Assignment 1
-
Socket Programming and Concurrency

James Craddock-Jones
256515
20/05/2022

Contents

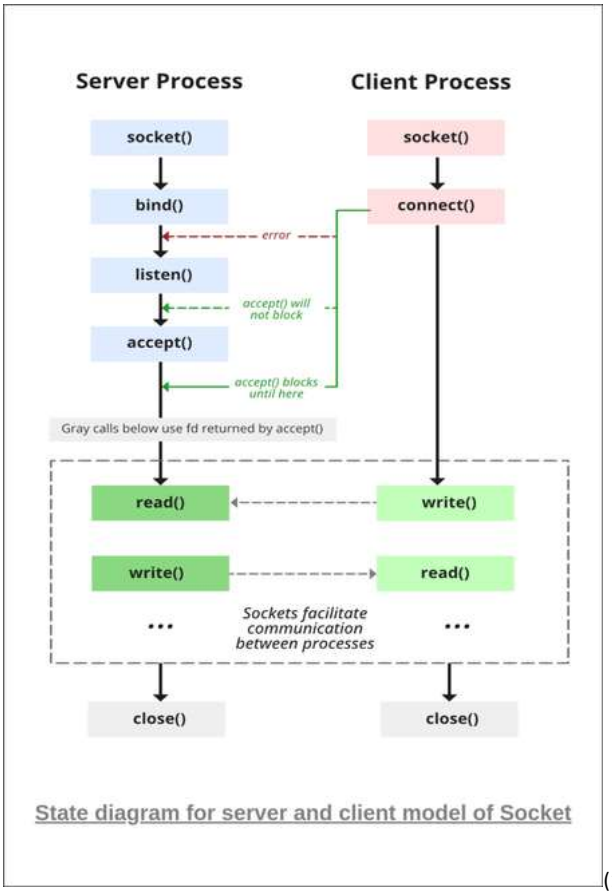
Task 1: Search Report 3
What is Socket Programming? 3
What is TCP/IP? 3
TCP/IP Layers 4
What is the Three-Way Handshake? 5
Three Steps of the Three-Way Handshake 6
The Two Generals' Problem 6
What Are Packets? 7
Why Use Packets? 8
Packet Structure 8
Maximum Transmission Unit 9
MPU and Fragmentation 9
What is the Protocol Data Unit? 10
Encapsulating a PDU 10
Asynchronous and Multithreading 11
Asynchronous Programming 11
Multithread Programming 11
Interoperability in Networking 12
What is Interoperability 12
Why Interoperability is Important 12
Task 2: Application Development 12
Intended Design 12
Introduction 13
Mock-Ups 13
Assets 13
Game Flow 16
Game Network Design 16
Game Maintenance, Extension, and Best Practices 17
Final Design 17
Demo 17
Game Maintenance, Extension, and Best Practices 18
Pitfalls and Drawbacks 20
References 20

Task 1: Search Report

What is Socket Programming?

Socket programming means communicating among multiple nodes connected to the network. While one socket listens on a given port associated with a particular IP address, the other socket connects. Client applications connect to the server, and the server then establishes a listener socket to monitor incoming and outgoing communications. (1)

The diagram below shows how this works with the flow of data transferred between the client and server and how the data is processed.



(1)

What is TCP/IP?

TCP/IP stands for Transmission Control Protocol/Internet Protocol and is called after its two most significant elements. TCP/IP consists of 5 layers. These are the physical layer, the data link layer, the IP layer, TCP/UDP layer, and finally, the application layer. (2)

Sending data over TCP/IP is like sending somebody a message printed on a completed jigsaw piece through the post. The message has been written down, and the jigsaw has been dismantled.

Each jigsaw component can then take a separate postal route, with some taking longer than others. They could be out of sequence when the jigsaw pieces reach their final location after journeying along their numerous paths. TCP/IP ensures that the jigsaw pieces reach their destination. (3)

TCP/IP can be compared to a jigsaw assembler on the endpoint whose job is to arrange the jigsaw pieces correctly, request that missing parts be resent to the endpoint, and inform the transmitter that the jigsaw has been solved. From the very beginning, the first jigsaw piece is transmitted until the endpoint receives the last bit. TCP maintains a link with the transmitter. (3)

Internet Protocol is connectionless. Each chunk of data is identified and transmitted separately from the sender to the targeted system, with no recognition from the destination. (3)

Internet Protocol

Internet Protocol is a network-layer protocol that supports transport protocols such as TCP and UDP and delivers packet data services between programmes. (2)

