Joystick

* Complex OOP Interfaces

FallBoundry.cs

Line 7 – 10

Getting an interface, to get its data, and call appropriate method.

GameManager.cs

Line 69 - 70

IHealth.cs

Implementation of an interface which is given to objects with a health implementation. So can easily grab a reference to an object, not directly the class, but the interface, and call its appropriate methods. Use polymorphism to easily implement different types of handlers for the methods.

IWalking.cs Definition of a Interface

IGun.cs Definition of a gun interface.

Bullet.cs

Line 73 – 87, use of a interface IHealth, to determine if the IHealth should take damage, and call take damage.

IAnimations.cs Definition

IWalkingPlayer, inherits from IWalking. Definition

IWalkingLocalPlayer, inherits from IWalkingPlayer. Definition

* Complex User Defined functions

SoundMusicManager.cs

Lines 42 – 47, 51 – 63, 67 – 77, 81 – 84 Scheduling

Scheduling events to occur in real time, via use of coroutines and IEnumerators that return waitForSecondsRealtime, to assign volume variables, in proportion to the number of frames that have passed, since that time. So able to fade music in and out, to a defined given time.

* Merge Sort/Recursive

ScoreboardManager.cs

Lines 93 – 156

Implementation of a recursive merge sort, to sort the dictionary of scoreboard entries, by the scoreboard score, with lowest indexes having lowest scores. So (soon) able to order the scoreboard by the players’ scores. When TAB held to open the scoreboard.

**Group B**

* Dictionaries

ServerSend.cs

Lines 132, 142, 157, 167

Using dictionary stored in other class, to send data packets via TCP and UDP to specific players.

ServerRead.cs

Lines 16, 43, 57, 89, 90

Using dictionary to call methods in specific clients, according to their ids.

Server.cs

Lines 12, 81, 83,

Storing dictionary of clients, with their iD, to access a clients TCP and UDP fields

KillFeedUI.cs

Lines 16, 17 getting data from PlayerManager’s of specific players from playerDictionary.

Etc…

* Simple OOP model

Player.cs

Lines 34, 39, 44, 48, 54, 59

Using Player class as a holder for data on individual player, which then gets sent to other players via use of ServerSend methods, at lines 69, 72, 78, 81

ClientRead.cs

Lines 9 – 133

Packet Handler methods that are called when receiving specific enum numbered server packets. Call functionality of relevant classes, to achieve wanted outcome.

* Simple user defined mathematical methods

DamageNum.cs

Lines 76 – 87 use of Vector3s to increase size of objects scaled with time and moving of objects scaled with time.

**Group C**

* Linear Search

Server.cs

Lines 55, 79

In line 55, initialising client dictionary

Line 79, finding an available tcp socket in the client dictionary, for a TCPConnect AsyncCallback Result, so can connect an empty TCP socket to a client who is sending tcp connect packets.

GameLogic.cs

Lines 14

Searching for players who are playing, and calling update methods, which send player data to other players.

GameServer – Client.cs

Lines 183

Searching for non user clients, which are not null, and spawning them for the player who just established TCP & UDP connection.

NetworkingUI.cs

Lines 161 – 179

Looking through invalid character indexes in a string given, and then removing the indexes.

etc…

**Group I don’t know which group this falls into**

* Delegates & Events

Server.cs

Lines 16, 60 – 68

Adding a delegate method dictionary, needed to handle specific packets using enum ClientPackets, by sending them to appropriate ServerRead methods.

Networking - Client.cs

Lines 22, 23, 319 – 330

Assigning handler methods to a delegate method dictionary, when receiving packets with a specific ServerPakcets enum.

InGameChat.cs

Lines 27, 28 & 47, 48

Subscribing to OnPlayerConnect/Disconneted events foind in GameManager, to display player name “joined”, “is in game” and “left” entries in chat.

Unsubscribing from events when being destroyed.

PlayerManager.cs (holy grail of delegates and events) (allows for easy integration of functionality to specific events like player dying event, without needing to hold reference to the original event invoker, only need to subscribe PlayerManager’s events.)

Lines 19 – 41 definitions of delegates and events, that other classes subscribe to

Lines 48 – 54, 69 – 72, 89 invoking the events defined, via public methods that other classes able to invoke.

GameManager.cs

Lines 14 – 18, 77 – 84

Definition and invoking events.

SoundPlayerManager.cs

Lines 21 – 26, subscribing to events with relevant methods.

Lines 29 – 42, assigning methods that will be fired when event invoked.

PlayerCameraController.cs

Line24, line 28 – 30, Subscribing to an event.

NonLocalPlayerGun.cs

Lines 53 – 56 subscribing to events with relevant methods.

NonLocalPlayerHealth.cs

Lines 63 – 64, subscribing to events.

NonLocalPlayerMovement.cs

Lines 29 – 32.

* TCP and UDP uses

GameServer - Client.cs

Lines 24 – 140, 142 – 176

Using UDP and TCP network streams, async callbacks, to allow for transfer of data over TCP/UDP, so can transfer data between devices on other networks, local Host, LAN, WAN, Internet.

Networking – Client.cs

Lines 77 – 198,

Using C# TCP/UDP system.net & system.net.sockets to establish an internet connection via TCP and UDP protocols.