**Computer Games Development CW208**

**Year IV**

**Technical Design Doc**

Shane Conroy

c00270031

11/05/2025

**Contents**

Brief.....................................................................................................3, 4

Sheep......................................................................5, 6, 7, 8, 9, 10, 11, 12

World..............................................................13, 14, 15, 16, 17, 18, 19, 20

Wolf...............................................................................21, 22, 23, 24, 25

Dog.....................................................................................25, 26, 27, 28

Grass.........................................................................................29, 30, 31

Economy................................................................31, 32, 33, 34, 35, 36

Combiner............................................................................37, 38, 39, 40

HUD....................................................................................41, 42, 43, 44

Particles..........................................................45, 44, 45, 46, 47, 48, 49

**Brief**

-Overview-

This is the Technical Design Doc (TDD) for my game “Save our Sheep!” A 2D, farm management game built with SFML and coded in c++. I will outline each class's purpose, its responsibilities, behaviours and how it interacts with the game’s core mechanics. This is intended to be a blueprint if someone else were to continue developing this game.

The game includes an Ai behaviour system, basic economy, and a Ui system with a particle system attached.

-Legend-

Purpose – Quick overview of what the class is

Responsibilities – What the class oversees

Data members – The important variables of the class

Methods – The class’s functions

Interactions – What other classes this class ‘talks’ to.

State Managment – Any internal states a class has

Ai logic – If applicable, the entities behaviour

Rendering – How it’s drawn

Scalability – Potential future additions

-Scope-

This document will cover the internal logic of:

Entity classes

* Sheep, Dog, Wolf and Grass.

Core classes

* World

Utility classes

* Combiner, GrassUtility, ParticleManager, Economy

Additional classes

* HUD, Fence, Menu

-Dependencies-

Language: c++

Graphics library: SFML

**Sheep**

-Purpose-

The Sheep class is the game’s main entity and the player’s only way of earning cash. The sheep are autonomous and interact with the world with certain behaviours based off in game conditions. The sheep will eat grass to grow their wool, which the player interacts with by sheering them. Sheep are always different with hidden stats that influence how they act.

-Responsibilities-

* The Sheep class handles:
* Creating Sheep
* The sheep’s behaviours (wander, eating, seek, returning to pen, etc.)
* Checking collisions with grass and triggering the eating behaviour
* Tracking wool amount, and growing wool after eating.
* Handling genetic stats inheritance
* Reacting to the Dog and Wolf

-Data members-

**int amountEaten** – This tracks how much the sheep has eaten and is used when changing the sheep’s wool colour

**sf::Vector2f lastEatenGrass** – Gets the position of the sheep’s last eaten grass, making sure when they pick a new grass to eat, they don't chose the same one

**float whistleDelay** – Timer so the whistle can’t be spammed

**bool isArrived** – Used to check if all sheep have arrived at the Combiner

**int greatness** –Used as ‘fitness’ for the Combiner function

**int prestige –** Increased when a sheep comes out of the Combiner

**float awarness –** Variable used so sheep react to the whistle at different times

**float fear** – The distance before sheep run away from Wolf

**int woolBonus** – Bonus cash awarded on top of the wool sell price

**float herdingTimer** – Sheep temporarily freeze when the Dog is herding them. Making herding easier.

**std::vector<sf::Vector2f> availibleGrassNodes** – Holds the location of all grass not being currently eaten

**sf::Vector2f closestPos** – The closest grass node

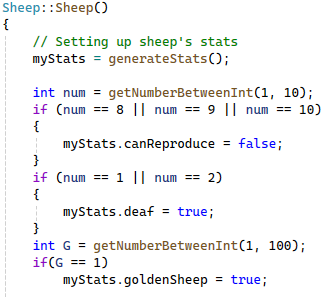
**sf::Vector2f exitTarget/Leader** – Position sheep go to when leaving the pen