Transport Control Protocol

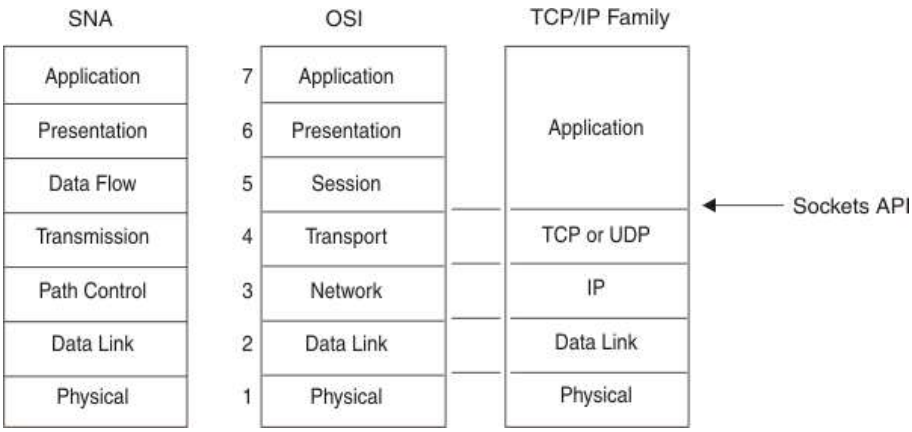
TCP is a protocol for the transport layer. Before data transfer begins, it creates a solid digital relationship among apps. Data using this protocol is given without misjudgements or repeats, while being recovered in the same sequence as it was sent. TCP considers data to be an infinite flow of bytes. (2)

User Datagram Protocol

A further transport-layer protocol that may be used rather than TCP is UDP, which stands for User Datagram Protocol. It links apps over a shaky connection. There is no end-to-end link. As a result, information is transmitted link by link. This protocol comes with no guarantees. Data packets may be received out of order, and data may be lost or replicated when delivered to the final destination. (2)

TCP/IP Layers

Below is a diagram that compares the various transmission protocols to one another, depicting the many layers each contains.



(2)

Application Layer

The application layer allows for the standardised flow of data between many different applications. HTTP, FTP, POP3, SMP, and SNMP are some of the many protocols used daily within the application layer. The actual data that is used within the application layer, which is presented to the end-user, is derived from the payload. (4)

Transport Layer

The transport layer has the responsibility of ensuring network-wide end-to-end connection. TCP provides various features for host-to-host interactions, such as flow control, multiplexing, and reliability. TCP and UDP are transport protocols that are frequently used instead of TCP for specific purposes, such as videogames, where connection speed is more critical than data integrity. (4)

Network Layer

The network layer, commonly called the internet layer, is used to manage packets and connect many different networks to send data over network boundaries. Network layer protocols utilised while sending and receiving data include IP (Internet Protocol) and ICMP (Internet Control Message Protocol), which would then be used for error monitoring all through this process. (4)

Physical Layer

The physical layer may be often referred to as the network interface layer. This layer contains protocols that can only operate on a single connection. The network component is one of the primary components used to link network nodes or hosts. (4)

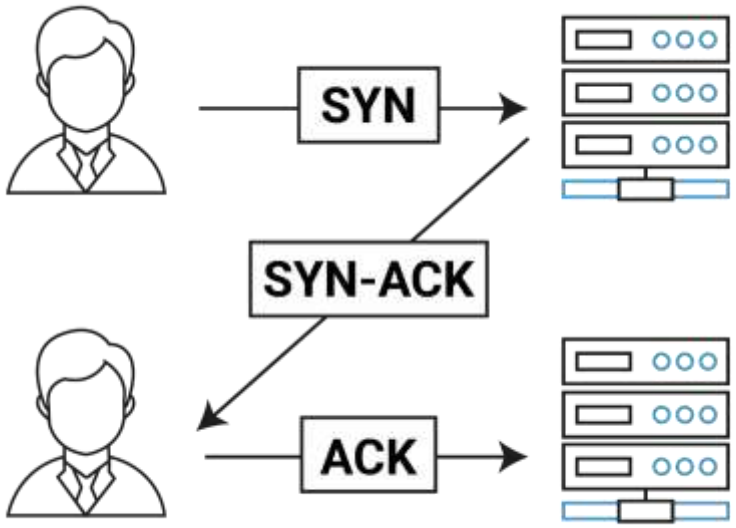
Ethernet is one example protocol that uses this layer to connect to other devices like switches or routers to computers to allow those computers to access the local network and extend into the wider internet. (4)

What is the Three-Way Handshake?

The three-way handshake is a method of communication between a server and a client used by the TCP/IP protocol. This involves three steps, as the name suggests. These include initiating a connection between a client and server and negotiating any parameters for the connection via the TCP socket connection before either party transmits any data to one another. (10)

The three-way handshake allows for data transmission to and from a device simultaneously through the network connection. The three-way handshake must witness both the client/s and server synchronising and acknowledging via SYN and ACK packets before any data transmission communications can be established between both parties Cl. (10)

The three-way connection is a necessary process that is taken out when connecting a client and server together. This is because this process ensures that the relationship between both parties is stable enough that data can be transmitted between each other. Any connection instabilities can be found easily by any data missing between the various SYN and ACK data packets sent between the server and client. (10)



(10)

Three Steps of the Three-Way Handshake

Step 1: the connection between both the server and client is established

The client sends an SYN data packet to the server to establish a connection between each other to transmit data to and from one another. For this to happen, the server must be open to new connections. The SYN data packet will include all the vital information for the server to communicate back to send its response as well as how the client expects data to be sent back to it. (10)

Step 2: The serve revives and processes the request that was sent from the client

Whenever the server node receives SYNC packet data from the client node, it analyses it and sends back a confirmation message in the shape of an ACK packet with two separate number sequences. These number sequences are the ACK one and another SYN, which are sent back to the client. (10)

The ACK in this transaction will add an extra bit onto the end of the SYN that was sent from the client to the server. The SYNC that is being sent to the client is new altogether. (10)

