TDD

Contents

Game Overview

A Native American themed, action-adventure set in an alternate reality prior to European colonisation. In this reality, everyone has an external soul in the form of a spirit animal. You, as the player, have the unique ability to change your spirit animal at will. Whilst mostly story driven, there is a big focus on relationships with neighbouring tribes, trading resources and waging wars over land.

Technical Summary

This project has a total development time of 11 months with a vertical slice demo to be released after 3 months, a team of 11 people will be working on the project, these include 3 programmers and 8 designers. The game will be made using unreal engine 5 and with assets used from the Unreal Engine marketplace and made using Blender and MetaHuman. Textures will be created using a mixture of: Krita, Photoshop, Gimp and Gravit. Sounds will be taken from freesound.org and self recorded. Sound editing will be done using Audacity. Landscapes will be made using Gaea. The game will be developed for Windows 10 PC.

Platform and system limitations

The game will run with a frame rate of 60, as it is open world there will be limited loading screens with the main one being on player death when the area around the respawn point is loaded.

Consideration of applications

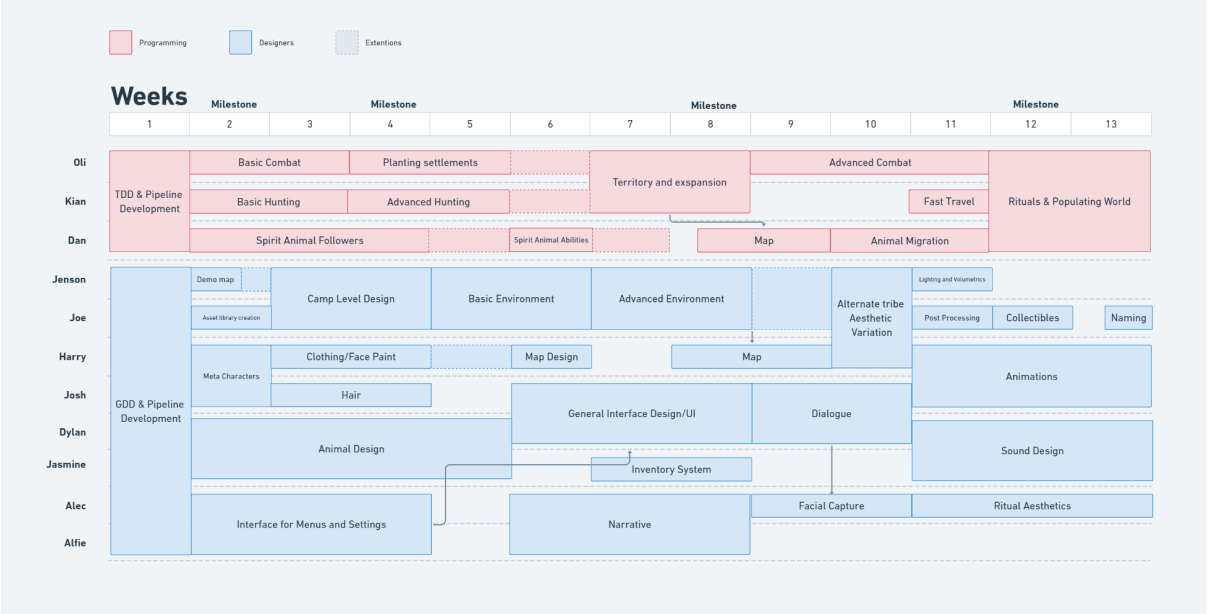
The latest version of ue5 was used for development as it provides the highest level of detail control for the game world with more realistic effects than programs used at the university. This is important for this project as the changing weather is a key game element and ue5 is the engine that can handle this best. Using metahumans as well as gaea to create our own assets with AAA studio. Using quixel bridge to obtain pre-made assets for weaponry.

Workflow

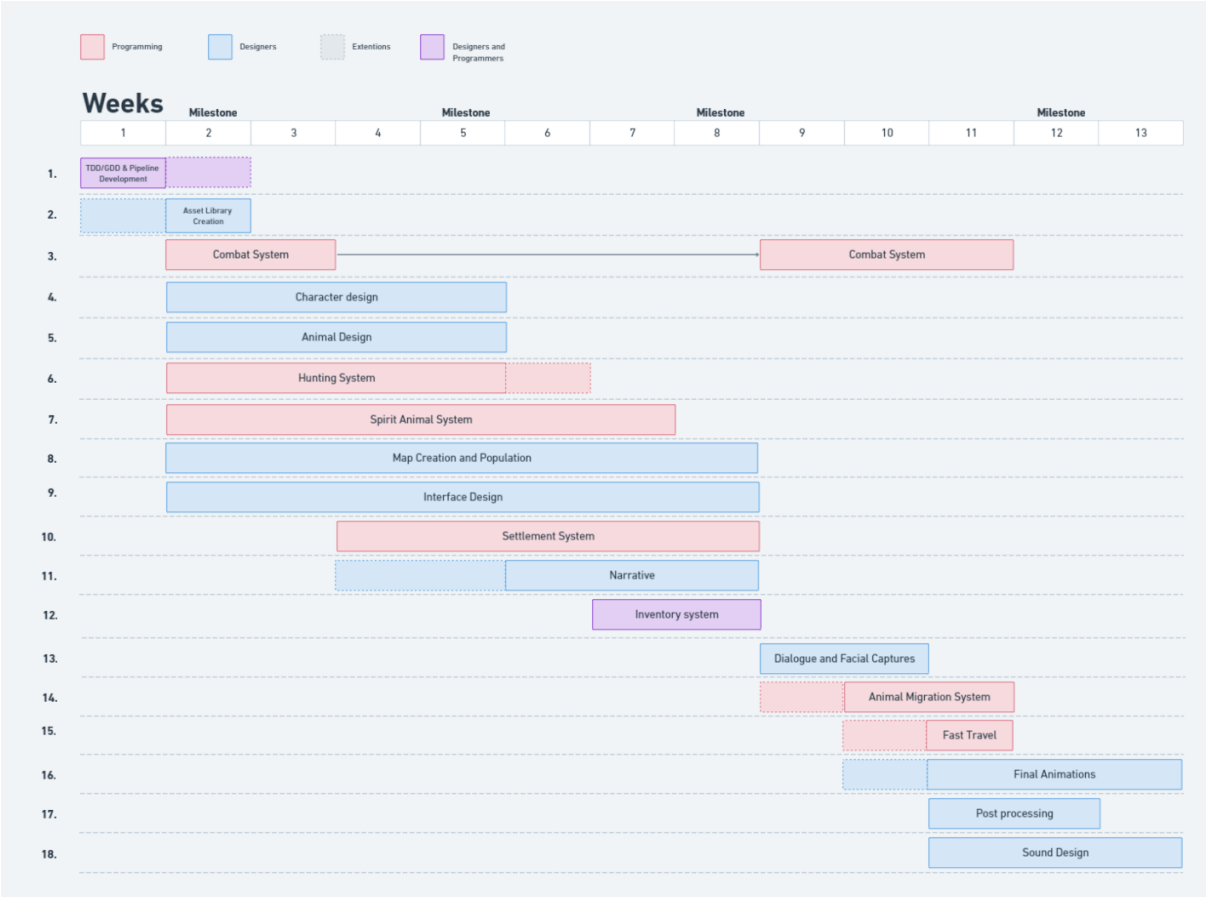
The Agile methodology is going to be used for the workflow as each of the programmers will be working on different mechanics and updating them till they are to a suitable standard and then moving onto the next mechanic. This will repeat until the game is completed.