**sf::Vector2f penTarget –** Position used for the whistle function when returning

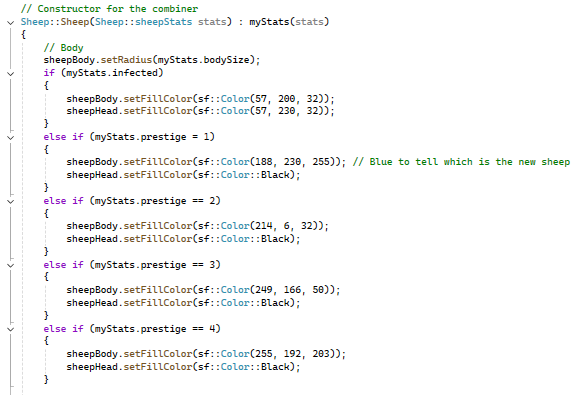
**sf::Vector2f randomPenPos** - “

-Methods-

**Sheep()** - When spawning sheep will be given random stats (generateStats(), see below) but when they are created will have a chance to be either deaf or unable to reproduce. This is decided by a simple global function call and ‘if’ statement

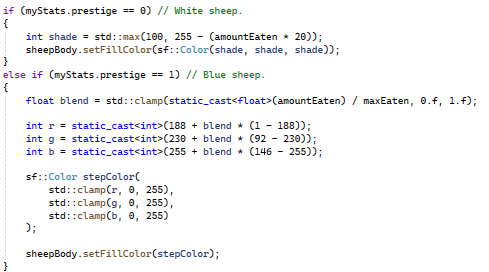


**Sheep(Sheep::sheepStats stats) : myStats(stats)** - This is the 2nd sheep constructor used for the when the combined sheep is being spawned. Pretty much the same apart from the prestige colours and the infected sheep check, as infected sheep can only be spawned from the Combiner.

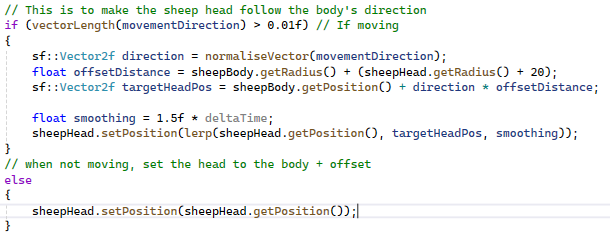


**Update()** - Called every frame to manage sheep’s logic. Handles all behaviors: eating, reacting, following the flock, or leading it. Also handles infected’s TTL. Adjusts color/size based on wool growth. Uses flocking logic (alignment, cohesion, separation), dog avoidance, and recall movement when moving sheep.

Wool growth – Inside Update when a sheep eats grass, they grow and change to a shade darker. This is to visually tell the player how much wool the sheep has. I do this by getting a gradient between two colours the ‘start’ colour and ‘end’ colour, I then use std::clamp to get each “step” between the shades rather than having hard coded stages of wool colour.



Basically it does the (1 – 188) first which is the end colour against the start colour. Then it’ll multiply that by the blend factor, which is gotten above and plus the 188 to get the final result. The additional std::clamp just makes sure the values stay between 0 and 255.



This function is used for the sheep’s head. It get the sheep current direction vector, adds some distance to it and places itself on that position, so the head is always ‘leading’ the body. I added the lerp function as there was a problem with the sheep making micro-adjustments while in a flock. The sheep were snapping a lot, so the lerp reduces that.

**leaderBehaviour()** - Handles the leaders behaviour based off certain conditions. If there’s no grass wander, if there is, seek towards it. Returns a movement vector to Update.

**followerBehaviour()** - Handles the other sheep’s behaviour. Follows the leader or closest grass node, exits the pen when the leader does, and reacts to the dog if within range.

**handleRecall()** - This herds the sheep back into the pen by giving them a point on the exitGate. The sheep will go to it, once at it, pick a point inside the pen and then move to that. Once at that point they will wait there for 10 seconds to avoid coming back out right away.

**seperation(), alignment(), cohesion()** - Flocking for the sheep. Makes sure the sheep stays together as a herd but dont overlap too much.

**whistleHeard()** - When the whistle button is clicked a timer goes off for each sheep (unless deaf). After the timer, this function will call the handleRecall() for the sheep.