Step 3: The client revives SYN and ACK from the server

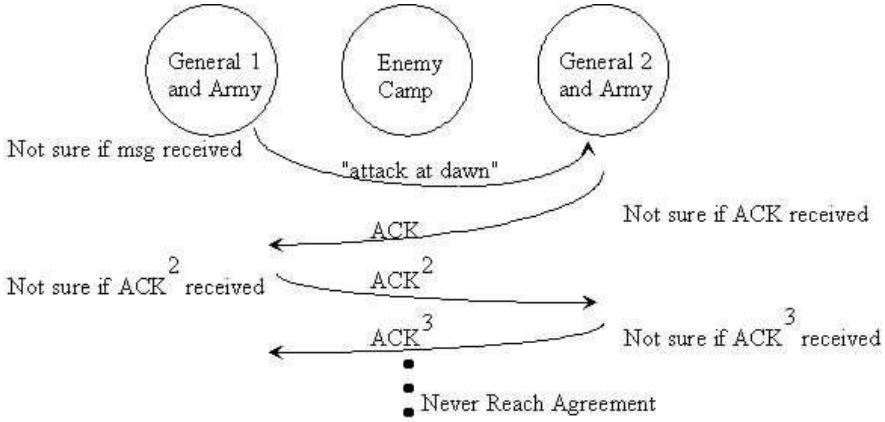
The third and final step to this handshake sees the client receiving the server's new SYN and ACK packets. The client adds a further bit to the SYN that was sent from the server like the server also did from the client. That updated SYN packet is then transmitted back to the server, and once that is done, the connection has been established between both parties. (10)

The Two Generals' Problem

The two generals' problem refers to the communication between two computers over the internet. There is no way to guarantee that communication can be made between both computers or that

transmission will ensure the integrity of the data being sent through the network. Communication containing information such as packets can become lost when travelling through the various routes to the end destination. (11)

The below shows this using two different armies trying to communicate with each other while communicating past the enemy camp. No matter how many communications are sent in the form of an ACK, the sender general will never be sure if the ACK has been sent to the other general. (11)



(11)

For example, if you were to buy a game off a game store like the PlayStation Store or Steam, you will be expecting a confirmation of purchase back from the server when you press buy and expect the game to then show up in your library. If the communication is lost between the client and the server, the client or the server will not know if the game has been purchased successfully, leading to an error. This can cause the money to be still taken from the user's bank account, which can be stressful for them to get back.

There is, however, a workaround that can be somewhat made when handling the lost connection. The most common workaround is to always expect data to be lost when transferring data over the internet and use, for example, connection protocols such as TCP to mitigate loss. (11)

On the other hand, there is an almost perfect solution to this issue. Get computer A to assume how long it takes to transmit the data through the connection route and spam messages for however long it takes until computer B sends a confirmation message back. (11)

After sending the message, computer B then waits for further communication from computer A with the absence of messages being sent back from computer A allowing computer B to build up knowledge on what has been agreed by computer A. (11)

What Are Packets?

A network packet is a small broken-down component of a more extensive transmission sent over a network. Data is divided into small packets and transmitted over computer networks like a local area network or the internet. When the packets are received at the endpoint, the endpoint rebuilds the packets in their correct order back into the extensive transmission. (5)

An excellent example of this is provided by Cloudflare, which states, "*Suppose Alice is writing a letter to Bob, but Bob's mail slot is only wide enough to accept envelopes the size of a small index card. Instead of writing her letter on normal paper and then trying to stuff it through the mail slot, Alice divides her letter into much shorter sections, each a few words long, and writes these sections out on index cards. She delivers the group of cards to Bob, who puts them in order to read the whole message.*" (5)

Why Use Packets?

Packets are not necessarily needed when sending data from one computer to another. When only two computers communicate with one another, they can transmit that data in one long line of bits to each other without issues. (5)

The issue with this comes in when communicating with many computers at one time instead of just two. Doing it this way with more than two computers starts encountering bandwidth issues. The connection speed is limited by hardware such as Ethernet cables, routers, switches, and external factors such as your ISPs provided speeds. Sending long bits of data, in this case, saturates those connections. (5)

