HOW IT BEGAN

Link to project:

Contents

Ori Connage
K1720485
Final Year Project How It Began.

Game Overview:

Philosophy

How it began will:

- Provide a first-person experience of a Survival game in medieval time.
- Be visually stunning.
- Be multiplayer.
- Keep you entertained for hours!
- Have two Game modes.

Common Questions

What is the game?

The game, How It Began, will be a 3D first person game built with unreal engine, the genre will lean towards survival, action, adventure with an element of world building. The game will be multiplayer and focus on combat between players. The game will have a medieval theme and have two game modes:

- Last man standing, Will be a player vs player battle royale.
- Creative mode, this mode will also be player vs player but allow the player to build base, gather resources, more in line with the core mechanic of a survival game.

Why creates this game?

My initial idea for my final year project was to create a survival game based in the stone age but from the lack of free assets I changed it to medieval time, I also got into playing survival games like fortnight and State of decay that influenced my thought process for this project.

I also never seen or played a survival game like Rust based in medieval time, so I thought it would be interesting to see how my project will turn out.

What I want how it began to do is tap into our survival instinct giving the player a sense of thrill and adventure as this is a survival game, I want the player to fight for their life and survive by using their wit and combat skills to come out on top.

What is the focus?

The focus changes depending on the game mode, within Last man standing the goal is to be the last player alive.

Whereas in the creative mode the focus is more on exploring the landscape, gathering resources, and building their base up, craft items and battle with other players that stumbles in their path.

What are the mechanics?

When it began will have a complex health system, it will consider Health, Stamina, hunger, thirst which the player must keep in check, when the player hunger or thirst reaches a dangerously low level it will decrease the player health and available stamina.

Stamina is used for the player to perform certain actions like running and attacking.

Ori Connage
K1720485
Final Year Project How It Began.
The game will have a building mechanics which allows the player to build a rudimentary base. I was influenced by Fortnite building style and used that within my game.

Who are the bad guys?

In the last man standing mode, the bad guys will be other players. while there's still other players that wish to harm the player the most dangerous enemy will be the player stomach, hunger and thirst will play an important role in this game mode.

Features:

What are the general features?

- 3D graphic
- Forest landscape
- Medium size world

What are the gameplay features?

- Multiplayer
- Survival
- Hunger/ Thirst system
- Medieval setting
- Building mechanics
- Combat system
- Crafting

MosCow

Must	Should	Could	Wont
Have basic controls: • **E to interact.** • **Right click to attack.** • **Left click to aim.** • **WASD to move around.** • **Shift to sprint** • **C to crouch**	Hit effect. • Blood particle	More building objects Doorways Floors Stairs	
Have basic animation. • **Running** • **Idle** • **Crouching** • **Attacking** • **Hit /struck.** • **Dying**	Score system • Being the last player alive wins	More craft able item Health potion Armour Gear Food Fire	
Multiplayer	Improve interface UI. • Widget for items • Ability to drag and drop items. • Tooltip for when you hover over an item.	Add a Story line: Player Got injured and knocked out on a battlefield, managed to survive by laying under another soldier body, the player wakes up in a foreign land with nothing but	

		his sword in his hand and a wound in his shoulder… how will he survive?	
Basic building mechanic • **Fortnite style building of placing walls**	Improve building system. Ability to snap walls together and check if there is a foundation		
Craft able items • **Bow** • **Arrows** • **Dry meat** • **Swords** • **Spears**	Score system - High score		
Resources • **Wood** • **Raw meat** • **stone**			
Easy to use and understand User interface			
Complex health system • **Health** • **Stamina** • **Thirst** • **Hunger**			
First person perspective			
Basic survival mechanics Degrading stats over time e.g., hunger /thirst			
Different weapons • **Bow** • **Sword** • **Spear** • **crossbow** • **axe**			
Different animation for weapons			
Basic inventory system: • **Ability to Store item** • **Ability to equip and unequip item.** • **Ability to eat /drink edible items.** • **Ability to drop items.**			
Ability to create / host a server			
Ability to join a server			
Ambient sounds • **animal sounds** • **background music**			

Ori Connage
K1720485
Final Year Project How It Began.

Personal Aims and Objectives

My goals for my How it began are to:

- Develop my skill and knowledge in C++.
- Learn more about unreal blueprints.
- Create a fun game.
- Familiarise myself and Implement Agile scrum methodology.
- Create a multiplayer game.
- Implement my ideas.
- Explore different gameplay mechanics.
- Finish this project in time.

Monetization

I plan to upload the game to steam in the future, it will be free to play with an in-game store where the player can purchase gear and emotes

State of the Art Review

With survival game becoming very popular genre within recent years with big titles like Minecraft and Rust making a name for themselves if we look at the core mechanics of each game, we see that they follow a pattern, albeit they may implement them differently a survival game will always incorporate these mechanics.

I've picked two games which I had enjoyed playing and has somewhat influenced what I have implemented for How it began.

State of decay 2

Quick Overview:

State of decay 2(2018), is a massive open world game set within a zombie apocalypse it is a mix match of survival horror and maintenance simulations, while the game does not have a direct story line it's designed in a way that allows the player to play any way, they like.

The creators of state of decay 2 explores an interesting premise - if the civilization as we know it has come to an unfortunate end are we still the same person we were before or can we adapt, evolve, and rebuild ourselves? The Zombie apocalypse can be a blessing or a curse depending on how you look at it and who you ask, but one thing for certain the event was an equaliser, regardless of your previous position or wealth in society weather you were a doctor or a bus driver everyone is now in the same position and can be valuable commodity to the community. While the player can make executive decisions on who to recruit or who will be used a cannon fodder for the zombie's horde, the game makes a point in saying anyone can become a hero – you can encounter a scrappy hungry survivor with horrible traits such as glutton and low stamina and take them on a journey to become a hero leading the charge to rebuild the community.

State of decay 2 also has a great range of diversity with the survivors coming from different cultures and backgrounds making them all the more relatable to the player.

Ori Connage
K1720485
Final Year Project How It Began.

Gameplay:

The gameplay for state of decay 2 is fast and in a constant flow from the moment you start the game. There is no time to relax or appreciate a job well done as there is always something that will require your attention, whether it is the lack of food, or a survivor in the world calling for help, or having to sneak pass hordes of zombies to gather material for a cure because someone in your community caught the plague. I found that the constant need to get something done creates an interesting gameplay loop.

The player controls a group of survivors each with their own personality, desires, and passion where the player has to go on missions, forage for supplies and build the base up to ensure that each survivor's needs are met, each decision the player makes has a lasting effect on the community and could be the reason the community thrive or fall apart.

Mechanics

- Community building
- Melee Combat
- Range Combat
- Drive cars
- Deteriorating equipment/ items
- Skill tree / traits/ specializations
- Base building
- Scavenging
- PVE (Players vs everything)

Rust

Quick overview

Rust is arguably one of the best survivals game out there, with its first coming onto the scene in 2013 and finally coming out of early access in 2018 Rust has firmly consolidated it spot as one of the goats in the genre.

Rust is a multiplayer first-person shooter survival game developed by Face punch studios using the unity game engine where it was originally created to be a clone of another popular survival game Day Z where it has then evolved passed fighting zombie and became its own game by actually removing zombies from the game altogether.

Gameplay

The objective of Rust is to survive in the wilderness by gathering, stealing, or looting materials. Players like in most survival games must micromanage their hunger, thirst, temperature as well as their health or risk death, despite the hostile landscape being residence for numerous nefarious wild animals like bears and wolves the most dangerous enemy that the player will encounter is another player- due to Rust being a multiplayer only game it greatest appeal is the community it has cultivated over the years. The players you encounter are often volatile and unpredictable, every player you meet can either be your killer, your saviour, or a friendly pass a by this created an interesting gameplay.

Ori Connage
K1720485
Final Year Project How It Began.

The game starts with the player waking up in an inhospitable world, which has been procedurally generated, filled with wild animals and roughly 200 other players on a server. The map will have several danger zones that are radioactive – the player must craft, find, or loot gear or clothes that will protect them from the radiation, entering those area unprepared will result in the players death. These areas are high risk high rewards as within these areas the player can find useful gear, material that would be hard to find elsewhere.

An essential part of the gameplay is gathering resources to build a home and to craft items to make survival easier such as tools, weapon, and gear – basic resource can be met by mining for stone or metal ore and cutting trees for wood but for more obscure resources the player must search the map for them. As the player explores the world, they will find blueprint which will allow the player to craft new items which will need hundred of resources to build / upgrade This creates a grind gameplay.

Another interesting feature that rusts implemented was the voice chat. This feature also made the game slightly more ethical as it puts a human behind the pixels, when developers tested this feature out they realised players no longer instantly tried to kill each other, they would have conversations, taunt, become friends or foe. I believe this feature alone contributed to the success of the game.

Mechanics

- Rust uses ballistic trajectory rather than ray cast hit scan creating a more realistic firearm mechanic.
- First person perspective
- Damage modifier – hitting a player in the head will cause more damage.
- Mining – gather resources like stone and metal ore.
- Crafting – using raw material to create a tool, weapon or gear required for survival.
- Hunting – hunting for animal can provide fur for crafting clothes and food.
- Cutting trees – provides wood for crafting.
- Multiplayer –
- Voice chat – the players can communicate with other players using their voice.
- Air drops – these are care packages that are dropped around the map, they are filled with supplies, and can result in other players swarming the location to get the content of the drop.

Survival gameplay.

How it began as a survival game one of my focus is the survival game experience, looking at a variety of different type of survival game I created a list of cliches things a survival game has:

- Crafting system
- Complex health system
- Limited resources around the map
- Food that magically heals you instantly
- Hostile entity – weather its zombies or wild animals

Ori Connage
K1720485
Final Year Project How It Began.