**awayFromDog()** - Applies a repulsion force to push sheep away from the dog within a certain radius.

**generateStats()** - Assigns the sheep’s internal stats at spawn so each sheep is unique. It does this by first checking how much buffs and debuffs a sheep gets, both from 1-3. Then choses which stat to randomise via a switch case. Each switch adds or subtracts a certain amount from the chosen stat using getNumberBetweenInt(x, y). Then returns the new stats.

-Interactions-

**World** – World owns all created Sheep instances. World calls Sheep’s Update() and Draw(). Handles state changes between Sheep and other classes, like Grass and eating grass or being eaten by Wolf.

**Dog –** Sheep knows the Dog’s position as to know how much to be repulsed by him when he comes near.

**Wolf –** Needed for checking the fear radius and when to flee. If caught, beingEaten is set to true and the sheep ‘freezes.’

**GrassUtility –** Needed so the Sheep knows about the locations of each grassNode. Sheep call " GrassUtility::FindClosestNodePosition()” to find grass.

**Combiner** – Combiner uses the Sheeps internal stats and greatness for it’s own function.

**ParticleManager** – Spawns particles when a Sheep dies or is sold.

**HUD/Economy/WoolParticle** – When sheep are sheered, they drop wool using WoolParticle, sell the wool which gives the Economy more money, and is visually shown with floating numbers from HUD.

-State Managment-

The Sheep class uses an enum called “behaviour” for it’s behaviour logic.

exiting – Initial state. Sheep will leave the pen after spawning

wandering – Random movement when no grass is found

seeking – Targeted movement toward grass node

fleeing – Triggered when a wolf is nearby

entering – Goes from field into pen

currentBehaviour – main variable for tracking and switching states, updated in Update()

eating – controlled by isEating and doneEating. Time is varied by eatSpeed

external controlled states – isEaten and eatenByWolf both changed by the Wolf class. whistleDelay controls going into the entering state. infectedTTL will kill the sheep after some time

-Ai logic-

The Sheep is controlled by two main Ai’s, state driven logic and flocking behaviour. Each sheep acts as one herd, but can split off and act independently if conditions are right, such as a follower coming close to a grass node.

Core Logic

The Sheep’s core decision making is made in their Update(). This is where currentBehaviour, sheep’s state changer variable, is mainly at play.

Leaders

Leader’s will lead the rest of the herd. If inside the pen, they will pick a point to exit from and leave the pen. If outside the pen and in the field they will either seek to the closest grass or wander if there’s no grass.

Followers

If the leader is inside the pen with them, follow the leader out, else just leave the pen. Once outside the pen, follow the leader unless they come near a grass node, in which case they will seek to it and eat the grass.

Leaders & Followers

Both react to the wolf and dog, both repulse from them.

Boid/Flocking

The sheep use boid style flocking to move in a herd movement, clumping together but not overlapping.

Infected Sheep

Infected Sheep ignore all other behaviours and only wander, like zombies.

Whistle Recall

Overwrites the sheep’s current behaviour and forces them to go back into the pen.

-Rendering-

The Sheep consists of 2 sf::circleShapes. One for the head and one for the body. The wool particles that fall off after being sheered are also smaller sf::circleShapes.

I draw the sheep’s wool differently based on how much grass the sheep has eaten, and different colours based off prestige.

All rendered in Worlds Draw().

**World**

-Purpose-

The World class is thecentral manager of the game’s logic and state. It initializes, updates, and draws all core entities (sheep, wolves, dog, grass, HUD, Econ) and handles interaction between them. It also manages the day/night cycle, economy, spawning, and input handling.

-Responsibilities-

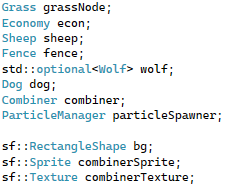
The World class has a lot of responsibilities as it’s acts like a bride for a lot of classes, and calls every class’s Draw() and Update()

Responsibilities:

* Udating and Drawing all entities
* Managing Sheep’s flocking behaviour and changing leader
* Handling almost all collisions (Sheep – Grass. Wolf – Sheep. Dog – Sheep)
* Handling Player inputs (Sheering, Button clicks)
* Updating Economy when sheering or selling sheep
* Managing UI state (pop-out menu)

-Data members-

All of the class declarations



std::optional<Wolf> wolf - This one in particular was used to fix a bug with the wolf entity. Basically my bug was the game was technically detecting a wolf was in the game and running certain behaviours such as Wolf::Hunt and killing some of my sheep without there actually being a wolf in the game. There was a ghost wolf. std::optional makes it so when the class is made, there isn’t a wolf spawned in by accident too.