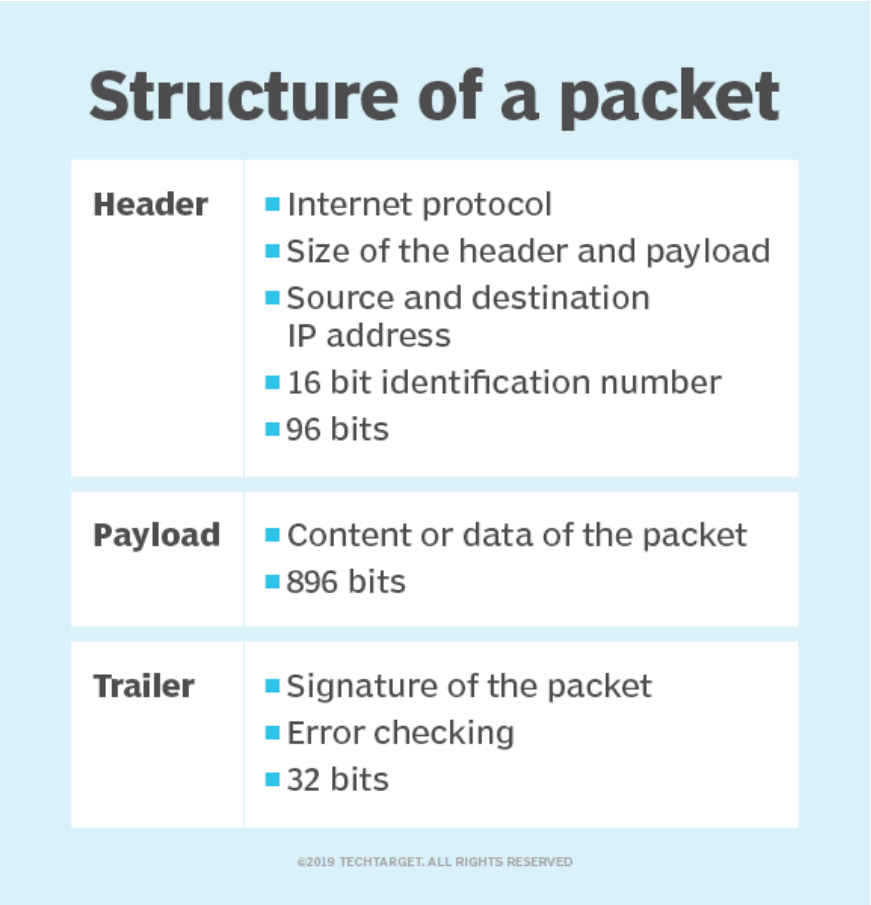
Using packets reduces this burden as data can be broken down into more minor elements of data that can be transmitted down the most effective routes through the internet or network to the end destination. There is no order in which these must be sent, allowing multiple connections over the same networking equipment. (5)

Packet Structure

The header is the most significant portion of a packet. This is due to the number of elements it includes to ensure that all data is transferred correctly. (6)

These elements include the checksum, which is the part of the packet that detects any errors with the data transmitted, a 16bit identification number, any flags that tell hardware such as routers if the packets are intended to be fragmented, and the fragmentation offsets. In addition, the destination address, number of hops the packet has made or can make, the IP address, length of the packet, and many more such as the packet number that is used when reassembling the data. (6)

The payload includes the actual data being sent with the trailer, including further error checking and mitigation. (6)



(6)

Maximum Transmission Unit

A maximum transmission unit is a unit of quantification used when transmitting data across the internet. It represents the biggest data packet that a networked device can handle. "*Imagine it as being like a height limit for freeway underpasses or tunnels: Cars and trucks that exceed the height limit cannot fit through, just as packets that exceed the MTU of a network cannot pass through that network.*" (7)

Any packet that exceeds the limit of the MPU can, on the other hand, be further broken down into smaller packets to fit within the MPU limit set by the network device. MPUs use bytes as a unit of measurement. One thousand five hundred bytes is the max MPU limit that is allowed. (7)

MPU and Fragmentation

When two computer devices establish a link and communicate data, the data is routed over numerous networks. Not only the MPU limit of all devices in-between the two computers must be

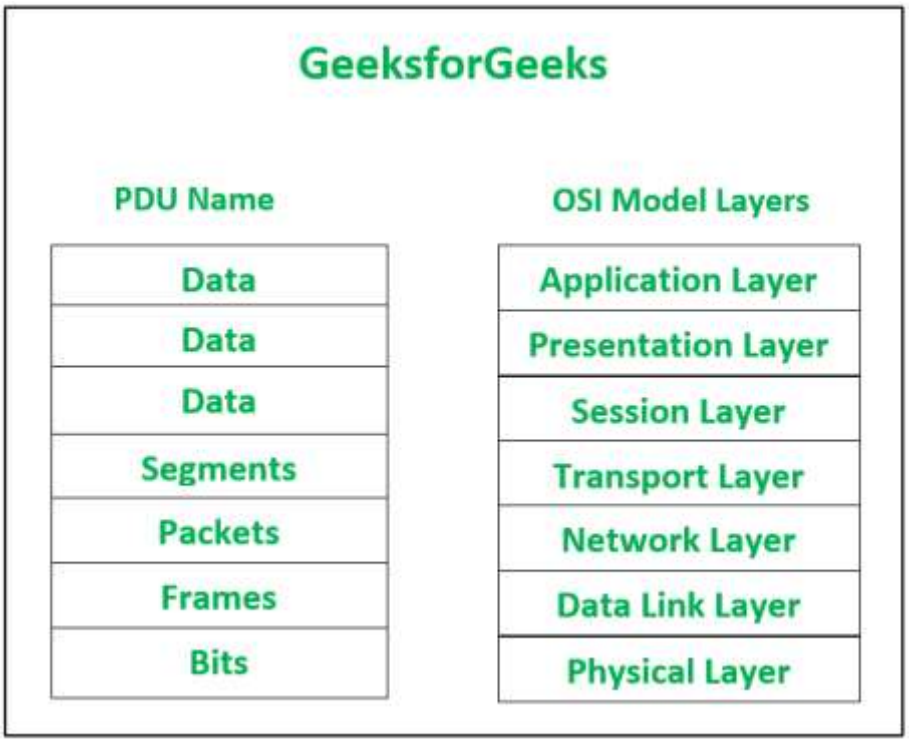
considered. Fragmented packets are those that surpass the MPU anywhere along the network route that the packets will travel across. It is a mix of control and user data described in a specific OSI model layer. (7)

What is the Protocol Data Unit?