Level design for a survival game

Survival game that are not narrative driven like Rust tend to be procedurally generated – for the map to be procedurally generated there needs to be a set of rules and situations that appear to create that survival game experience, I wanted to see if I can take these rules and apply it to my game.

Using Maslow hierarchy of needs to create a survival gameplay.

Maslow hierarchy of need points out that we five innate needs:

- **Physiological needs** – the need for food, water and sleep are a basic mechanic in survival game.
- **Need for safety** – in survival game we create a sense of danger that the player must overcome to feel safe, for example, in State of decay 2 there are bases that the player can claim as a safe haven, but first they have to overcome the zombie that have taken up resident.
- **Need to feel love / belonging** – we can see in Rust and state of decay the player can form a group and become a community.
- **Need to feel esteem** – in State of decay 2 if you survive long enough you become a hero making the player feel respected by other member of that community.
- **And need for self-actualization.** Self-actualization is harder to design in a game as this purely depends on the player as an individual, but by creating a forum or allowing player to have tutorials can give players a sense of purpose.

How can I utilise the mechanics to create an interesting gameplay?

While creating How it began, I was plagued with the question of how I can use the survival game mechanics to make my gameplay fun, I realised In most survival game they use the food / drink mechanic as a vital component to staying alive, but it does not force the player to make interesting decisions based on those mechanics for example if I run out of food and water, I simply die, which will just forces the player to keep scavenging for food, but if we added an option of an alternative food source something that isn't exactly healthy such as saw dust or perhaps something not ethical like cannibalism we can see how different player will interact with the game.

For the drinking mechanics I can make all types of water drinkable but drinking dirty water will have a probability of giving the player worms causing the player to receive 50% less of the restoration / health they get from eating food. With this mechanic we will see player risking catching worm for survival.

For the crafting mechanics I can implement a button bashing element where it can affect the quality of the item the player crafting, coupled this with the item degrading over time mechanic will create an interesting gameplay.

While I have not implemented these ideas for How it Began yet it's something I would like to add to the game before uploading it to steam.

Ori Connage
K1720485
Final Year Project How It Began.

Tools and Technologies

Engine, Languages, Assets

I will be using Unreal engine 4 to develop my game. The reason I am choosing Unreal, While the interface may not be as simple as unity, unreal has a vast catalogue of features take takes the edge for me, such as:

1. The built-in tools:

Out of the box Unreal engine is more of a complete package whereas in unity you would have to import extension from the marketplace. Unreal provides a vast built-in tools catalogue that makes developing games easier such as:

• Open world tool – Comprise of two tools: The Grass Tool and Procedural Foliage Tool. Which allows you to place static meshes inside your levels using an algorithm to calculate the optimal placement for said meshes. I would be using this tool a lot for my level design as it will not only speed up my workflow, but I can also create a natural looking environment filled with different genus of trees and plants instantly.

• Landscape – As an Open world game When it began would need a terrain the player can transverse. Unreal has a built-in tool that allows to create mountains, valley, cave etc with a range of tools in the sculpt tab.

2. Visual scripting:

You can develop with Unreal two ways C++ codes and with blueprint; with blueprint you can use to create a whole game without writing a single line of code. While running blueprint is slower than compiling dedicated C++ code, unreal has improved the performance of this over the years.

3. Mega scan:

Quixel Megascan is a collection of assets including surfaces, vegetations, objects that have been standardized and physically based it normally accessible with a subscription but for users of unreal engine Mega scan is completely free. I will be using asset from mega scan instead of the marketplace for my project.

Ori Connage
K1720485
Final Year Project How It Began.

Design

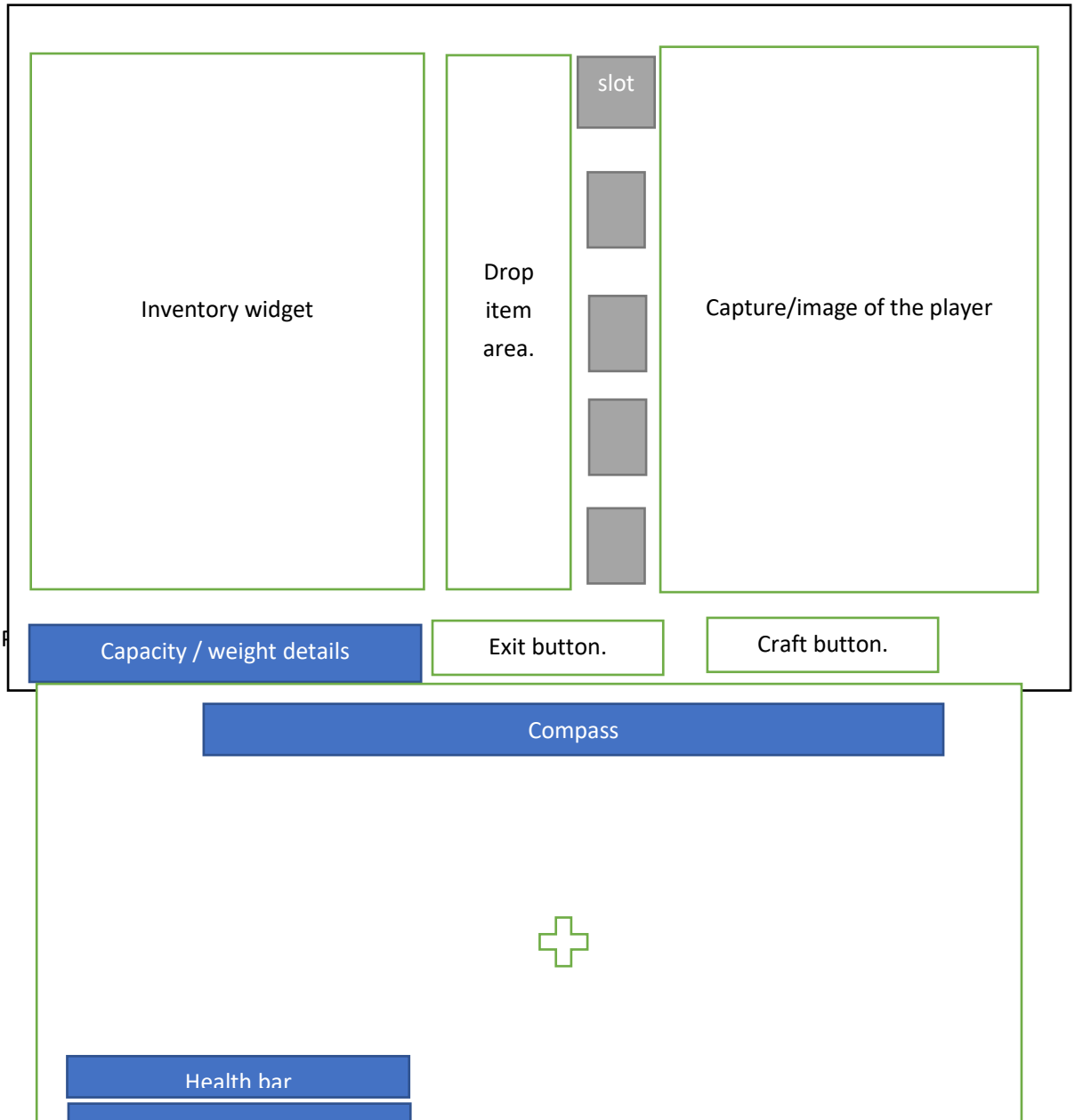
Conceptual design:

As I have mention earlier within the document How It Began is a survival game set within the medieval times which brings about the notion of how you would survive in the wilderness with nothing but primitive tools or knowledge of modern technology that you now know.

There is not a story line for how it began yet, but it is something that I can see being implemented in the future. The gameplay that I have designed for this project is purely focused on combat and the survival mechanics.

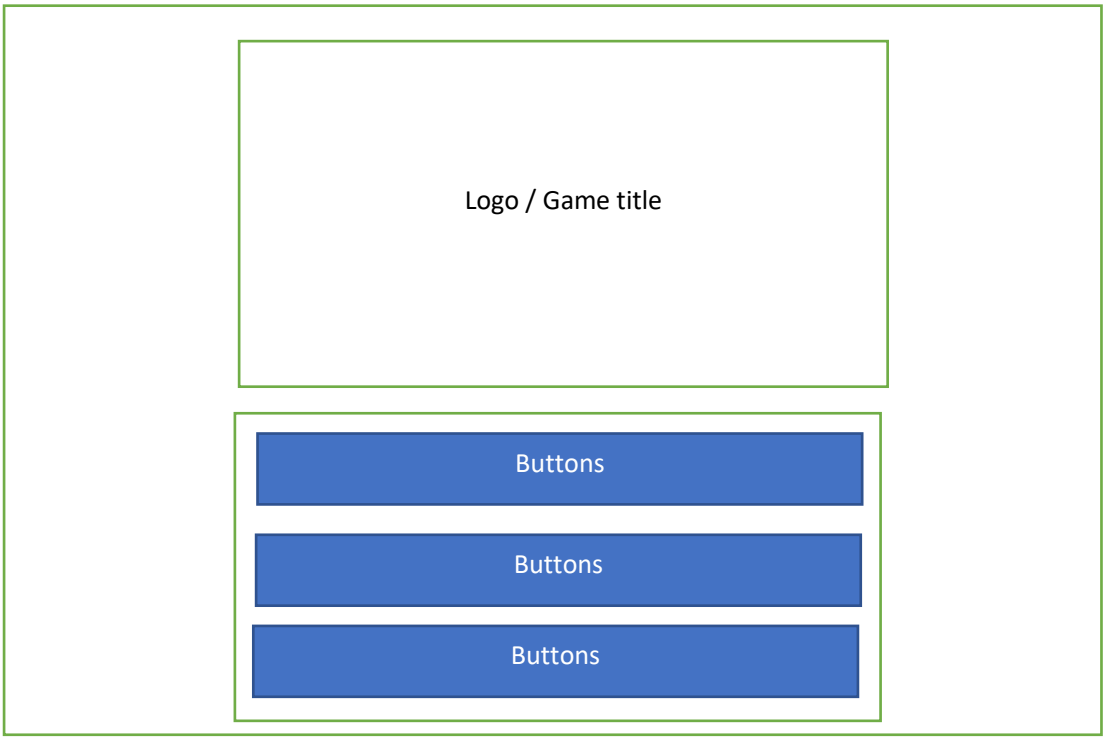
something I spent a lot of time working on was the user interface, I wanted to create an interface that is somewhat stylish but not at the expense of ease of use. the first interface the player will encounter will be the HUD – here the player will see their health, stamina, thirst level and hunger level, they will even see what weapon they are wielding as well as the amount of ammo they have. I've created interaction widget that will pop up when the player looks at an object prompting them to interact with it, I have also created a inventory UI that list all of the items inside of the player inventory.