**float transition** – Used in the day/night lerp function for changing the world’s colour

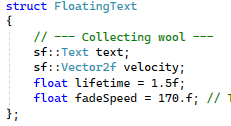
**bool showHoverText –** Used for when the player hovers over the upgrade buttons, displays the price

**std::vector<Sheep> sheepArray** – Holds a copy of all sheep in the world, the main array for sheep logic

**std::vector<Grass> grassNodeArray** – Holds grass objects, gets passed to Sheep through GrassUtility

**std::vector<Sheep\*> herd** - Holds a pointer to the real sheep object in an array, used by Wolf when he’s hunting

**FloatingText -** Struct for text, used by the cash fade when picking up wool and when hovering over upgrades



-Methods-

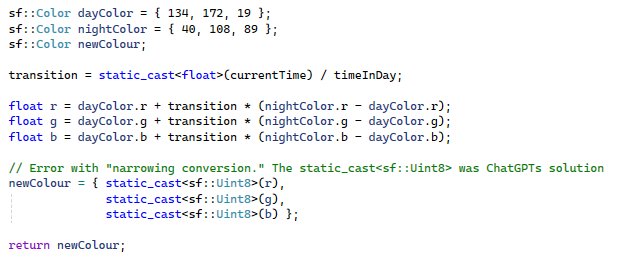
**SpawnGrassNodes()** - This function fills an array full of Grass objects each time one is bought

**UpdateGrassNodes()** -This function checks if a sheep is near a grassNode and flags the sheep as eating and the grassNode as taken. The function then updates the availible grass nodes array, to make sure sheep are only going to grass nodes that no one else is at.

**PassGrassToSheep()** - Takes all grass nodes that aren’t being eaten and passes an array to Sheep. The Sheep will then pass the array to GrassUtility

**spawnCashOrb(const WoolParticle& orb) -** When a wolf eats an infected Sheep, this is how he spawns a cash orb. It reuses the same system as the wool particles dropping.

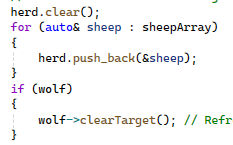
**DaylightCycle() -** This function controls the colour of the background, indicating the time of day. This uses the exact same logic as my sheep colour changing.



It’s the same maths as before, although one alteration I made in the Sheep wool code, I didnt figure out here (I wrote the Sheep’s code second) I had an error with assigning the r, g, b to newColour. I was getting a narrowing conversion error, but I hadn’t a clue what that meant. I put the error into ChatGPT and it gave me the solution of using static\_cast<sf::Uint8>(). I realise now the error was the answer was out of the 0-255 bounds for colour. Which I fixed with std::clamp in Sheep.

**updateFencedGrass() -** Uses newColour to do the same logic as above, but for the pen’s grass. The pen’s grass is a different sf::Rect so I could easily use .contains() to see what sheep were inside the pen.

**PopulateWorldWithSheep()** - Similar to UpdateGrassNodes(), when a Sheep is bought, checks if the player’s sheep amount is less than the current sheep cap. If the player has space, pushes it into a sheep array, else the player gets refunded. Handles selling sheep too, popping the sheep out of the array and erasing him.



Also refreshes the Wolf’s array to avoid strange bugs.

**up\_SheepMax(), up\_WoolSell(), up\_SheepAmount(), up\_GrassAmount() -** Functions that are called when an upgrade is bought. Ups a variable like how much wool sells for or how much sheep are bought at one time.

**shearsFunc(sf::Vector2i mousePos)** - When clicked the player will equip sheers and can sheer sheep. Spawns particles of wool when a sheep gets sheered and returns the sheep back his lightest colour, his amountEaten back to 0 and his body size back to normal

**WorldTime()** - int counts up then down, returns day or not

**woolCollectFunc(sf::Vector2i mousePos, int value)** - When wool is sheered, sucks the wool to the mouse then when collected, displays the cash earned with floating text.