Scheduling

The original organisation chart:

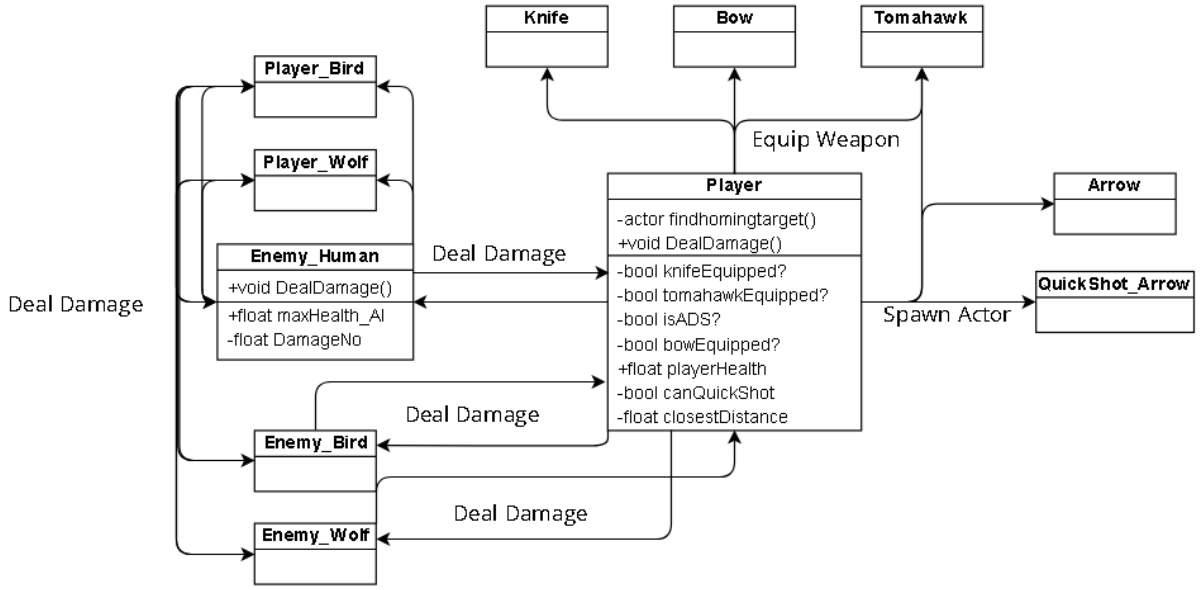


Updated gantt chart:



Core Systems

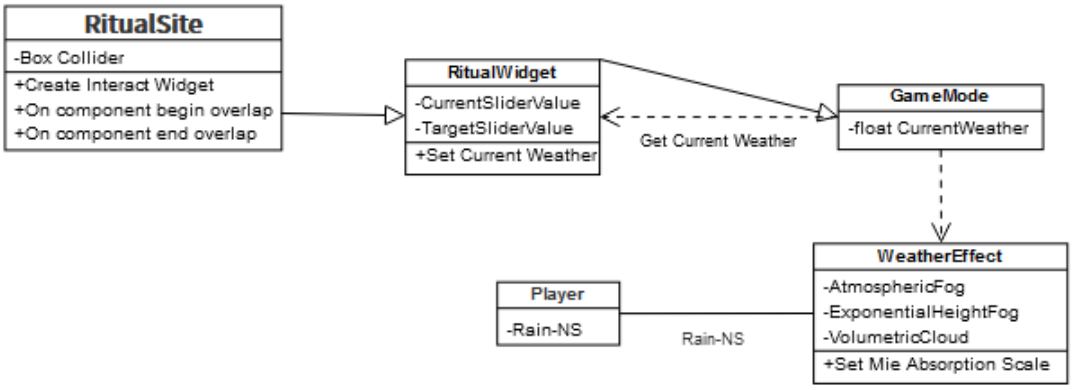
Combat System



The combat system in project tomahawk consists of three equipable weapons: a knife, a tomahawk and a bow each with its own combat mechanic. This is achieved by having an input action event for each weapon type and attaching the corresponding actor to a socket on the player's skeleton when the key is pressed. When each weapon is equipped / de-equipped a boolean variable is changed, these variables are used to determine which weapon is equipped and therefore what should happen when the player presses the attack button. Another mechanic to note is aiming, when the player presses the aim button whilst either the tomahawk or the bow is equipped then the player's FOV will be decreased, movement speed is decreased and a crosshair appears on the screen. If the knife is equipped and the quick attack button is pressed, a short-range knife attack will be performed. If the player has the bow or the tomahawk equipped and they are aiming in, then the appropriate projectile is fired. To ensure an accurate projectile combat system, just before the projectile is spawned a ray cast is performed out of the centre of the player's camera, the location of the hit is used to find a look at rotation between this and the player's location. The player's rotation is then set. What this does is snap the rotation of the player to wherever the player's crosshair is pointing, ensuring that the projectile always goes where the player is aiming. Another combat mechanic with the bow is the quick shot. A box collider is attached to the player and collisions are checked with the enemy human, if a collision occurs and the player presses the attack button without aiming in then the player's rotation snaps to face the closest enemy and an arrow is fired at it. This is achieved by creating a function to find the current target, the function works by getting all actors of class with tag and feeding this array into a for each loop. Within the loop the distance between the player and the enemy is compared and the closest one is assigned to a variable. This variable is used within the quick attack event to find the look at rotation between the closest enemy and