<u>Player inventory wireframe</u>



Ori Connage
K1720485
Final Year Project How It Began.

Main menu widget



The appearance of the game matches the medieval theme I had in mind- the “Last man standing map” is set within a forest at night, I used volumetric fog to create a god rays effect with the moon light creating an cool night time atmosphere, although there is no animal within the levels I have implemented ambient audio of several night time animal such as owls and foxes I have also paid attention to details by in adding a few assets like lanterns around the forest. I created spawn point around the map where random loot will spawn – this ranges from food to gear.

Ori Connage
K1720485
Final Year Project How It Began.

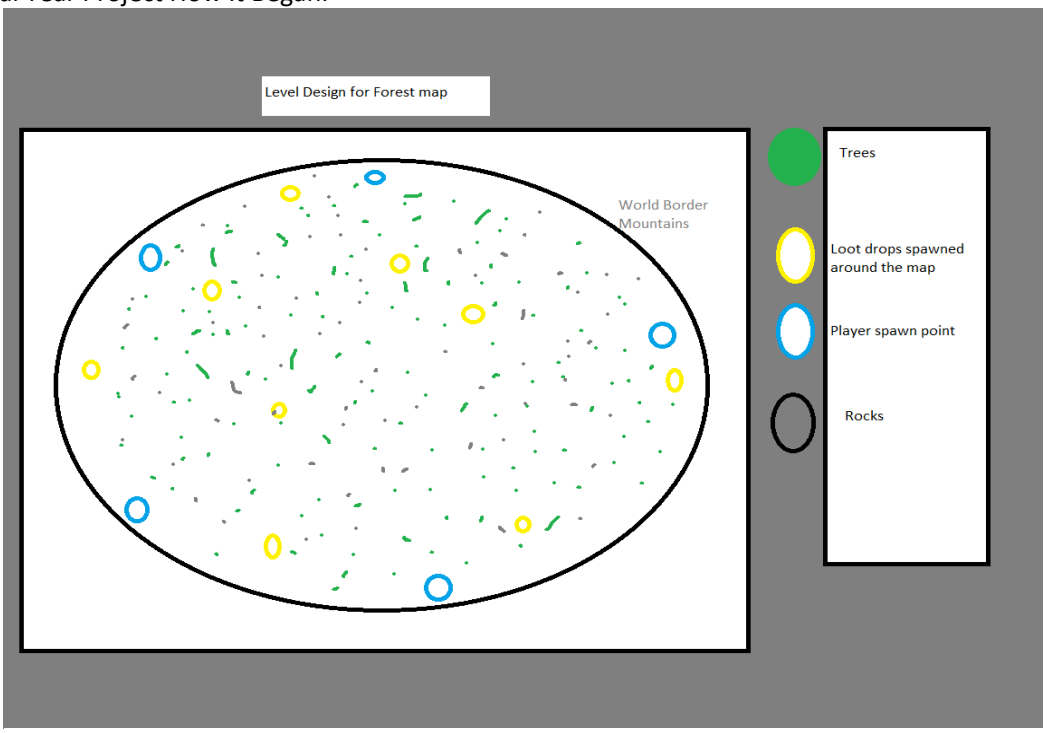
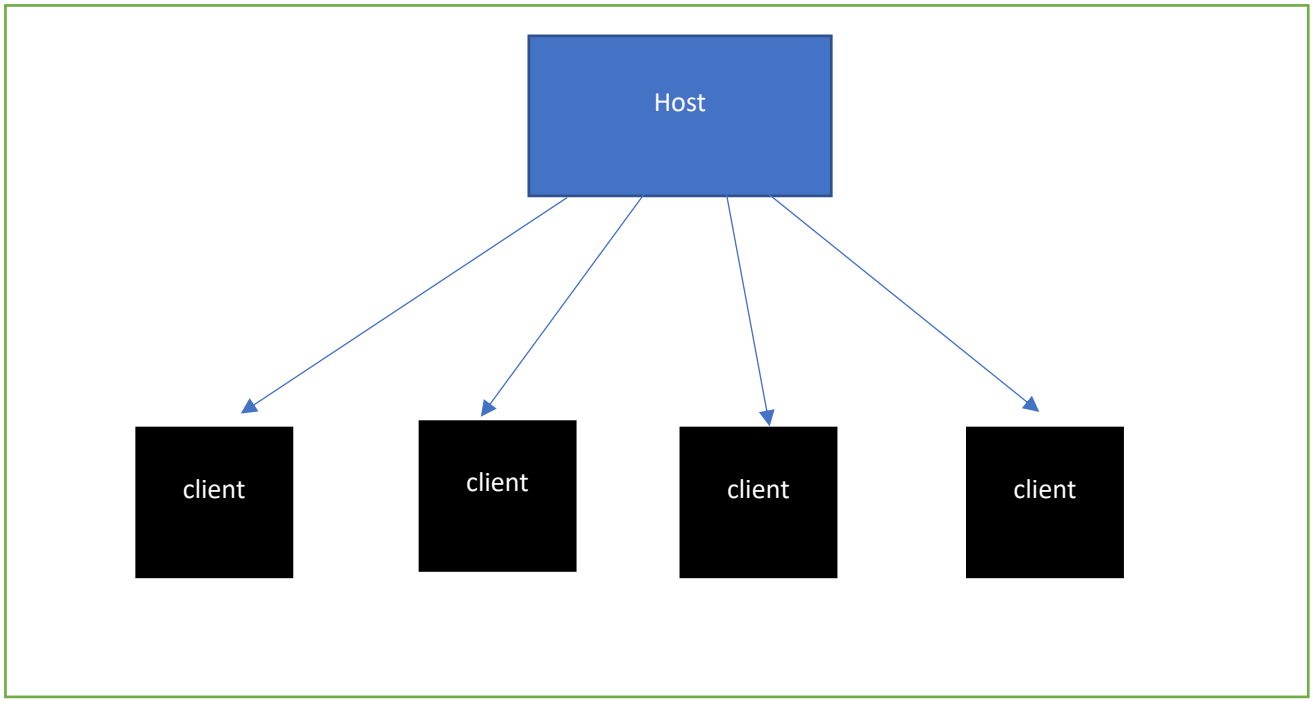


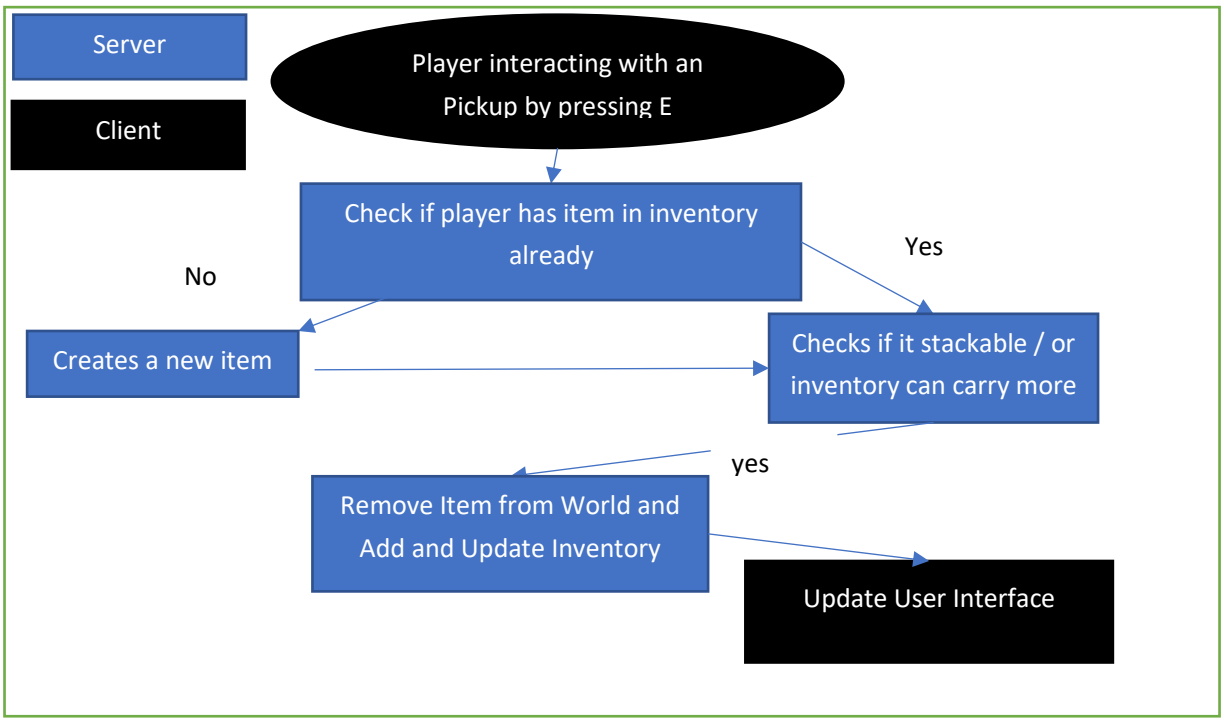
Figure 1 Level Design for Last Man Standing

Technical design

As a multiplayer game, How It Began will be Server authoritative meaning that the game will run on the server and that player will only be able to control their pawn. Unreal engine uses a client server model meaning one computer acts as the server and host a session which all the other client and the network can join. So, while the server processes the gameplay the client just updates the user on what is happening.