**combineFunc(std::vector< Sheep>& sheepArray)** - Counts all reproductive sheep, if over 30, sucks all of them into the Combiner, when the sheep are close, flags a bool, when all are flagged, sorts by greatest, then calls the Combiner function. Afterwards erases the 30 sheep used up in the combining.

**displayCosts(sf::Vector2i mousePos)** - Displays how much an upgrade costs when hovering over one

**Update(float deltaTime, sf::Vector2i mousePos)**

Sheep

* Always checks if there’s an active leader, if not, promote someone else. Helps when leader is eating or being eaten
* Updates all Sheep outside the pen
* Handles the Whistle and Stuck button
* Increments greatness based off different criterias

Wolf

* If it’s night and there’s no wolves, spawn one. If it’s day, despawn him
* If the dog barks, freeze the wolf
* Calls Hunt() on the Wolf, passing in the pointer array from earlier.

Sheep Cleanup

* Loops through all sheep and checks if any were eaten or if any infected sheep expired. Spawns the appropriate particles.

Dog

* Handles barking and stun logic
* Lets Dog follow the mouse

Player Inputs

* SheersFunc
* Opening the fence
* All buying and upgrading buttons

Grass

* Buying and despawning of grass nodes

Particles

* Sheep, infected sheep, and Wolf dying
* Floating text when collecting wool

Other

* Tracks sheepHeartCooldown for UI interaction
* If combining is triggered, runs combineFunc() using sheepArray
* Rebuilds the herd vector with up-to-date sheep pointers
* If the wolf is marked as dead, spawns a death particles on him

- Interactions-

**Sheep** – Creates and updates all sheep. Handles their recall and collisions.

**Dog** – Mouse input to move dog. Passes dog position to sheep for herding behavior.

**Wolf** – Controls when wolves spawn and who they hunt. Kills sheep when sheep eaten. Handles stunning by dog.

**Economy** – Adds or deducts funds based on shearing, combining, selling, and upgrades.

**GrassUtility** – Places grass in the world and checks if it's eaten by sheep.

**HUD** – Reads button clicks and triggers upgrade functions.

**Particles** – Spawns visual effects for shearing, wolf death, sheep death, and wool collection.

-State Managment-

**Night or Day** – The only real state is if it’s either night or day. During the day the player can sheer sheep and be jolly. During the night wolves will spawn and the player can no longer sheer the sheep.

-Ai logic-

N/A

-Rendering-

The world is rendered from the one, top-down camera angle

The world renders everything else in the game

The background is the only interesting thing the world renders, dynamically changing to visually tell the player what time it is rather than having a UI element

**Wolf**

-Purpose-

The purpose of the Wolf is to add a threat to the game that the player has to deal with. The Wolf will spawn during the night, seek the closest Sheep and eat it. After eating it, he will move onto the next Sheep he sees.

-Responsibilities-

Finding the nearest Sheep

Eating the sheep

Telling World ‘im dead’ in order to drop a cash orb

Becoming stunned when the Dog barks near him

Dying after eating an infected sheep

-Data Members-

**sf::CircleShape wolfBody, sf::CircleShape wolfHead, sf::ConvexShape wolfTail, wolfEars, wolfSnout** – All used for the Wolf’s body parts. I kept the simple shapes by using sf::ConvexShape, maintaining the simple, cute art style.

**Sheep\* targetSheep** – Pointer to the current sheep being hunted

**bool isEating** – Whether the wolf is currently eating

**bool isWolfDead** – True if the wolf consumed an infected sheep and should be removed

-Methods-

**FindClosestSheep(std::vector<Sheep\*>& flock, sf::RectangleShape innerGrass)** - Goes through the entire sheep array and check which one is closest. Returns the closest, which is used in Hunt()

**Hunt(std::vector<Sheep\*>& flock, float deltaTime, sf::RectangleShape innerGrass, World\* world)** - Handles if the wolf is stunned or not. If not, continues the main logic. Checks if I have a target, if I dont calls FindClosest. Once the wolf has a target, moves towards it, and once the sheep is in range, the sheep becomes eaten and the Wolf stays still while he eats.