the player, and then using this to set the actor's rotation before spawning the arrow (note: this mechanic was cut from the final game). The way the damage detection works in project tomahawk is by assigning colliders to each actor involved. On collision overlap is used on each target to check for collisions and depending on which weapon overlapped a damage number is set and then the deal damage function is called. This function subtracts the damage number from the target's health, and checks whether health is less than or equal to zero, if this is the case then death logic is run. This logic is used within the hunting mechanic as well and the damage system can be easily changed to implement new weapons if they are added. Within the game, damage can be dealt between the player and its spirit animals and the enemy and its spirit animals. Damage dealt to the spirit animals is applied to the owner with a modifier. When the enemy dies an animation is played and a lootable death box is spawned at the location of its body. When the player dies an animation is played, a death screen appears and the player is then teleported back to their closest settlement. If the player does not have a settlement built then they will be teleported to the location of the friendly village. This was achieved by storing the location of the totems in the settlements in an array. The distance between the player and the totems is compared and the actor's location is set to the closest one upon death (with a small offset to ensure the player doesn't spawn inside of the totem), if the array is empty then the spawn location will use the default value of the array which is the location of the friendly village.

Rituals



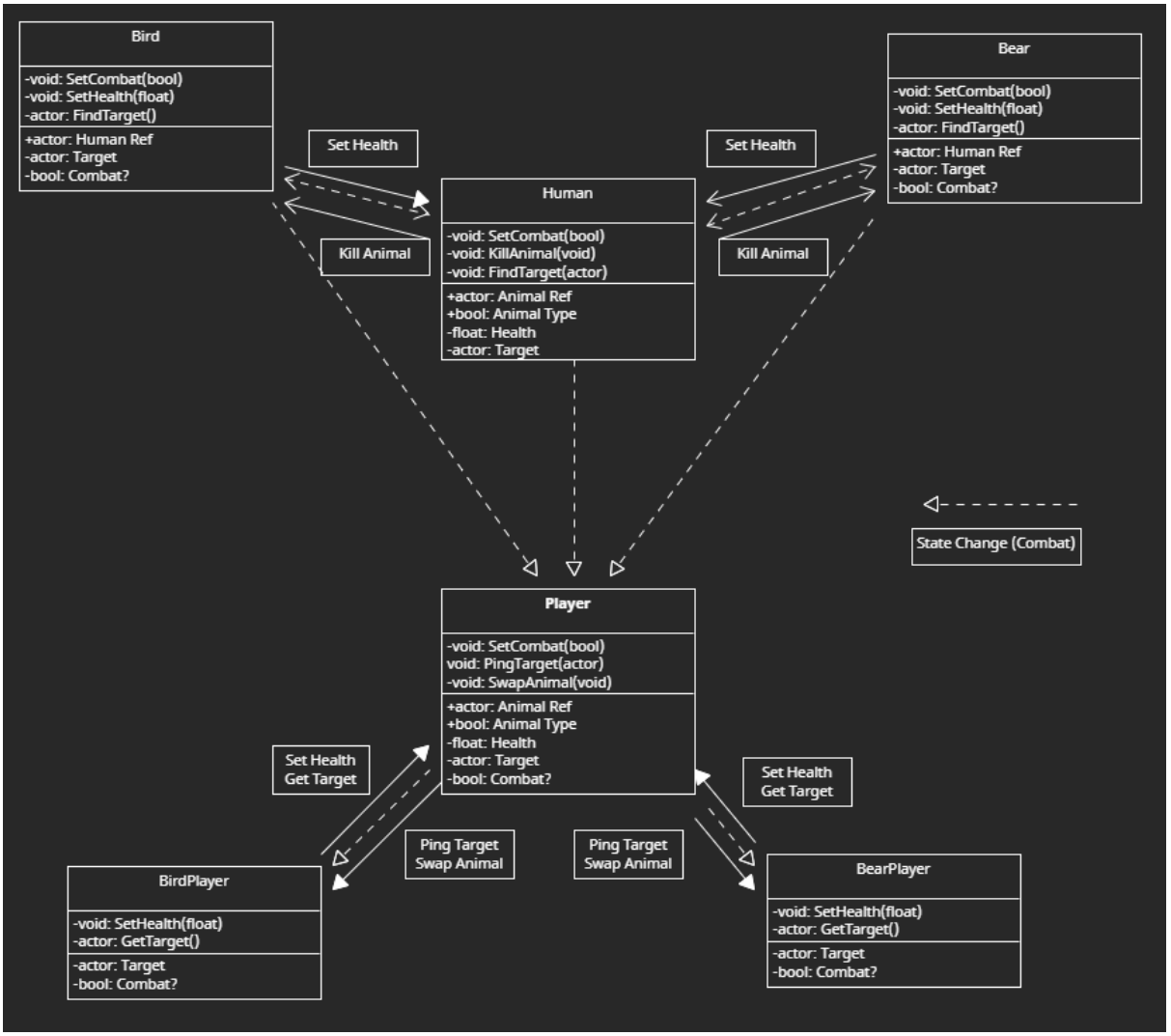
The “Ritual” system is an important mechanic featured in the game, it has the ability to interact and change many aspects of the game when used. The main concept is that the player will use this ritual site to perform a ritual that changes the weather based on its intensity. Alongside the weather change comes many benefits towards the player but can also come as a risk. For example, when the weather is set to a foggy, rainy, atmosphere, the

enemies alertness and visibility will decrease significantly, allowing for the player to manipulate the flow of the game and adopt their strategy to a more stealth-focused approach. These effects also reach beyond just combat. Another use of the ritual system is that it could help the settlement with crop production by having a balanced weather, changing from rain to sunny.