Ori Connage
K1720485
Final Year Project How It Began.
How the server manages the Inventory:



Implementation

Development Methodology

Progression

To monitor my progression with this project I've been using a Gannt chart coupled with the Moscow chart to benchmark my progression and to see if I can implement a feature within the allocated time slot.

development

Using the agile scum methodology was one of the best decisions I've taken on this project, not only is it inexpensive to implement, it allowed me to break my targets into segment – which had to be completed within a allocated time span usually ranged from two weeks to a month. This allowed me to implement the important mechanic first, review and test if it works then move on to the next requirement from the Moscow chart.

While I used both C++ and Blueprint for this project, in this documentation I will be focusing on my implementation from the C++ side.

Implementation of the Building system

A survival game needs to have some building elements, whether you're building to protect yourself from the elements or to provide a safe shelter from other players having the ability to build four walls is a must. For this game I opted for the fortnite style of building.

Ori Connage
K1720485
Final Year Project How It Began.
The implementation of the build system is fairly simple, I start of by creating a building component and adding it to my player character.

```
25      //Build component
26      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Building")
27      class UBuilderComponent* BuildComponent;
28
```

within the building component I set up some value for the grid size , build distance the floor position and a class reference to the buildable object aka the wall I also have a reference to the character class

```
public:
    // checking if building
    bool isBuilding;
    //Variables
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Building")
    float GridSize = 100.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Building")
    float BuildDist = 500.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Building")
    float FloorPosition = 130.f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Building")
    TSubclassOf<class ABuildable> BuildableItems;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Player")
    TSubclassOf<class ACharacterBaseClass> player;

};
```

Ori Connage
K1720485
Final Year Project How It Began.
Toggle build mode function changes the is building Boolean when toggled.

```
void UBuilderComponent::ToggleBuildMode()
{
    isBuilding = !isBuilding;

/*  if(APlayerCharacter* OwnerCharacter = Cast<APlayerCharacter>(GetOwner()))
    {
        if(OwnerCharacter && OwnerCharacter->IsLocallyControlled())
        {
            if(AMyPlayerController* OwnerController = Cast<AMyPlayerController>(OwnerCharacter->GetController()))
            {
                //OwnerController->ToggleBuildMode();
            }
        }
    }*/
    GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green,
    FString::Printf( TEXT("ToggleBuildMode -> %d"),isBuilding));
}
```

request build calls the build function in the buildable class

```
void UBuilderComponent::RequestBuild()
{
    if (isBuilding == !isBuilding || currrentBuild ==nullptr)
    {
        return;
    }
    GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("RequestBuild"));
    currrentBuild->build();
    currrentBuild = nullptr;
}
```

```
void ABuildable::build()
{
    if (!HasAuthority())
    {
        Severbuild();
    }

    BuildMesh->SetVisibility(true);
    BuildMesh->PlayAnimation(BuildAnimation, false);
    CollisionVol->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
    PreviewMesh->PlayAnimation(DestroyAnimation,false);
    PreviewMesh->DestroyComponent(true);
}

void ABuildable::Severbuild_Implementation()
{
    build();
}

bool ABuildable::Severbuild_Validate()
{
    return true;
}
```

we use raycast to get the position/ rotation of the object and snap it to place if the raycast hits another object

Ori Connage
K1720485
Final Year Project How It Began.

```
FVector UBuilderComponent::GetNextNodePos() const
{


    FHitResult OutHit;
    FVector EyesLoc = PlayerCamera->GetComponentLocation();
    FRotator EyeRot;
    //Thischaracter->GetController()->GetPlayerViewPoint(EyesLoc, EyeRot);

    FVector Start = EyesLoc;
    FVector directionVector = PlayerCamera->GetForwardVector();
    FVector End = ((directionVector * BuildDist) + Start);

    FCollisionQueryParams Queryparams;
    // ignore the player
    //Queryparams.AddIgnoredActor();

    bool Hit = GetWorld()->LineTraceSingleByChannel(OutHit, Start, End, ECC_Visibility, Queryparams);

    //checking if we hit an Object
    if (Hit) {
        if (OutHit.GetActor())
        {
            return OutHit.Location.GridSnap(GridSize);
        }
        else
        {
            return End;
        }
    }
    else {
        return End;
    }

    /**
    directionVector += GetOwner()->GetActorLocation();
    return FVector(
        FMath::GridSnap(directionVector.X, GridSize),
        FMath::GridSnap(directionVector.Y, GridSize),
        FloorPosition
    );
    **/
}
```

Implementation of the Inventory system

Introduction:

One thing a survival game must have is a scalable inventory system, the role that an inventory system can have on a survival game where the player must manage their inventory can seem like a fun minigame at times or frustrating to the player. In How it Began I do not put much emphasis on the management of the inventory but more on the functionality of the inventory such as:

- the player will use their inventory component to equip or unequip gear.
- The ability store items.
- eat /drink food from the inventory.
- craft items from the inventory.
- and even drop items for other players to pick up.

from this you can see how the player interaction with the inventory system is vital to the gameplay.

Things I had to think about while implementing the inventory system:

- since How It Began is a multiplayer game the inventory needed to be cheat proof and networked to other players.
- I needed the ability to add an inventory to anything – to a player, loot able chest or even a rock if I wanted to.
- The User interface – will it be easy to use and how will it meet the requirement?
- How am I going to code the inventory item? am I going to use actors? Structs or even a UObject? I ended up going with UObject due to the nature of an inventory item, they don't exist in the world for it to be an actor, I don't want to use struct because I want to be able to create items from the editor plus from reading the unreal documentation for Struct it mentions that UStructs aren't considered for replication and if I wanted more complicated interactions within my projects UObject was my best bet.

Creating the Item class and setting it up for replication

Before even starting to code the inventory, I have to code the inventory items, I created a UObject class called Item Object which store data about what the item is, such as: the name, the thumbnail, the rarity, weight, and whether or not it's stackable or if the item is craft able.

```
35          Quantiy = 1;
36      }
37  };
38
39  UCLASS(Abstract, Blueprintable, EditInlineNew, DefaultToInstanced)
40  class HOWITBEGAN_API UItemsObject : public UObject
41  {
42      GENERATED_BODY()
43  protected:
44
45      virtual void GetLifetimeReplicatedProps(TArray<class FLifetimeProperty>& OutLifetimeProps) const override;
```

Figure 2 - ITEM Object UCLASS Tags

Within the UCLASS I added the following tags:

Abstract – specify that the item class will be an abstract base class, this would allow me to create different item classes that can be added to the level. Such as, Gear items , weapon items, and edible item.

Blueprintable – which would allow us to create blueprint of the item class

EditInlineNew – the object of this class can be created from the editor window

Unlike the actor class replicating in a UObject is kind of tricky we have to call and override the Get life time replicated props and the " is supported for networking()" functions

Ori Connage
K1720485
Final Year Project How It Began.

```
void UItemsObject::GetLifetimeReplicatedProps(TArray<class FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);

    DOREPLIFETIME(UItemsObject, Quantity);
}

bool UItemsObject::IsSupportedForNetworking() const
{
    return true;
}

UWorld* UItemsObject::GetWorld() const
```

For isSupportedForNetworking we return true, this is how Unreal checks if we want this UObject to be networked.

Then for the GetLifeTimeReplicatedProps we tell Unreal that we want the quantity value to be replicated this also means that the server is going to manage this value and that when the server changes the quantity it going to be sent to the player over the network. This also makes cheating by using a glitch or code to multiply items quantity harder for the player to do.

```
protected:

    /**The amount of the item*/
    UPROPERTY(ReplicatedUsing = OnRep_Quantity, EditAnywhere, Category = "Item", meta = (UIMin = 1, EditCondition = bStackable))
        int32 Quantity;
};
```

Every time the quantity get changed by the server it will tell the client by calling the OnRep_Quantity function

```
}

void UItemsObject::OnRep_Quantity()
{
    OnItemModified.Broadcast();
}
```

OnItemModified is a multicast delegate I declared which bounds to the User interface in blueprints – so whenever the item update OnItemModified is triggered which updates the UI. I used broadcast() due to is being safe to call on a multicast delegate even if nothing is bound.