**spawnWolf(int spawnPos)** - Takes in an int, which spawns him at one of the edge of the screen, excluding below.

**setStunned(float time) { stunTimer = time; }** - Passes an int in and stuns the wolf for that long

-Interactions-

**World** – Spawns the Wolf and calls its Hunt() and Draw() methods. Also despawns it in the morning.

**Sheep** – Targets and eats sheep. Triggers internal state inside the sheep, like beingEaten and eatenByWolf.

**Dog** – If the dog barks nearby, the wolf becomes stunned via setStunned() in Wolf.h.

**WoolParticle** – When the wolf dies from eating an infected sheep, spawns a unique pink cash orb using spawnCashOrb().

-State Managment-

targetSheep – gets set when the Wolf calls FindClosestSheep(). Used to begin hunting state

isEating – Active when the wolf is close enough to a sheep and eatTimer is counting down, makes the Wolf stay still

stunTimer – Freezes the Wolf during duration

isWolfDead – Tells World to erase the wolf and spawn a cash orb

-Ai logic-

The wolf’s Ai is quite simple, using a basic greedy algorithm to find the closest Sheep. Then uses basic movement to seek towards it, using a direct vector and constant speed.

-Rendering-

The Wolf’s body is composed of many different shapes, the most out of any entity in the game.

The Wolf doesn’t have any interesting render quirks like the previous two classes.

Drawn by World’s Draw()

**Dog**

-Purpose-

The purpose of the Dog is to have some sort of defense against the wolves attacking. The Dog can herd the flock and stun the Wolf. It’s also kinda fun to have the Dog run around while you’re idle.

-Responsibilities-

The Dog class handles:

* Following the player’s mouse cursor
* Triggering a bark to stun wolves
* Generating the bark shockwaves for visual feedback
* Allowing the player to toggle sitting mode using the middle mouse button as to not accidentally herd sheep while sheering

-Data members-

**std::vector<BarkShockwave> shockwave** – used for creating the Dog’s barks

**bool middleMousePressedLastFrame** – there was a bug with the dog sitting down and then up immediately, this is my solution. It flags a bool after the sit call so it can’t be repeated immediately after

Everything else should be self explanatory.

-Methods-

**follow(sf::Vector2i mousePos)** - gets the mouse’s position and move towards it. Every body part’s movements are done in here too, as well as sitting.

**Bark() -** creates a shockwave on right mouse click

**UpdateBark(float deltaTime)** - Expands the shockwave, decreases it’s alpha and when the the alpha is 0 or the lifetime is over, erases it.

-Interactions-

**World** – Calls Update() and Draw(). Reads barkTriggered to check if Wolf was in range.

**Wolf** – If Wolf is within range of the dog during a bark, the wolf gets stunned.

**Sheep** – Sheep check the dog’s position and apply a repulsion force if nearby.

-State Management-

The Dog can either follow the mouse or be sitting

-Ai Logic-

N/A

Player controlled

-Rendering-

The bark’s shockwave slowly fades out while expanding to give the impression of a shout that’s losing volume over time

Each part of the dog is drawn seperately similar to the Wolf

World also draw the Dog

**Grass**

-Purpose-

The purpose of the Grass class is for the sheep to have a main food source. They get created by World and erased when a Sheep eats them. They let Sheep grow out their wool

-Responsibilities-

The Grass class handles:

Spawning at positions within the field

Flagging taken or eaten

Erasing once fully eaten

-Data Members-

**bool taken** – Flagged when a Sheep begins eating the grass to avoid more than one Sheep at the same node

**bool eaten** – Marks the grass node for deletion after the Sheep has eaten it

-Methods-

N/A

Grass node has no functions that aren’t getters or setters

-Interactions-

**World** - Spawns, updates, and removes grass objects. Maintains the availible grass array

**Grass Utility** – Acts as a middle man so Sheep can find grass nodes without directly knowing about the Grass class

-Ai logic-

N/A

-Rendering-

World draw each grass node

No special effects or gradients. I didnt want the grass node’s to fade similar to the background because I didnt want the grass to blend in or be hard to see.