The ritual system is controlled by interacting with a ritual site, found dotted around the map. Once opened, a GUI will display to the players screen, showing the current weather set, represented by a moveable radial slider. There are two buttons featured above the slider that sets the weather to either 0 (sunny and clear) or 100 (cloudy and raining), but the weather can be adjusted linearly with the slider as well.

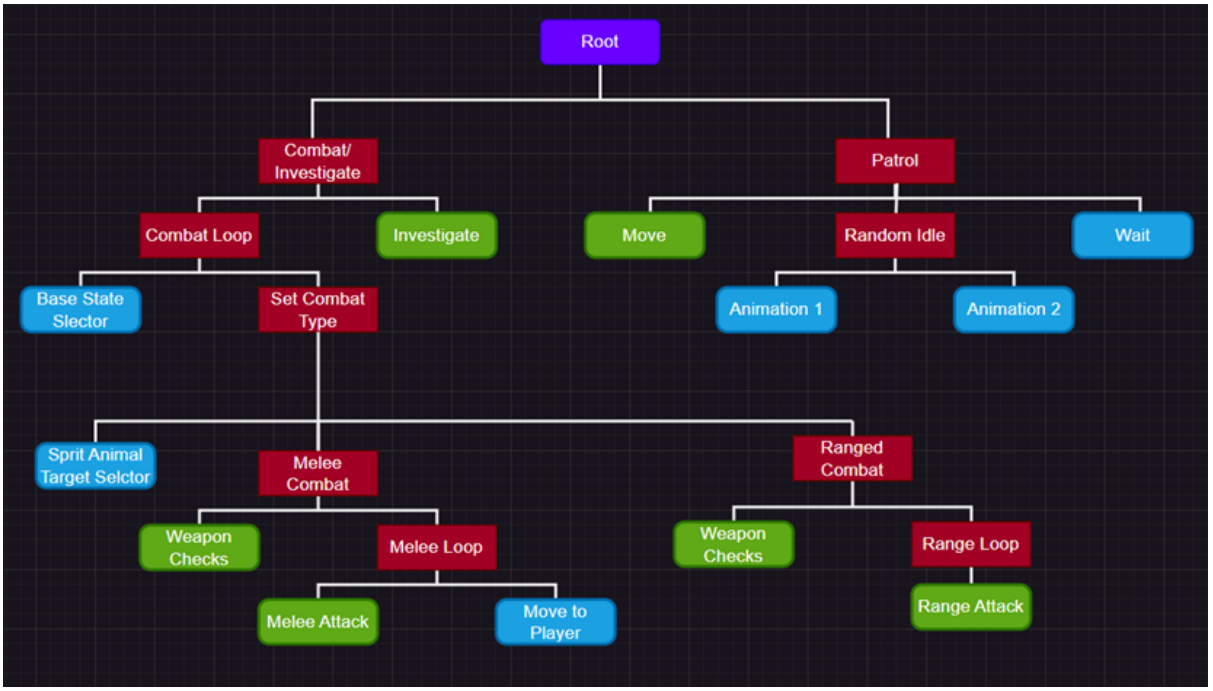
When a ritual is cast, the change in weather comes gradually, where the sky will slowly fill with more clouds and darken the sun behind them. At a weather intensity of 50 or higher, rain will slowly begin to appear and fall upon the map. Rain will also make the textures in game look wet to add to the realism.

AI Humans



The above UML diagram shows the flow of data between the 5 main AI classes and the player, it also includes the key functions and variables used in this process (removed functions and variables not involved in communication). The dotted arrows highlight which classes can update the state of the other classes, this state will then be used within the class in areas like the behaviour trees. The two main values shared between classes other than states are the health and target values, both the player and the Human AI can send targets to their spirit animal. The spirit animals need to send any damage they take back to their owner as the two share the same health.

The reason for the state changes being laid out in a somewhat strange way is to prevent the player spirit animal from attacking before combat has started. If the AI is already aggressive towards the player before the fight starts then the player needs to spot this rather than their spirit animal attacking automatically. Additionally, the AI will not attack the player's spirit animal before they see the player (this would feel annoying for the player), because of this the ‘wild’ AI and the player spirit animals don’t need any connections.

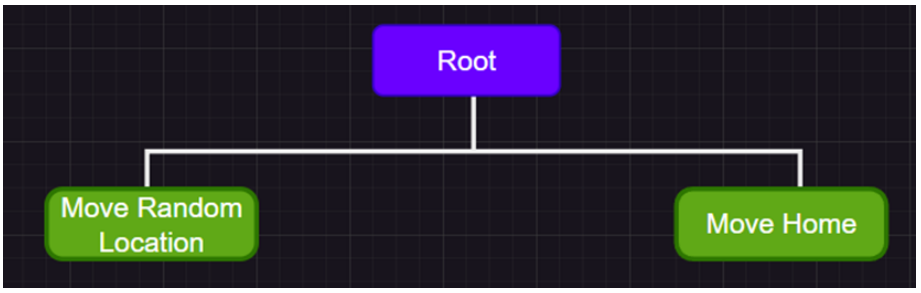


This is the Enemy Human AI behaviour tree simplified with red showing selectors (most of these have decorators attached), green sequences and blue individual tasks. The enemy human AI uses states to move around its behaviour tree, the two main states are combat and patrol, these are moved between based on line of sight to the player. It also has three sub combat states which are, melee, ranged and none, these are moved between based on the distance to the player character and will decide if the AI should stab or fire its bow at the player.

The AI can take in both sight and sound as a prescription which is used in the investigation sequence, this has multiple functions depending on if the player has been previously spotted. If the AI only hears a sound and hasn't seen the player they will go to the location of the sound to check if it was made by the player, if the player has been spotted before, the AI will go to the last place they saw the player look around that area for the player.