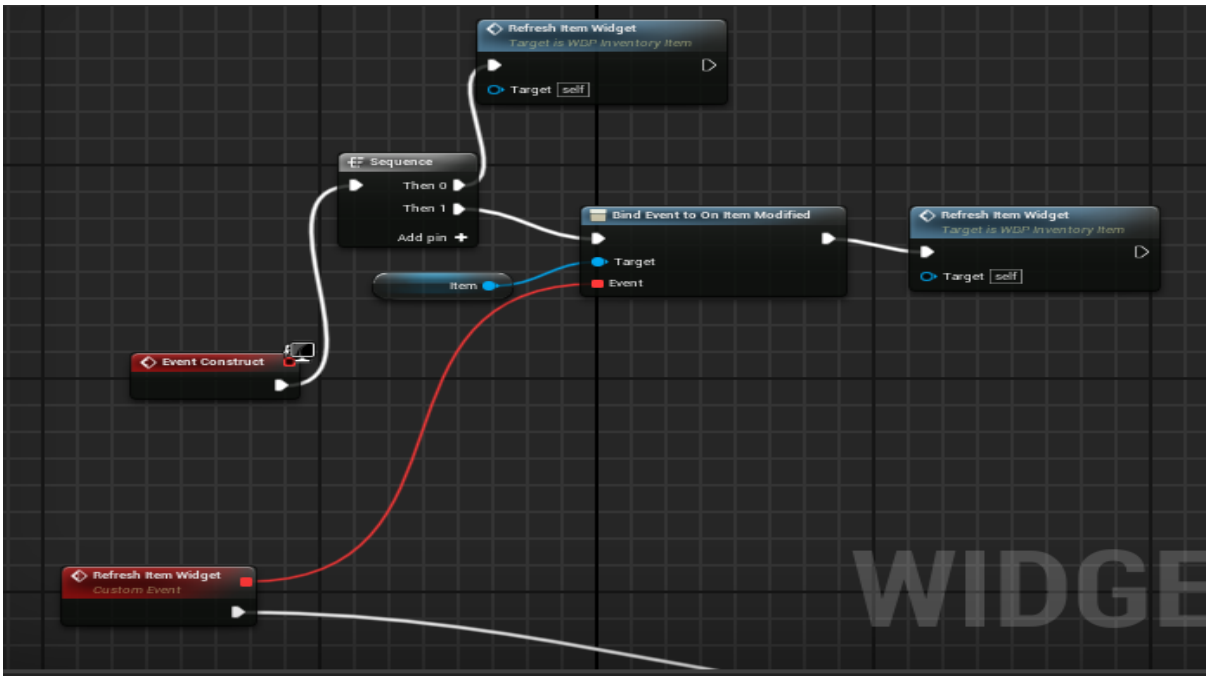


Figure 5 OnItemModifed Usage

Repkey is a key that we change whenever we change something about the item, for example when the quantity update, we change the Repkey, so the server knows when to update the item on the client side.

We update the repkey using the function MarkDirtyForReplication() we also replicate the item within the inventory component her as well – I will go into more details later on in the document when I talk about implementing the inventory system.

```
void UItemsObject::MarkDirtyForReplication()
{
    //Mark this object for replication
    ++RepKey;

    //Mark the array for replication
    if (OwningInventory)
    {
        ++OwningInventory->ReplicatedItemsKey;
    }
}
```

Creating different items

The Edible item class inherits from the ItemObject class and overrides the Use function.

Ori Connage
K1720485
Final Year Project How It Began.

```
UCLASS()
class HOWITBEGAN_API UEdibleItems : public UItemsObject
{
    GENERATED_BODY()
public:
    UEdibleItems();

    /**The amount for the food to heal*/
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Healing")
    float HealAmount;

    /**The amount of stamina to restore*/
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Restoring")
    float StaminaAmount;

    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Restoring")
    float Hungeramount;

    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Restoring")
    float ThristAmount;

    virtual void Use(class APlayerCharacter* Character) override;


    class UMaterialItems* MatItems;
};
```

Figure 2 Edible Item class

```
void UEdibleItems::Use(APlayerCharacter* Character)
{

    UE_LOG(LogTemp, Warning, TEXT("NUM Food Eating"));
    if (Character)
    {
        const float ActualHealedAmount = Character->PLayerStatsComp->ModifyHealth(HealAmount);
        const float ActualStaminaAmount = Character->PLayerStatsComp->ModifyStamina(StaminaAmount);
        const float ActualHungerAmount = Character->PLayerStatsComp->ModifyHunger(Hungeramount);
        const float ActualThirstAmount = Character->PLayerStatsComp->ModifyThirst(ThristAmount);

        /*
        *  Check how much the food actual modify the player stats and if it is close to zero
        *  that means that player doesn't need to eat
        *
        */
        const bool bNeedHealth = !FMath::IsNearlyZero(ActualHealedAmount);
        const bool bNeedStamina = !FMath::IsNearlyZero(ActualStaminaAmount);
        const bool bNeedFood = !FMath::IsNearlyZero(ActualHungerAmount);
        const bool bNeedWater = !FMath::IsNearlyZero(ActualThirstAmount);
        if (!Character->HasAuthority())
        {
            if (AMyPlayerController* PC = Cast<AMyPlayerController>(Character->GetController()))
            {
                if (bNeedHealth||bNeedFood||bNeedStamina||bNeedWater)
                {
                    PC->ClientShowNotification(FText::Format(LOCTEXT("AteFoodText", "Ate {FoodName}, healed {HealAmount} health."), DisplayName, ActualHealedAmount));
                }
                else
                {
                    PC->ClientShowNotification(FText::Format(LOCTEXT("FullHealthText", "No need to eat {FoodName}, health is already full."), DisplayName, HealAmount));
                }
            }
        }

        if (bNeedHealth || bNeedFood || bNeedStamina || bNeedWater)
        {
            if (UInventoryComponent* Inventory = Character->PlayerInventory)
            {
                Inventory->ConsumeItem(this, 1);
            }
        }
    }

}
```

Figure 3 Edible item Use function

Ori Connage
K1720485
Final Year Project How It Began.

Equippable items

the Equippable item class is a base class for the Gear and weapon item, it inherits from the Item objects class. I created an Enum class to name the different types of slots that the player can equip items to.

```
//All the slots that gear can be equipped to.
UENUM(BlueprintType)
enum class EEquippableSlot : uint8
{
    EIS_Head UMETA(DisplayName = "Head"),
    EIS_Helmet UMETA(DisplayName = "Helmet"),
    EIS_Chest UMETA(DisplayName = "Chest"),
    EIS_Arm UMETA(DisplayName = "Arm"),
    EIS_Legs UMETA(DisplayName = "Legs"),
    EIS_Skirt UMETA(DisplayName = "Skirt"),
    EIS_Hands UMETA(DisplayName = "Hands"),
    EIS_Pauldron UMETA(DisplayName = "Pauldron"),
    EIS_Cloak UMETA(DisplayName = "Cloak"),
    EIS_Backpack UMETA(DisplayName = "Backpack"),
    EIS_PrimaryWeapon UMETA(DisplayName = "Primary Weapon"),
    EIS_Throwable UMETA(DisplayName = "Throwable Item"),


};
/**
 *
 */
UCLASS(Abstract, NotBlueprintable)
class HOWITBEGAN_API UEquippableItem : public UItemsObject
{
    GENERATED_BODY()
public:
    UEquippableItem();

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Equippables")
    EEquippableSlot Slot;

    virtual void GetLifetimeReplicatedProps(TArray<class FLifetimeProperty>& OutLifetimeProps) const override;

    virtual void Use(class APlayerCharacter* Character) override;

    UFUNCTION(BlueprintCallable, Category = "Equippables")
    virtual bool Equip(class APlayerCharacter* Character);

    UFUNCTION(BlueprintCallable, Category = "Equippables")
    virtual bool UnEquip(class APlayerCharacter* Character);

    virtual bool ShouldShowInInventory() const override;
    virtual void AddedToInventory(class UInventoryComponent* Inventory) override;

    UFUNCTION(BlueprintPure, Category = "Equippables")
    bool IsEquipped() { return bEquipped; };

    /**Call this on the server to equip the item*/
    void SetEquipped(bool bNewEquipped);

protected:

    UPROPERTY(ReplicatedUsing = EquipStatusChanged)
    bool bEquipped;
    UFUNCTION()
    void EquipStatusChanged();
};
```

```
}
void UEquippableItem::GetLifetimeReplicatedProps(TArray<class FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);

    DOREPLIFETIME(UEquippableItem, bEquipped);
}
```

The value that I want the server to manage is the bEquipped.

Ori Connage
K1720485
Final Year Project How It Began.

when the bEquipped status has changed the client calls the equipStatusChange() function

```
void UEquippableItem::EquipStatusChanged()
{
    if (APlayerCharacter* Character = Cast<APlayerCharacter>(GetOuter()))
    {
        UseActionText = bEquipped ? LOCTEXT("UnequipText", "Unequip") : LOCTEXT("EquipText", "Equip");

        if (bEquipped)
        {
            Equip(Character);
        }
        else
        {
            UnEquip(Character);
        }
    }

    //Tell ui to update
    OnItemModified.Broadcast();
}
```

I use a ternary operator which acts like an if statement to change the use action text from unequip to Equip and vice versa and to equip or unequip the item from the character body.

This then called the OnItemModified multicast delegate to update the User interface.

Gear items

```
*/
UCLASS(Blueprintable)
class HOWITBEGAN_API UGearItem : public UEquippableItem
{
    GENERATED_BODY()
public:

    UGearItem();

    virtual bool Equip(class APlayerCharacter* Character) override;
    virtual bool UnEquip(class APlayerCharacter* Character) override;

    /**The skeletal mesh for this gear*/
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Gear")
    class USkeletalMesh* Mesh;

    /*material instance to apply to the gear*/
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Gear")
    class UMaterialInstance* MaterialInstance;

    /**The amount of defence this item provides. 0.2 = 20% less damage taken*/
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Gear", meta = (ClampMin = 0.0, ClampMax = 1.0))
    float DamageDefenceMultiplier;
};
```

The gear items Overrides the equip and unequip function, it takes a skeletal mesh and a material instance to apply to a gear.

Ori Connage
K1720485
Final Year Project How It Began.

Weapon items

```
 */
UCLASS(Blueprintable)
class HOWITBEGAN_API UWeaponItem : public UEquippableItem
{
    GENERATED_BODY()
public:
    virtual bool Equip(class APlayerCharacter* Character) override;
    virtual bool UnEquip(class APlayerCharacter* Character) override;

    //The weapon class to give to the player upon equipping this weapon item
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Weapon")
    TSubclassOf<class ABow> BowClass;
};
```

Takes a reference to the weapon class which I incidentally named ABow although I created all type of weapons from that class such as sword, crossbow, spear, and the Great axe.

I will talk more about the weapon class later within this document.

Creating the inventory system

As I mentioned I wanted the ability to add the inventory system to any actor/pawn in my level - while I could build the inventory system into an actor it would make much more sense to create a component and add it to an actor.

As I want the inventory system to be server authoritative I called the SetIsReplicated() function and set it to true in the constructor, this enables every clients on the server to knows what inside another player inventory and that they can loot from that player inventory upon death.