Protocol Data Units are the units of information sent across network connections containing control information, address information, or data transmitted from one computer to another. (8)

In the OSI model, the PDU takes many forms:
layer 1: the PSU is just a bit of data ---> **Layer 2:** the PDU is now a frame of data ---> **Layer 3:** the PDU is now a packet that will be transmitted over the Internet ---> **Layer 4:** the PDU is now a segment of data if using TCP and a datagram if using UDP. (8) (9)

After Layer 4, the DPU is the data used within those layers, such as the application and presentation layers. (9)



(9)

Encapsulating a PDU

Encapsulating a PDU involves wrapping one PDU within another PDU. This happens when one PDU is sent lower than another PDU. For example, if that PDU was sent in the data field. A way of looking at it is if a PDU is acquired from a higher protocol stack and includes information containing pertinent lower layer header information. This data has been readied for transportation to the network layer. The PDU indicates its status as data moves from one plane to another. (9)

Asynchronous and Multithreading

Asynchronous Programming

Asynchronous programming is the process of parallelisation in which a sequence of instructions runs separately from the primary programming process. (12)

Suppose a stalling action were to occur in the programme, but we wish to proceed with the processing without waiting for a response. In that case, we utilise asynchronous programming to work around this issue. It enables programmers to design processes that can execute concurrently without waiting for other processes to be completed. (12)

How to use Asynchronies Programming

Within C#, asynchronous programming is done by using async and await functions. For example, you may have set up a private void that you wish to become asynchronous. This can be done by using the await function within the void and using async when declaring the private void, as shown below.

```
0 references
private async void example()
{
    i++;
    await Task.Delay(1000);
    Console.WriteLine($"It has been {i} seconds.");
}
```

When to use Asynchronies Programming

Asynchronous programming should be utilised while executing processes that could stall or delay the process tree across the many methods being conducted. "*When performing blocking operations between the methods, we should use asynchronous programming. In scenarios where we have a fixed thread pool or when we need vertical scalability in the application, we use asynchronous programming.*" (12)

Multithread Programming

A thread is a singular steady stream of management inside a programme. Multithreading is just a mechanism in which the microprocessor within the computer employs many threads to run numerous tasks simultaneously. (12)

Multithreading is mainly employed when programmers wish to take advantage of multi-threaded computers by having many workers operate independently from the main thread, usually the GUI thread. (12)

How to use Multithreading

Within C#, multithreading is slightly more complex compared to asynchronous programming. You must create a new thread that is linked to a task and then start that thread for the job to be processed.

```
int i = 0;

0 references
public void StartNewThread()
{
    Thread t1 = new Thread(new ThreadStart(example));
    t1.Start();
}

1 reference
public void example()
{
    id++;
    Thread.Sleep(1000);
    Console.WriteLine($"It has been {i} seconds");
}
```

To stop a running thread, you would then do t1.Abort();. Note that it is important to make sure a thread is thread-safe before merging a thread outcome with the main thread. For example, updating the GUI from a thread. Not making an independent thread thread-safe, results in a program crash.

When to use Multithreading

Multithreading should be used when many independent tasks are being excited at the same time. "*When we have independent functions performing independent tasks, we should use Multithreading. One good example of this will be downloading multiple files from multiple tabs in a browser. Each download will run in its own thread*." (12)

Interoperability in Networking

What is Interoperability

Network interoperability permits connected networks to continue sending and receiving data while retaining the acceptable level of quality required by the endpoint with no adverse effect on the broadcasting and acquiring platforms. (13)

Network interoperability mainly alludes to the effective interconnectivity of telecommunication services across multi-vendor, multi-carrier interconnections under intensive or possibly even routine settings while adhering to the necessary standards and criteria. (13)

Why Interoperability is Important

In order to accomplish end-to-end communication, network interoperability is a critical process that needs to be undertaken. If the networks are more varied, the more critical it will be to guarantee that they can interconnect to enable end-to-end transmission. (13)

Many benefits are brought from using network interoperability, as stated below:

"*The user benefits as he can communicate with whom he wants to with a single terminal.*

The telecom operator benefits as it can select the best equipment available from different manufacturers based on best performance and prices.

The product/service manufacturer benefits as it can extend the same equipment set to different operators." (13)

Task 2: Application Development

Intended Design

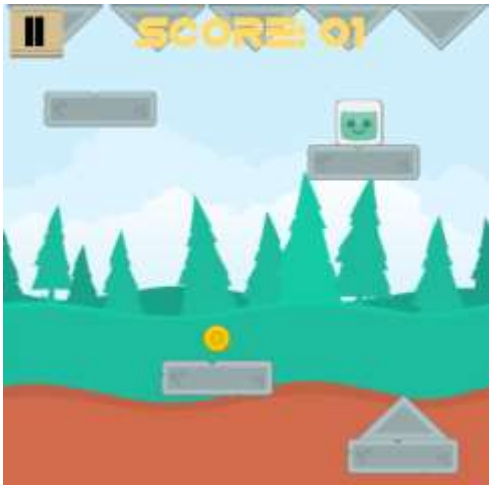
Introduction

Gravity is a hyper-casual game in which the player must travel a randomly generated terrain as far down as possible without dying by falling into the abyss, onto a spiked platform, or be transported to the top by the platforms only to be slain by even more lethal spikes.