When the AI enters combat state it will send instructions to its spirit animal on what to attack, it takes into consideration which spirit animal the player is using when assigning a target as it can set the target to be the player or their animal (e.g. if th player and the AI both have bird it will set the target to be the player's bird).

When an enemy AI's health drops below zero it plays a death animation and is deleted from the game world, it will then spawn a skull which can be opened by the player to get loot. A custom animation blueprint is used on the enemy to change its idle and movement animation depending on if it is in the combat state or not.



The friendly AI uses a much simpler behaviour tree as all it needs to do is move around randomly and return to its home at night, the spirit animals attached to this AI are also a stripped down simple version of the normal animals, this helps improve performance. The friendly AI does not use a behaviour tree component in ue5 and instead its behaviour tree has been directly implemented in the it's blueprint. The AI has a custom animation blueprint that controls its idle and movement animation but also plays a sleeping animation when the AI is in its home at night, a waking up animation is also played when the time changes back to day.

Outside of the AI behaviour of each of the Humans there is also a script that runs on spawn inside of their blueprints that will change their appearance. The Humans will be given a random gender, hair and face which helps the gameworld feel more alive with the different appearances of the AI.

Hunting

The hunting system in the game involves several mechanics, such as the combat system as well as the inventory system, in order to function as proposed. A “huntable” animal has different reactions to other enemies in the game, mostly due to the fact that they are deemed harder to engage with in combat due to the nature of their reactions to the player. When an animal is, for example, shot by an arrow, it will change its current state to run away, rather than engaging with the enemy. The animals have 4 different states, roaming, eating, resting, and running away. These states are triggered by different events, such as eating when hunger is low or resting when energy is low.

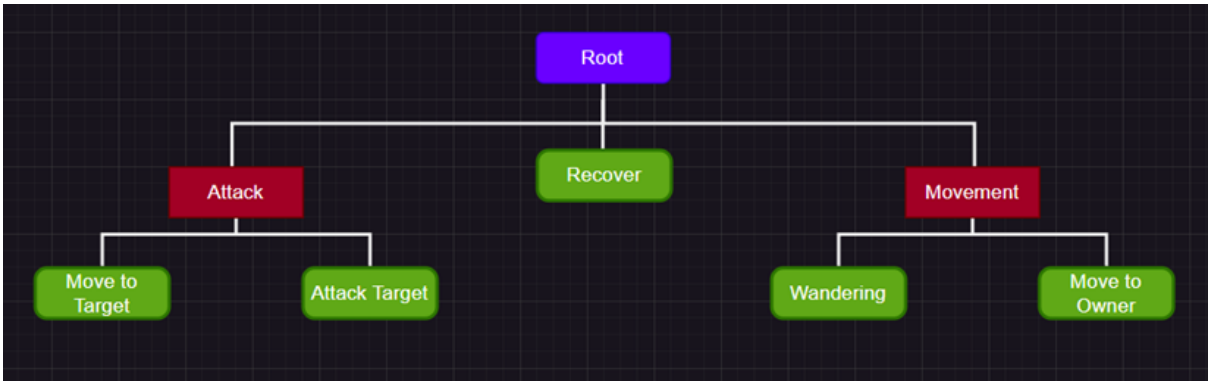
When an animal is killed, its body becomes lootable. Each animal and enemy will have its own inventory system, in which they contain materials, food items (such as meat from the bison), or arrows(if carrying or if shot by one). The looting system works the same as taking items from a chest, where the player can drag the items from one inventory to their own. Huntable animals will remain present in the game until its body has been looted, and its despawn timer has run out.

The spawning of each huntable animal in the game is controlled by their own unique “homes”. The animal's home is set as their spawn point as well as the safety area in which they will roam towards when looking to eat, sleep or run away to. Each home is contained as its own blueprint that can be placed anywhere on the visible map, when placed, the home is given a max population where the animals will continue to spawn until that limit has been reached. The animals that are spawned from the home are given a set of variables that define their individual stats, such as health, energy and hunger. This makes each animal unique to each other and allows for a more realistic feel with the hunting mechanic as some animals will be harder to hunt than others. The animals are also given a home location to roam around in, making sure that they don't stray too far away from safety and remain in the vicinity. With the deer animals, there are both stags and does. From an individual deer home, one stag will spawn and the rest of the population will be does, the stag has a much larger range from home in which it is allowed to roam, whereas the does are given the stags position and a short range of which to roam around in, mimicking the natural acts of a pack of deers where they will follow the stag and stick together.

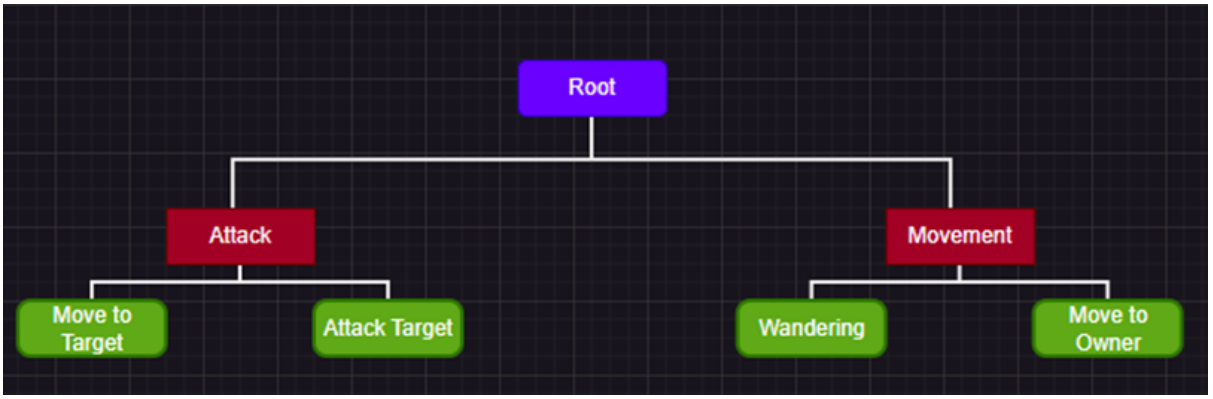
Spirit Animals

There are two spirit animals implemented in the game, a wolf and an eagle, there are three different variations of each, one for the player, one for the enemy AI, and a stripped down version for the friendly AI. All the spirit animals follow their owner around and will attack a selected target, they also share health with their owner meaning if they take damage this value is sent to the animal's owner (this can be seen in the UML in the AI section with the wolf replacing the bear).