Create an Array of ItemObject and replicate using the OnRep_Item() function and this function just updates the User interface in the inventory system.

```
protected:

    //The maximum weight the inventory can hold. For players, backpacks and other items increase this limit
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Inventory")
    float WeightCapacity;

    //The maximum number of items the inventory can hold. For players, backpacks and other items increase this limit
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Inventory", meta = (ClampMin = 0, ClampMax = 200))
    int32 Capacity;

    /**The items currently in our inventory*/
    UPROPERTY(ReplicatedUsing = OnRep_Items, VisibleAnywhere, Category = "Inventory")
    TArray<class UItemsObject*> Items;
```

To enable the ItemObject to replicate across the server we need two functions:

Ori Connage
K1720485
Final Year Project How It Began.

GetLifeTimeReplicatedProps() a function we already implemented before and ReplicateSubObjects() this is where the magic happens – this function replicate the UObject class in the inventory component replication channel

I added a bool check to make replicating more efficient instead of having each item constantly replicating. So bUpdate check whether we have updated the channel then we check if the inventory key been updates if it has we iterates through the array of items Objects and checks if any items needs to be replicated

```cpp
void UInventoryComponent::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);

    DOREPLIFETIME(UInventoryComponent, Items);
}

bool UInventoryComponent::ReplicateSubobjects(class UActorChannel* Channel, class FOutBunch* Bunch, FReplicationFlags* RepFlags)
{
    bool bUpdated = Super::ReplicateSubobjects(Channel, Bunch, RepFlags);

    //Check if the array of items needs to replicate
    if (Channel->KeyNeedsToReplicate(0, ReplicatedItemsKey))
    {
        for (auto& Item : Items)
        {
            if (Channel->KeyNeedsToReplicate(Item->GetUniqueID(), Item->RepKey))
            {
                bUpdated |= Channel->ReplicateSubobject(Item, *Bunch, *RepFlags);
            }
        }
    }

    return bUpdated;
}
```

Due to the blueprint not being able to read the values for the array of item or the capacity and weight capacity I created a function to expose it.

```cpp
//Get the current weight of the inventory. To get the amount of items in the inventory, just do GetItems().Num()
UFUNCTION(BlueprintPure, Category = "Inventory")
float GetCurrentWeight() const;

UFUNCTION(BlueprintCallable, Category = "Inventory")
void SetWeightCapacity(const float NewWeightCapacity);

UFUNCTION(BlueprintCallable, Category = "Inventory")
void SetCapacity(const int32 NewCapacity);

UFUNCTION(BlueprintPure, Category = "Inventory")
FORCEINLINE float GetWeightCapacity() const { return WeightCapacity; };

UFUNCTION(BlueprintPure, Category = "Inventory")
FORCEINLINE int32 GetCapacity() const { return Capacity; }

UFUNCTION(BlueprintPure, Category = "Inventory")
FORCEINLINE TArray<class UItemsObject*> GetItems() const { return Items; }
```

I created a struct call FItemAddResult and this struct basically returns the results of adding an item, for example when we have a stack of 10 apple but the inventory capacity/ weight limit only 5 it will return the result of that.

Ori Connage
K1720485
Final Year Project How It Began.

```
public:

    FItemAddResult() {};
    FItemAddResult(int32 InItemQuantity) : AmountToGive(InItemQuantity), AmountGiven(0) {};
    FItemAddResult(int32 InItemQuantity, int32 InQuantityAdded) : AmountToGive(InItemQuantity), AmountGiven(InQuantityAdded) {};

    //The amount of the item that we tried to add
    UPROPERTY(BlueprintReadOnly, Category = "Item Add Result")
    int32 AmountToGive = 0;

    //The amount of the item that was actually added in the end. Maybe we tried adding 10 items, but only 8 could be added because of capacity/weight
    UPROPERTY(BlueprintReadOnly, Category = "Item Add Result")
    int32 AmountGiven = 0;

    //The result
    UPROPERTY(BlueprintReadOnly, Category = "Item Add Result")
    EItemAddResult Result = EItemAddResult::IAR_NoItemsAdded;

    //If something went wrong, like we didnt have enough capacity or carrying too much weight this contains the reason why
    UPROPERTY(BlueprintReadOnly, Category = "Item Add Result")
    FText ErrorText = FText::GetEmpty();

    //Helpers
    static FItemAddResult AddedNone(const int32 InItemQuantity, const FText& ErrorText)
    {
        FItemAddResult AddedNoneResult(InItemQuantity);
        AddedNoneResult.Result = EItemAddResult::IAR_NoItemsAdded;
        AddedNoneResult.ErrorText = ErrorText;

        return AddedNoneResult;
    }

    static FItemAddResult AddedSome(const int32 InItemQuantity, const int32 ActualAmountGiven, const FText& ErrorText)
    {
        FItemAddResult AddedSomeResult(InItemQuantity, ActualAmountGiven);

        AddedSomeResult.Result = EItemAddResult::IAR_SomeItemsAdded;
        AddedSomeResult.ErrorText = ErrorText;

        return AddedSomeResult;
    }

    static FItemAddResult AddedAll(const int32 InItemQuantity)
    {
        FItemAddResult AddAllResult(InItemQuantity, InItemQuantity);

        AddAllResult.Result = EItemAddResult::IAR_AllItemsAdded;

        return AddAllResult;
    }

};
```

The try add item function is called for items that are already in the inventory while the try add item from class is for new items that aren't in the inventory. You can see that I created a new item instance from the class

```
FItemAddResult UInventoryComponent::TryAddItem(class UItemsObject* Item)
{
    GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("Try add item"));
    return TryAddItem_Internal(Item);
}

FItemAddResult UInventoryComponent::TryAddItemFromClass(TSubclassOf<class UItemsObject> ItemClass, const int32 Quantity /*=1*/)
{
    UItemsObject* Item = NewObject<UItemsObject>(GetOwner(), ItemClass);
    Item->SetQuantity(Quantity);
    return TryAddItem_Internal(Item);
}
```

Ori Connage
K1720485
Final Year Project How It Began.

Item Spawn class

As the name intel this is the class that generates random items to spawn into the game – the item spawn is a target point class and reads data from a TableRowBase, the table contains data of the items and the probability of that item spawning.

```cpp
void Aitemspawn::SpawnItem()
{

    if (HasAuthority() && LootTable)
    {
        TArray<FLootTableRow*> SpawnItems;
        LootTable->GetAllRows("", SpawnItems);

        const FLootTableRow* LootRow = SpawnItems[FMath::RandRange(0, SpawnItems.Num() - 1)];

        ensure(LootRow);

        float ProbabilityRoll = FMath::FRandRange(0.f, 1.f);

        while (ProbabilityRoll > LootRow->Probability)
        {
            LootRow = SpawnItems[FMath::RandRange(0, SpawnItems.Num() - 1)];
            ProbabilityRoll = FMath::FRandRange(0.f, 1.f);
        }

        if (LootRow && LootRow->Items.Num() && PickupClass)
        {
            float Angle = 0.f;

            for (auto& ItemClass : LootRow->Items)
            {
                if (ItemClass)
                {
                    const FVector LocationOffset = FVector(FMath::Cos(Angle), FMath::Sin(Angle), 0.f) * 50.f;

                    FActorSpawnParameters SpawnParams;
                    SpawnParams.bNoFail = true;
                    SpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButAlwaysSpawn;

                    const int32 ItemQuantity = ItemClass->GetDefaultObject<UItemsObject>()->GetQuantity();

                    FTransform SpawnTransform = GetActorTransform();
                    SpawnTransform.AddToTranslation(LocationOffset);

                    APickUp* Pickup = GetWorld()->SpawnActor<APickUp>(PickupClass, SpawnTransform, SpawnParams);
                    Pickup->InitializePickup(ItemClass, ItemQuantity);
                    Pickup->OnDestroyed.AddUniqueDynamic(this, &Aitemspawn::OnItemTaken);

                    SpawnedPickups.Add(Pickup);

                    Angle += (PI * 2.f) / LootRow->Items.Num();
                }
            }
        }
    }
}
```

Health system

the server manages the values for the Thirst, Hunger, Health, Stamina

```cpp
void UPlayerStatsComponent::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);

    DOREPLIFETIME(UPlayerStatsComponent, Thirst);
    DOREPLIFETIME(UPlayerStatsComponent, Hunger);
    DOREPLIFETIME(UPlayerStatsComponent, Health);
    DOREPLIFETIME(UPlayerStatsComponent, Stamina);
}
```

Ori Connage
K1720485
Final Year Project How It Began.

I clamp the value between the min and the max value and return the difference between the current value and the original value to get the change.

```cpp
float UPlayerStatsComponent::ModifyHealth(const float Delta)
{
    const float oldHealth = Health;

    Health = FMath::Clamp<float>(Health + Delta, 0.f, MaxHealth);
    return Health - oldHealth;
}

float UPlayerStatsComponent::ModifyStamina(const float Delta)
{
    const float oldStamina = Stamina;

    Stamina = FMath::Clamp<float>(Stamina + Delta, 0.f, MaxStamina);
    return Stamina - oldStamina;
}

float UPlayerStatsComponent::ModifyHunger( float Delta)
{
    const float oldHunger = Hunger;

    Hunger = FMath::Clamp<float>(Hunger + Delta, -100.f, MaxHunger);
    return Hunger - oldHunger;
}

float UPlayerStatsComponent::ModifyThirst( float Delta)
{
    const float oldThirst = Thirst;

    Thirst = FMath::Clamp<float>(Thirst + Delta, -100.f, MaxThrist);
    return Thirst - oldThirst;
}
```

When Health is less than 50, it sets the max stamina to 50

```cpp
// Called every frame
void UPlayerStatsComponent::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction)
{
    Super::TickComponent(DeltaTime, TickType, ThisTickFunction);

    //Penalty for low health
    (Health < 50) ? MaxStamina = 50 : MaxStamina = 100;

    if (isThisSurvivalPlayer) {
        if (APlayerCharacter* Character = Cast<APlayerCharacter>(GetOwner())) {

            if (Hunger <= 0)
            {
                Character->Damage((Hunger * -2), FDamageEvent(), Character->GetController(), Character);
            }
            if (Thirst <= 0)
            {
                Character->Damage((Thirst * -2), FDamageEvent(), Character->GetController(), Character);
            }
        }
    }
}
```

Ori Connage
K1720485
Final Year Project How It Began.