You have been transported to a dimension that consists of a continuous flow of platforms moving up into space. To survive in this dimension, you will have to overcome complex challenges such as dodging spiked platforms, which could result in your death by falling into the void beneath the earth.

Mock-Ups

The mock-up depicts the game itself, in which the player must either fall to the next platform and maybe die or stay on the current platform and perish when it hits the spikes at the top of the screen.

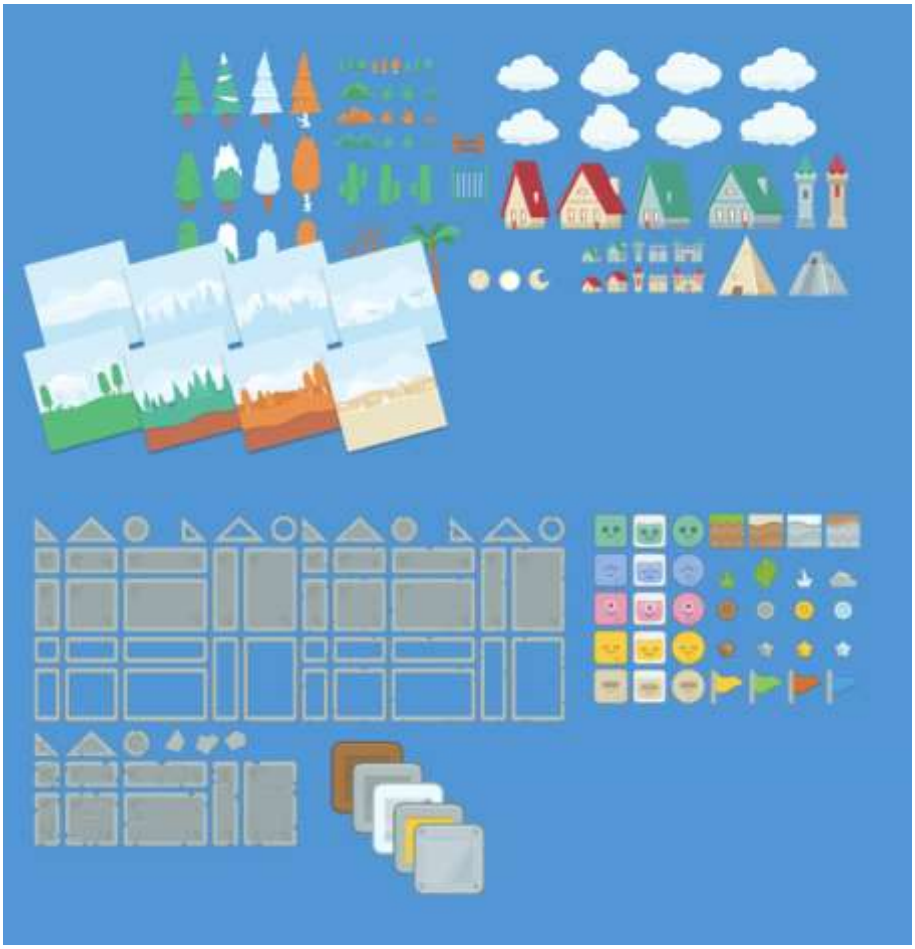


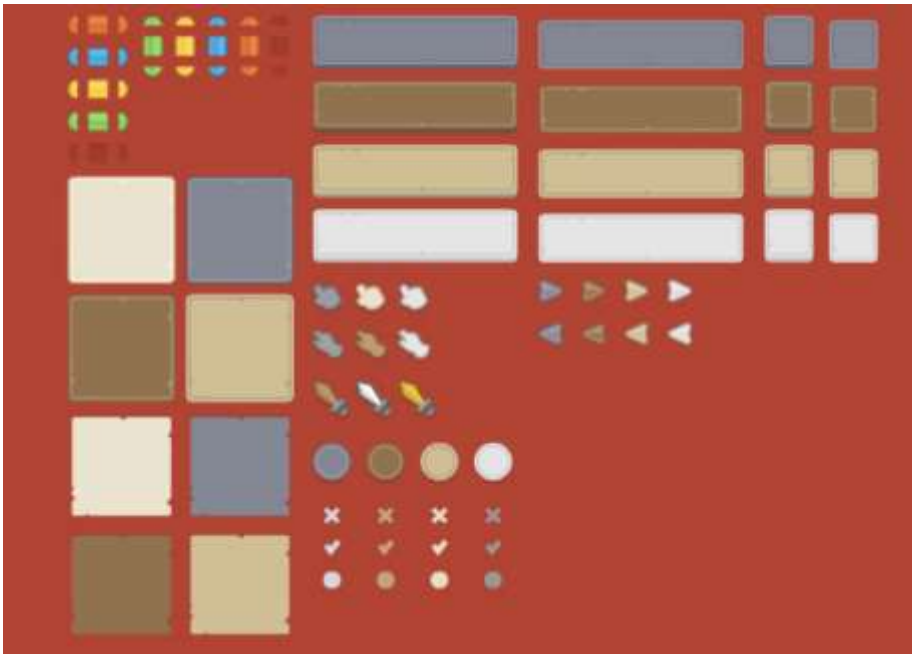
Assets

I have selected to use the assets from kenney.nl, a website that offers free copyright and license-free materials for game development.