The behaviour tree for the spirit animals are controlled using state, both the bird and the wolf have the follow and attack states, with the bird also having a recover state. The default state is "follow" with attack being triggered if a valid target is set, the bird's recover state is triggered after each attack with the state being changed back to attack after the recovery is finished.

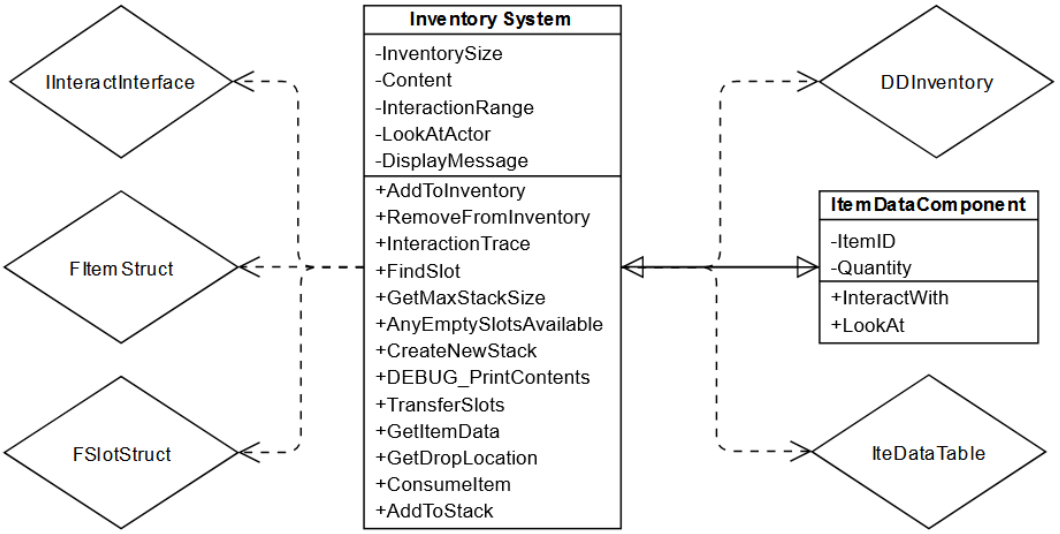


Bird Behaviour Tree



Wolf Behaviour Tree

Inventory System

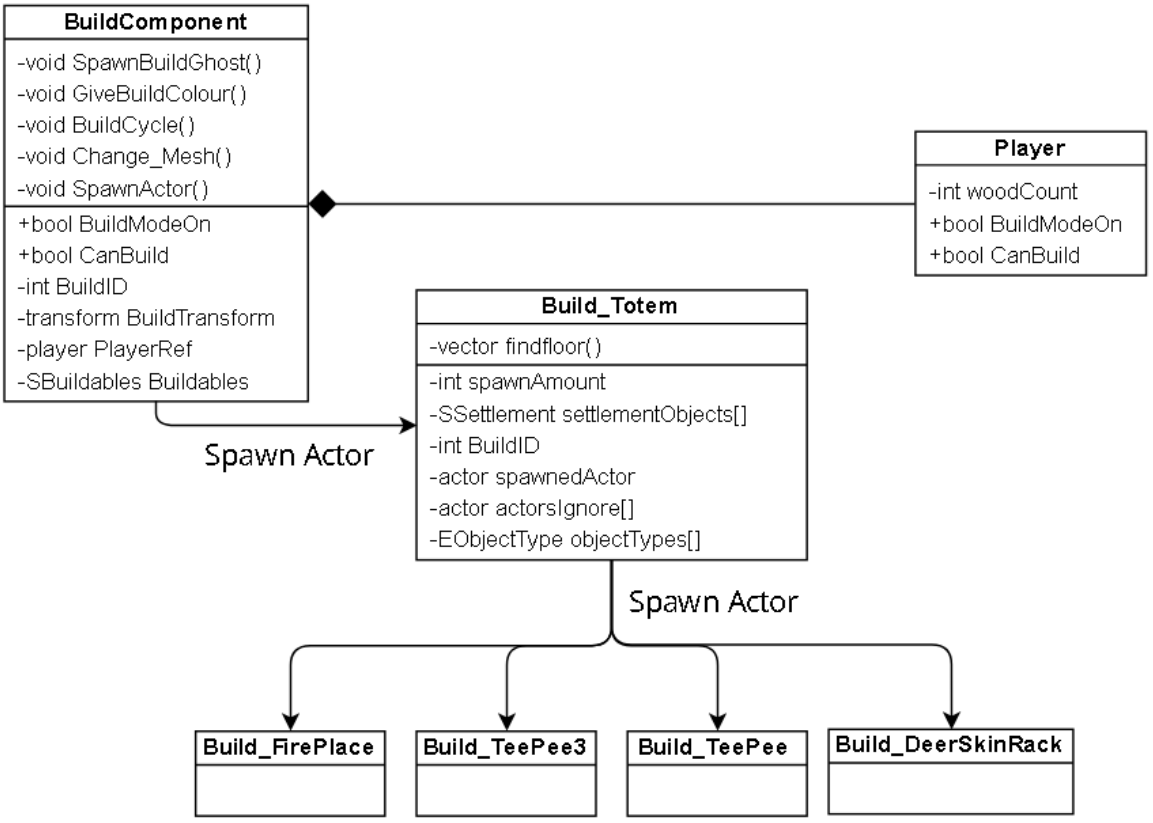


The inventory works by using a drag and drop system, in which the player can open their inventory, and freely move any item they are carrying to any slot in their inventory. This works the same way for chests and animal looting. When the player interacts with either a chest or a lootable animal, the players inventory will be displayed on screen to the left, and the inventory of the chest, animal or enemy will be displayed to the right of the screen, allowing for the user to compare and drag over the items from one inventory to the other.