Testing

As I was using agile scum method throughout this project testing been happening from the moment, I started implementing each mechanic.

Black Box Testing

Test Name	Expected output	Actual Output	Comment
Decreasing Hunger / Thirst	Hunger and thirst decrease by N amount at intervals	Worked as expected	
Health Decrease when thirst /hunger goes below 0	When hunger or thirst goes below zero, it decreases the player health by the value of the negative Thirst/Hunger	Worked as expected	
Toggle Inventory	Pressing Tab key opens the inventory	Worked as expecting	
Is Inventory focusable?	When we open the inventory the mouse cursor and able to interact with the widget	Mouse was not focusable	Had to make the widget focusable and turn cursor on and set the input mode to UI
Can I rotate the 360 view of the player in player capture - inventory?	Player should be able to rotate the image of themselves to get a 360 view of the gear they are wearing	Works as expected	
Craft menu	Can we transition to the craft menu from the inventory	Works	
Does crafting use consume the required material?	When we clicked a button to craft the material required should be consumed	Worked as expected	
Does crafting tooltip appear?	When the user hovers their mouse over the crafting menu item a tool tip should appear informing them of the required material	Did not work	Still in the progress of fixing this – I moved on to get another feature done.
Does crafting add item to inventory?	Crafting an item add it to the inventory	Works as expected	
Adding Items to the inventory	Picking up an item add it to the inventory	Works as expected	
Removing items from inventory	Dropping an item removes the item from the inventory	Works as expected	
Stackable Items	Items that are stackable can stack	Works as expected	

Ori Connage
K1720485
Final Year Project How It Began.

Non-Stackable item	Item that are not stack able don't stack	Works as expected	
Equipping items	Able to equip items. Tested using the sword and leg plates	Worked as expected	
Unequipping items	Able to unequipped items	Worked as expected	
Does Item Tool tip appear?	When the mouse hover over an item in the inventory a tool tip appears	Works as expected	
Can pick up?	Can we pick up the spawn able object	Works as expected	
Player name in inventory	Player name should be in the inventory	Works as expected	
Max inventory capacity	Player cants exceed max capacity	Works as expected	
Max inventory weight capacity	Player can't over encumbered	Works as expected	
Does interaction widget appear?	When player looks at an interactable object a widget appears on screen	Works as expected	
Toggle build mode	Pressing B toggle build mode on and off	Works as expected	
Does buildable preview mesh show?	Preview mesh should show	True	
Does the preview mesh animate?	Mesh should animate	Didn't work as expected	
Does the building show?	Build mesh should show after the preview mesh disappear	Works as expected	
Max Speed	Max speed is 600 while sprinting	true	
Does sprinting reduce stamina?		Worked as expected	
When Stamina Low can player sprint?	When the player stamina is low they shouldn't be able to sprint till the stamina regenerate	Worked as expected	
Does Eating restore Health	Eating food will restore health	Works as expected	
When the player health is full can he eat?	When the player has full health, they can't eat	Works as expected	
Player health below 50%	When the health bar is below 50% the player stamina get reduced by 50%	Works as expected	
Does eating restore stamina	Eating food will restore stamina	Works as expected	

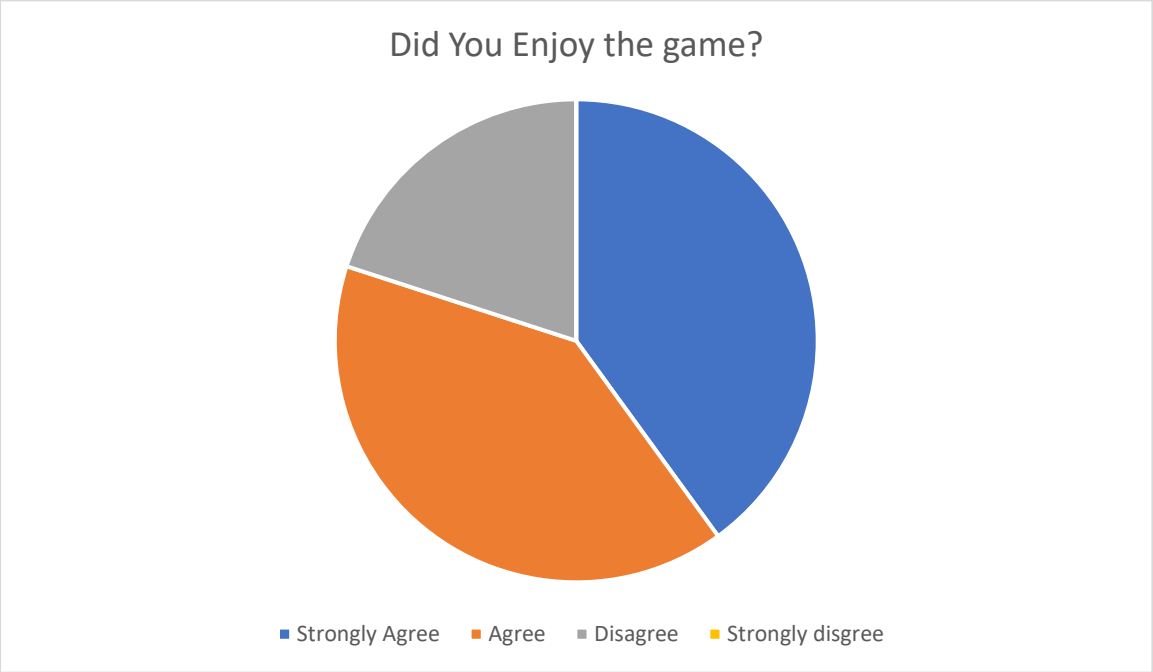
Ori Connage
K1720485
Final Year Project How It Began.

Does Eating restore hunger	Eating food will restore hunger	Works as expected	
Does eating restore thirst	Eating food will restore Thirst	Works as expected	
On death event	On Death the killed screen will appear and send the player to the main menu screen	Worked as expected	
Pressing E to spectate	Pressing E should make the player spectate the game	Did not work as expected	Moved on to focus on more important features
Is movement replicated?	Is movement replicated	Worked as expected	
Is animation replicated?	Animation should be replicated to clients	Works as expected	
Changing weapons – change in animation	Different attack animation for different weapon – should be replicated across network	Can of work	The attack animation only get replicated when the player hit another player
Weapon Damage player?	Should be able to damage player	Works as expected	
Player spawning at different player start	Player should be able to spawn at different player starts	Works as expect	
Item spawner Testing			
Item spawn : common item	Should be the most common items on the map	true	
Item Spawn: uncommon	Should be the second most common items on the map	True	
Item Spawn: Rare items	Should be rarely found on the map	True	
Item spawn: legendary item	Possibility of finding one should be extremely low	true	
Can user make a server/ host?	Player can make a server for another player to join	Work as expected	
Max player	Player can set a max amount of player who can join their server	Works as expected	
Can user join a max server?	User should not be able to join a maxed-out server	Works as expected	
Can user join a game?	User should be able to join game from the lobby	Did not work. {Update} works now	I forgot to add “?listen”

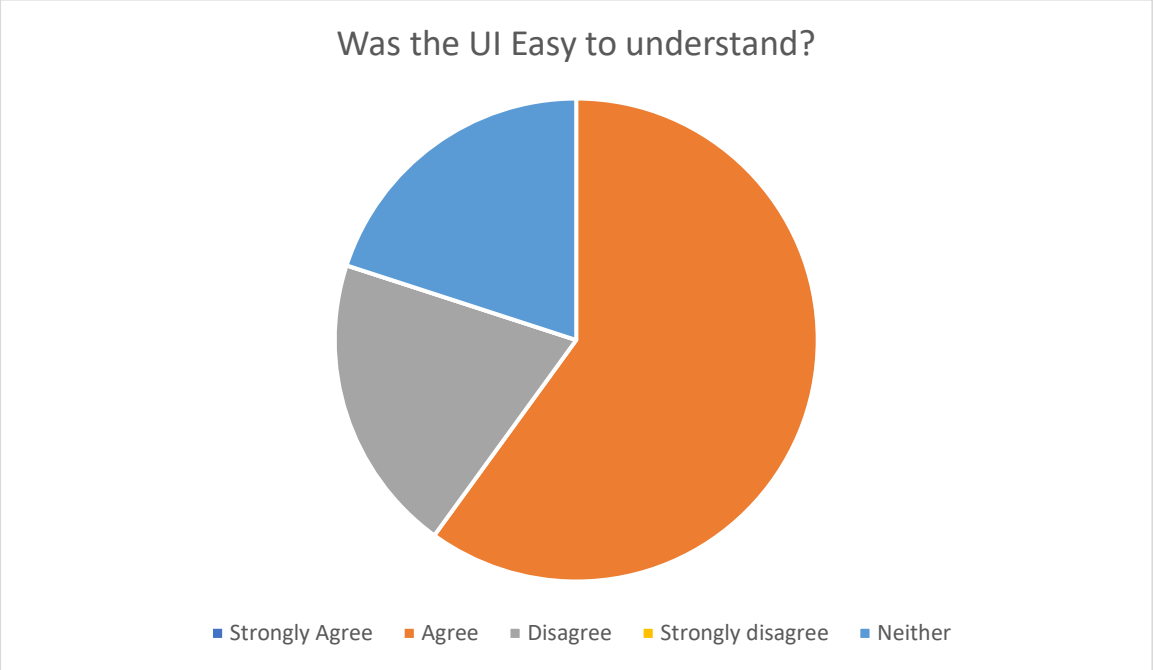
Ori Connage
K1720485
Final Year Project How It Began.