The physics components will be used for the player sprites and some of the platforms that the players will be able to drop down into.

The RPG UI bundle will be used for the game's interface because it matches the game's design and feel.

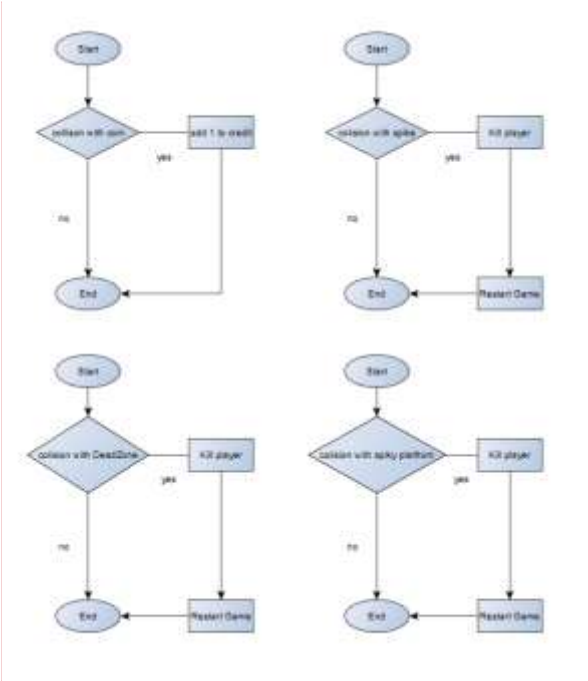




Links to assets:

https://www.kenney.nl/assets/background-elements-redux
https://www.kenney.nl/assets/physics-assets
https://www.kenney.nl/assets/ui-pack-rpg-expansion

Game Flow



Flowchart 1 is used to determine whether or not the player has collected a coin. If the player has, they will receive 1 point; if they do not, the game will not add any points to the score.

According to the second flowchart, the player has reached the spikes at the top of the map. If the player reaches them, the game will pause, and a text menu will show before restarting.

The third flowchart checks to determine if the player has entered a dead zone, where platforms spawn. If they go below this level, the game item will be destroyed, and the game will end when a menu appears.

According to the fourth and final flowchart, the player has plummeted onto a platform with a spike on top of it. If the player lands on a menu, a message will show for a brief period before the game restarts.

Game Network Design

The game will be using both TCP and UDP protocols to transfer data to and from the game and game server. Essential data such as user IDs will be sent over TCP to ensure data relating to such types reach the server and client 100% complete. On the other hand, movement data will be sent via UDP for the data to be sent, received, and displayed in real-time. It is also the least important data being transmitted.

The server will be developed within a console application rather than being developed in Unity like the game's client. Player movement will be handled on the client-side and sent to the server rather than the server handling movement entirely. This will be done as the game was intended to be a

single player. It also ensures player gravity works as intended when falling without being simulated on the server.

Game Maintenance, Extension, and Best Practices

Commenting on the code is a fundamental process that will be done while the game is being developed. Depending on the coding language, it can be as easy as typing / for a single line remark or /**/ for multi-line comments.

Another way they will abide by best practices is that the files and folders will be organised logically. This allows folders to be divided into parts, such as the assets that would be kept within them, such as sprites and scripts in this case.

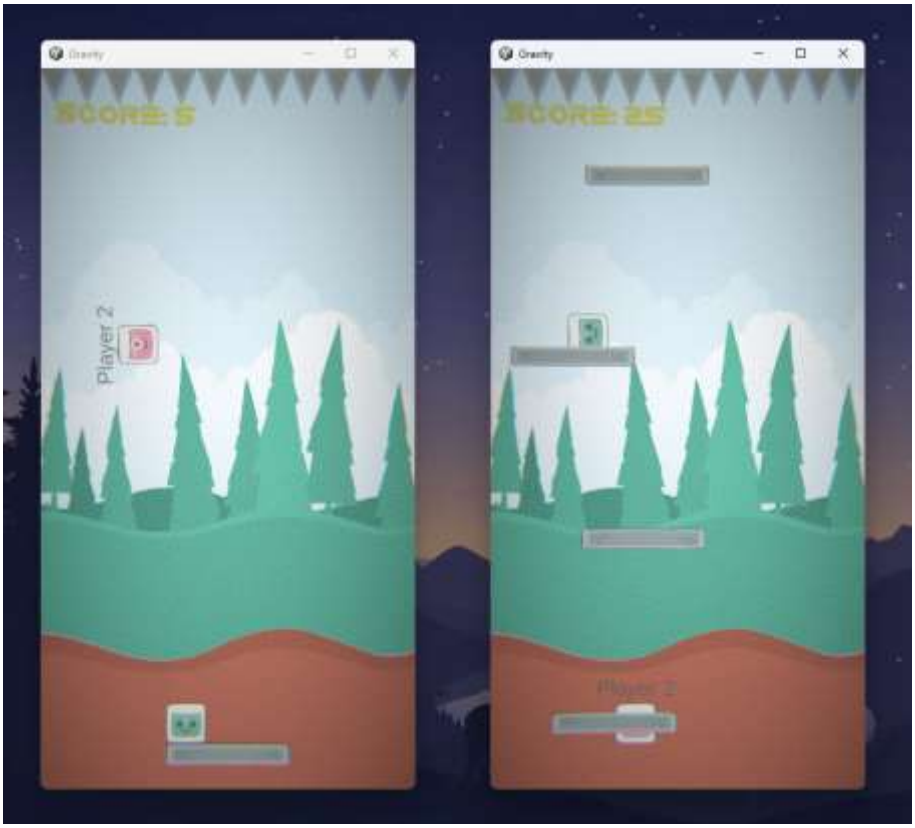
Furthermore, appropriately identifying variables will also be used to aid maintenance and withstand best practices. For example, if one was titled moveSpeed, it shows clearly that it is a variable that regulates the movement speed of the object to which the script is tied.