Items are created using the ItemDataTable, which is organised by the FItemStruct. An Item can have: a Name, Description, Thumbnail (for displaying in the inventory), an Item Class, Stack Size, and Item Effect. The item class allows for the item to be spawned as a 3D object and dropped from the inventory into the world map. A dropped item can be picked up by the player as well. The item effect is used to differentiate the different uses of each item, for example, the item effect of an apple is to heal the player, so it is given the “Heal_Player” effect.

The interact interface allows for the player to interact with objects when looking at them, using raycasting to work out what the user is aiming at. The “Look At” interface will let the player know what it can interact with by displaying a message to the screen. For example, when looking at a chest, a box in the bottom area of the screen will say “Open Chest”.

Settlements



The settlement building works by attaching a build component to the player. “On event begin play” a reference to the player is set and the build data table is added into an array. When the player presses the build button, build mode is activated and a build ghost is spawned in front of the player which the player can move with the camera. The build ghost is a projection of the object that the player can build, the projection is green if the object can be placed and red if it cannot. This is achieved by adding a static mesh component to the player with a relative build transform of the impact point from a ray cast from the player's camera. The colours are changed by detecting collisions and changing the colour based on whether there was a collision or not. When the player presses the left mouse button whilst in build mode and is able to build then the object is spawned at the location of the build ghost, then build mode is turned off. Currently, the only buildable object is the totem, however, the settlement system’s design allows for easy addition of objects just by creating a new row in the build data table. Each object has a build ID, this variable is used within the build component to determine which object should be spawned.

Once the totem is placed down a settlement is built meaning various actors are spawned at a random point within a circular radius around the totem and a chime is played for each spawn. Within the totem’s blueprint “on event begin play” the settlements objects data table, which contains a mesh, actor and trace channel for each settlement object is fed into an array. This array is then used in a for each loop which runs through every settlement object within the data table. Within the loop body sphere overlap actors node is used on the current object to add a sphere collider around it and return an array of actors that it has overlapped

with. The node is set to only interact with the other actors that will spawn in the settlement. If the length of the array is zero then the actor is spawned at a random point within a circle around the totem. This is achieved by getting a random float between -180 and 180 (where around the totem to spawn), converting it to a rotation X vector and multiplying it by a random float in range from 500 to 2500 (how far from the totem to spawn). This is added to the location of the totem. This is the spawn transform. The build ID is then increased and the spawned actor is added the actors ignore array. The spawned actor is set to face the totem in the middle by looking at rotation between the spawned actor's location and the totem's location and then setting the actor's location using this value. To ensure that the objects that spawn around the totem are always on the floor a function is created. Within this function, a downward ray cast is performed out of the base of each object. The location of the ray cast hit is returned. This function is called after the actor's rotation is set and the return value is used to set the actor's location.

UI

The UI works through various blueprints. Widget blueprints are used to create the visual section of the UI. Very few actor blueprints are used and the level blueprint is used for the main menu.

The main menu is a level blueprint and when you are navigating through the menu it uses create widget nodes and removes the parent node (The previous menu), instead of generating a new level each time. When you load the game it creates the main level using an open level node.

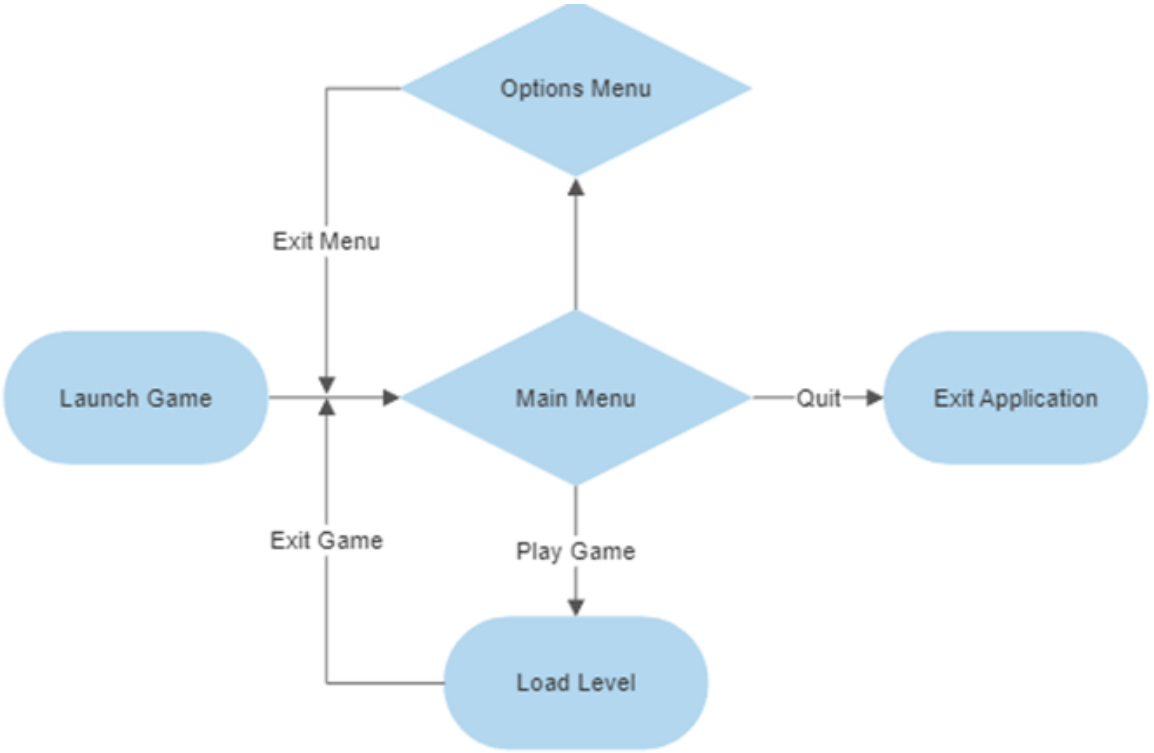
The settings menu allows the player to change the framerate, quality and audio settings for the game. To change the resolution of the game a function is used to see what the player has selected and put it into effect. To set the framerate multiple variables are used within a function to call upon what the player has chosen in the menu and implement it. The same is done with the screen mode being windowed or Fullscreen. Audio settings are implemented using 3 different functions and various variables. Upon changing the value of the sliders and pressing save, it calls upon the save function to save the slider’s position so that when you exit the menu and go back on to it the values are the same. The next function is the load function which checks to see if the save game slot is defined and calls upon it to load into the viewport. The final function is used to update the widget, so the position in which you left the slider last time appears on screen. Sounds are defined into different sound classes and sound mixes so that the value of the sliders determines the volume of them.