Player testing

Due to the global situation with coronavirus, I wasn't able to get my pairs to test my game, but I fortunately had a small sample size of 5, (family and friend) who may be biased but kindly offered to participate in my testing. During this test I did not inform the player on how to play and just let them explore and workout the mechanics on their own, I also asked each player 4 question where they can answer with strongly agree, Agree, strongly disagree, disagree or neither. The question the asked are:

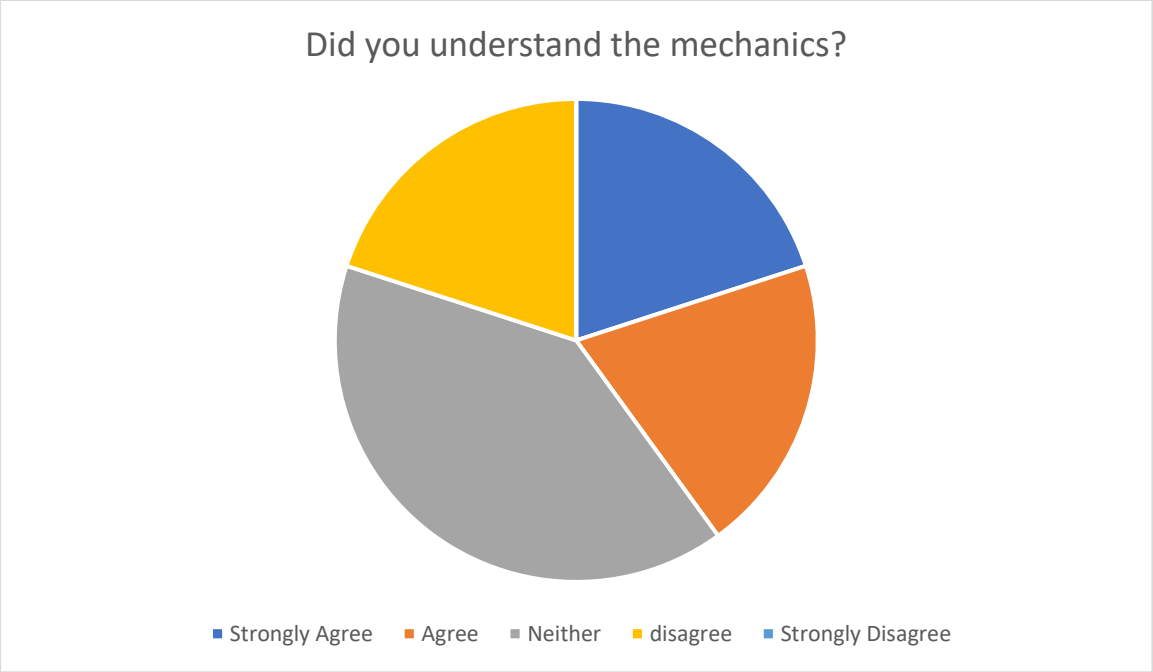


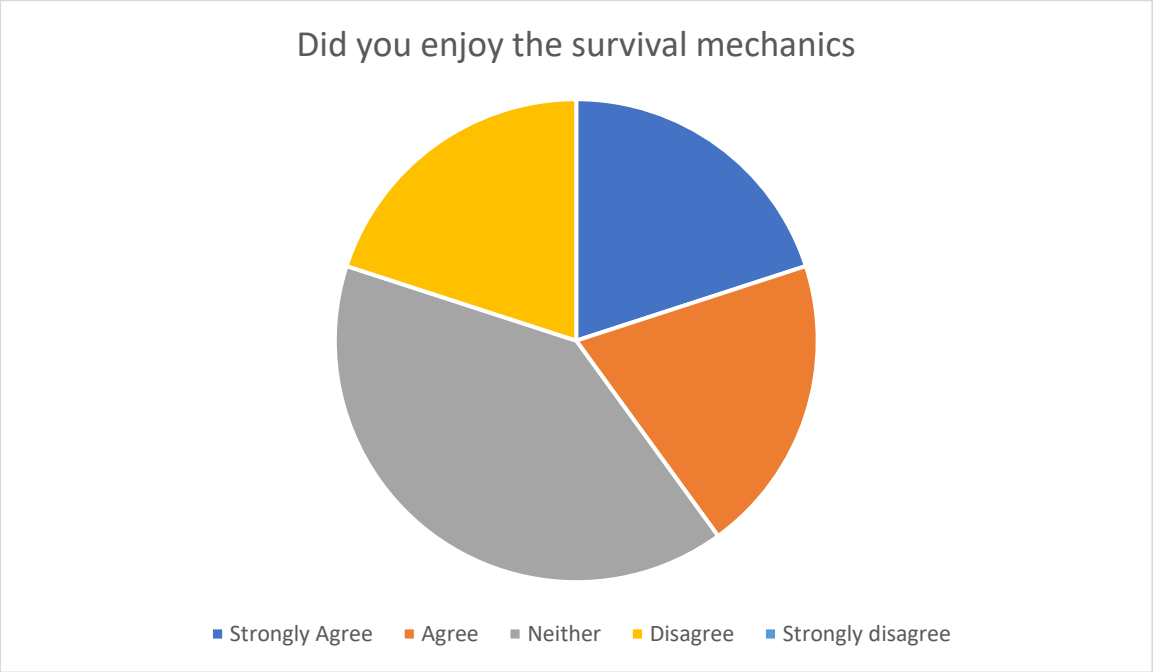
We can see that the game was enjoyable for the tester



From the data we can see that the User interface was easy to understand for majority of the testers, I asked the ones that said thought otherwise what they thought was and they mention that they didn't understand how the craft menu works.

Ori Connage
K1720485
Final Year Project How It Began.





Ori Connage
K1720485
Final Year Project How It Began.

PEGI Rating, copyright, ethical issues

PEGI

The likely PEGI rating for this game would be is 18, due to the graphic violence toward humanoid characters with sound effects that puts emphasis on the suffering the character is going through.

Ethical issue

When developing a game, you must think about the character morality and how it can potentially influence the player aggression, previous studies have shown that a character appearance, clothing and identity can influence the gamer attitude and behaviour for example, when the player kills for a moral reason such as self-defence the player aggression level will rise but compared to a player who kills indiscriminately the level of aggression is much lower. How it began is a battle royale survival game, that pits player against player but just like rust will implement voice chat feature that will put a human behind the pixels reducing the violence player will do to each other.

Copyright

The intellectual property in my game will be sourced from the asset stores and Megascan, from reading the term and conditions the license I obtained is the personal license which is allowed for freelancers / individuals who work alone, I'm allowed to use Megascan asset in my project. I will also be importing characters from Mixamo, from my research I can use the character rig and animation royalty free for personal and commercial uses.

When I release my proposed game, unreal engine requires me to credit them by adding the following line to my product credits: "[Product Name] uses Unreal® Engine. Unreal® is a trademark or registered trademark of Epic Games, Inc. in the United States of America and elsewhere. Unreal® Engine, Copyright 1998 – xxxx, Epic Games, Inc. All rights reserved."

Security implications and data protection

As a networked game how it began need to think about what data that it takes from the player, as this game is design to be uploaded to steam, the in game use of the player data will be their steam username so we can identify each player, at this point in time I don't have any micro transaction that would require the player card details. The precaution I have taken to protect the user information is to use steam as steam will handle the transaction and protect the user details.

Ori Connage
K1720485
Final Year Project How It Began.

Critical Review and conclusion

Summary of achievements

Although the end result is different from what I originally proposed for my project which was a story base survival horror game set in the stone age which transitioned to a multiplayer survival game set in the medieval period, I managed to make a lot of achievements such as, players can join and create a lobby, player can wield different type of weapons, player can craft, and player can build walls. players can interact with the environment by picking up items.

Personal achievement

As someone who loves a challenge, I picked a game engine I didn't have much experience in to develop my game. This was my second time using Unreal Engine, while the learning curve was steep, I used How it began as a tool to sharping my knowledge in Unreal engine, C++, and blueprint while I develop a fun game.

Throughout the process I learnt:

- How the network interacts with the clients.
- The difference between a dedicated server and a listen server.
- Editor hacking – Changing the unreal editor with code.
- Using Unreal engine components / classes
- Design patterns
- Agile methodology
- Building User interface
- Replicating

Time management

While changing the direction of the project was not the smartest thing to do in the mid development stage which caused a lot of delays trying to learn new things while I implement the basic again, I managed to use the Gantt chart and MosCow chart to prioritise the more important mechanics.

I also had issue with Unreal engine with my computer being somewhat old it kept on crashing – and for the longest time I was not able to design a landscape due to unreal crashing while using the sculp tool which delayed me a lot as a survival game mechanic is based on the landscape.

Possible Improvements

While I had a thorough plan I didn't quite stick to it which led me to changing direction with my game which then caused further delay for me. If I was to go back and retry again this time I would stick with the plan.

Future work

How it began at it current form has a decent foundation that need to be built upon before I can upload it to steam. My next step is to first iron out the mechanics for resource gathering, now resources just

spawn randomly around the map which doesn't really provide that survival experience as I'm literally giving the player resources instead of them working for it, for example one of my improvement will be to create a node of stone that the player will have to mine to get stone and to smelt to get any type of metal ore and another will be to have tree that the player can cut for wood.

I also need to create more items that cater to a specific need as at the moment the food item restore all of the player stats.

I also need to add more buildable objects, allowing the player to have more variety for their building. I can also implement a damageable building mechanic which allows player to damage or even break another player base.

I need to add more craft able item such as a bow, health potion, bandage, shield.