Final Design

Demo

Link to demo video: https://youtu.be/lin08tsM-j0

Link to code: Development



This is the final product. The right image is the game as it should be, while the left is used as a demonisation of working networking as it would be impossible to have two instances of the proper game running as the player would die straight away. The platform spawning and movement are done locally, so the players are battling to last longer than each other.

Game Maintenance, Extension, and Best Practices

The game was mostly commented throughout both the client and the server projects that have been developed. Below is an example of this with the connection and send data methods where the player data is managed when making a connection and where data will be sent to the player/s.

```
public void Connect(TcpClient _socket)
{
    //Get socket
    socket = _socket;
    //Set buffer size
    socket.ReceiveBufferSize = dataBufferSize;
    socket.SendBufferSize = dataBufferSize;

    //Get socket stream
    stream = socket.GetStream();

    dataRecieved = new Packet();

    //Collect buffer data
    receiveBuffer = new byte[dataBufferSize];

    stream.BeginRead(receiveBuffer, 0, dataBufferSize, ReceiveCallback, null);

    ServerSend.Welcome(id, "Welcome to the server!");
}

3 references
public void SendData(Packet _packet)
{
    try
    {
        //Check is socket is empty
        if (socket != null)
        {
            //Begin stream write to send to player/s
            stream.BeginWrite(_packet.ToArray(), 0, _packet.Length(), null, null);
        }
    }
```

In addition, all the assets used throughout the development have been organised into folders and subfolders that are relevant and easy to understand for future development and maintenance. The images show this with the main assets folder with all the major components split into their own folder categories and subcategories.

In the following image, the scripts have been split up into their own subcategories, socket pro for the socket programming scripts, player for the player scripts, platform for the platform scripts, and MainMenu for the main menu scripts.

Pitfalls and Drawbacks

There are many drawbacks and pitfalls with the current implementation. One is that the platform spawning system is done at a local level, meaning that the game will differ for each player and vary in difficulty.

In addition, the current implementation does not handle disconnections properly and requires the whole game executable to be restarted every time the player dies for the disconnection to occur. This causes an error on the client when trying to re-join without closing and reopening the whole game.

Furthermore, the current implementation continuously spams the server and clients with player locations when not needed. A work around could be used on the local player's side to only send data when movement is detected.

References

1. GeeksforGeeks, *Socket Programming in C/C++*, Socket Programming in C/C++ - GeeksforGeeks Data accessed: 14/04/2022
2. IMB, *TCP/IP TCP, UDP, and IP protocols*, TCP/IP TCP, UDP, and IP protocols - IBM Documentation Data accessed: 14/04/2022
3. Cloudflare, *What is TCP/IP?*, What is TCP/IP? | Cloudflare Data accessed: 16/04/2022
4. M.E Shacklett, *TCP/IP*, TCP/IP: What is TCP/IP, and How Does it Work? (techtarget.com) Data accessed: 20/04/2022
5. Cloudflare, *What is a packet? | Network packet definition*, What is a packet? | Network packet definition | Cloudflare Data accessed: 22/04/2022
6. A. Zola, *network packet*, What are network packets, and how do they work? (techtarget.com) Data accessed: 22/04/2022
7. Cloudflare, *What is MTU (maximum transmission unit)?*, What is MTU (maximum transmission unit)? | Cloudflare Data accessed: 25/04/2022
8. Techopedia, *Protocol Data Unit (PDU)*, What is a Protocol Data Unit (PDU)? - Definition from Techopedia Data accessed: 26/04/2022
9. GeeksForGeeks, *Protocol Data Unit (PDU)*, Protocol Data Unit (PDU) - GeeksforGeeks Data accessed: 29/04/2022
10. N. Shalom, *Three-Way Handshake*, What is a Three-Way Handshake? - Definition from Techopedia Data accessed: 29/04/2022
11. S. Rout, *The Two Generals Problem*, The Two Generals Problem. And why two computers can never… | by Siddharth Rout | Geek Culture | Medium Data accessed: 29/04/2022
12. Code Maze, *Difference Between Asynchronous Programming and Multithreading in C#*, Difference Between Asynchronous and Multithreading in C# (code-maze.com) Data accessed: 10/05/2022

13. Comviva, *All You Need To Know About Network Interoperability*, All You Need To Know About Network Interoperability - Comviva Data accessed: 10/05/2022