The in game HUD uses the player blueprint to create both the objective and compass viewports. The objective blueprint uses a function and multiple variables to grab the text in the blueprint and show it on the screen. The string is defined when the objective blueprint is placed within the map. A collision box is used to trigger the blueprint updating the objective.

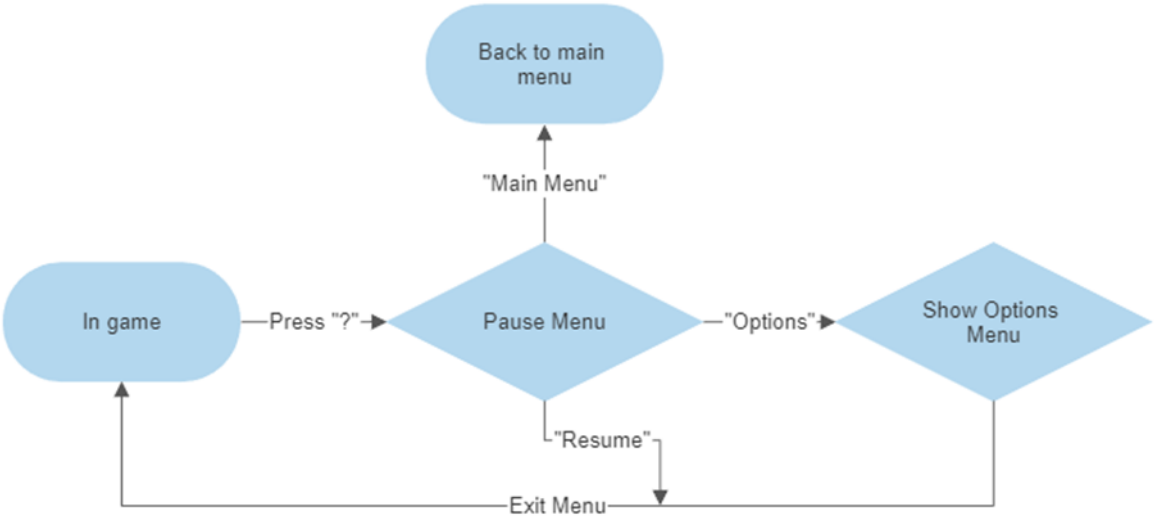
The compass uses variables to define the world rotation and reflect it on the compass widget.

The pause menu attaches to the player blueprint. This doesn't use any variables, just a flip flop which determines if the pause widget blueprint is active. If it is, the game state is set to paused and you can navigate the menu, going into the settings as well.

Main menu flowchart:



Pause Menu Flowchart:



References

Adobe Inc. 1987. *Photoshop* [computer program] Adobe

Alludo. 2015. *Gravit Designer* [computer program]

Artofficial Entertainment. 2023.*Unreal Engine 5 AI Tutorial - Finalizing Combat: Adding Ranged and Melee* [online] youtube.com. Available from: https://www.youtube.com/watch?v=VA3GYHKYiWE [Accessed 18 May 2023]

Artofficial Entertainment. 2023. *Unreal Engine 5 AI Tutorial - Setting Up Combat* [online] youtube.com. Available from: https://www.youtube.com/watch?v=1JButQ7hwXo [Accessed 18 May 2023]

Aspland.M.2021. *How To Create An Animal AI In Unreal Engine 4* [online] youtube.com. Available from: https://www.youtube.com/playlist?list=PLQN3U_-lMANNZtjAcKP8-VA7I8n13l1Cm [Accessed 18 May 2023]

Aspland.M.2021. *How To Play Audio Between Levels* [online] youtube.com. Available from: https://www.youtube.com/watch?v=M5RnDBNnjx4 [Accessed 18 May 2023]

Blender Foundation. 1994. *Blender* 3.5.1. [computer program] Blender.org

CodeViper. 2017. *Save/Load Menu Settings - Unreal Engine 4 Tutorial* [online] youtube.com. Available from: https://www.youtube.com/watch?v=rbEESQUWgfM [Accessed 18 May 2023]

Epic Games. 2021. *MetaHuman.* [computer program] Epic Games

Epic Games. 2014. *Unreal Engine Marketplace* [online] unrealengine.com. Available from:https://www.unrealengine.com/marketplace/en-US/store [Accessed 20 Feb 2023]

Epic Games. 1998. *Unreal Engine 5.* [computer program] Epic Games

GIMP Development Team. 1998. *GIMP* 2.10.34. [computer program]

It's Me Bro. 2022. *UE4 Base Building / Lets Get Started* [online] youtube.com. Available from: https://www.youtube.com/watch?v=Z8u1EUJxfOI&list=PLnz5Pff5C3gfiT0divCpxfHdpNSEEC177&index=3 [Accessed 18 May 2023]

KDE,1998. *Krita* 5.1.5 [computer program] Deventer: KDE

Layley.R.2023.*How to Make an Inventory System in Unreal Engine 5* [online] youtube.com. Available from: https://www.youtube.com/playlist?list=PL4G2bSPE_8umjCYXbq0v5IoV-Wi_WAxC3 [Accessed 18 May 2023]

Quadspinner. 2008. *Gaea* [computer program] Quadspinner

The Audacity Team. 2000. *Audacity* 3.3.2 [computer program]