**Economy**

-Purpose-

Economy manages everything related to the Player’s funds. It handles current funds, purchasing, upgrades, and interaction with the UI. It handles everything related to buying sheep and grass, applying upgrades, enabling shearing, and issuing passive income. It also integrates with the HUD to show visual feedback for button clicking

-Responsibilities-

* Player current funds
* Buying and selling sheep
* Buying grass nodes
* Upgrades and level cap checks
* Communicates to HUD what buttons were pressed for visual feedback
* Used to manage passive income but I took it out in place of selling wool
* Triggering Sheerin or Combining, also handles the “unstuck” button

-Data members-

**int passiveIncome** – This used to be calculated by how much Sheep the Player had, but I got rid of it because the Player would earn a LOT of money quickly. I replaced passive income with the current wool selling mechanic.

**bool sheepPurchased –** Bool used for flagging World to spawn or despawn a Sheep

**bool sheepSold -** “

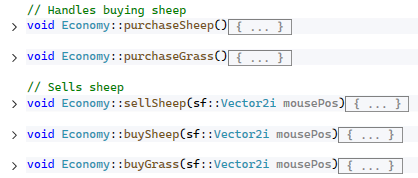
**std::map<std::string, std::map<float, float>> upgradeMap -** This is an std::map that maps a string to another map. I use this for the upgrades and their prices. The outermost map holds the name of the upgrade and the second map. The second map holds the upgrades current price and it’s scaling agent.

**bool whistle, combine, stuck, sheersOn** – Flags for special actions

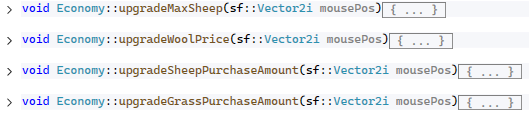
-Methods-

**addFunds(Funds\_Enum fundType)** - Takes in an enum class and based off that, updates the players current funds. Check if player funds doesn’t exceed the limit

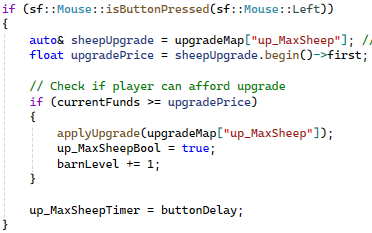
**addFunds(int amount) -** Updates the players current funds, but takes in an int. I use this for an edge case with the sheep’s wool

 These all behave similarly, they have a cooldown when the button is clicked, and while it’s on cooldown the button is visually turned more ’grey’ to indicate it has been clicked. The functionality of all of them is simply checking if there was a mouse and button collision, then if there was flag a bool. World usually take over from then.

**popOutPanelFunc(sf::Vector2i mousePos) -** Checks if Tab is pressed, and if it is, it flags a bool for the pop-out panel to be drawn further on the screen.

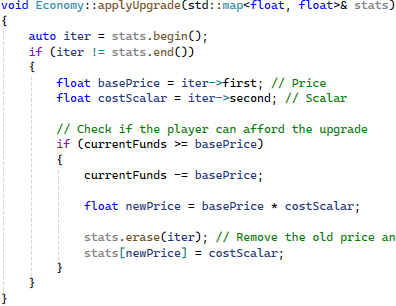


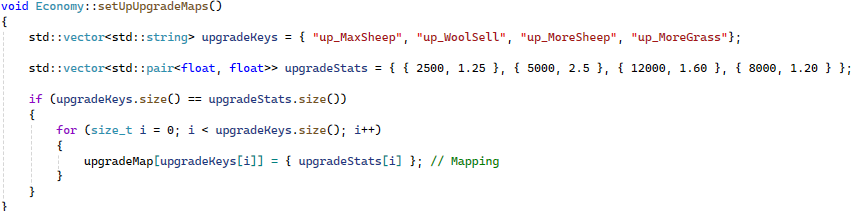
These four are similar to each other as well. They work by checking if there was a button click on top of their respective button. If there was, it’ll do the internal logic.



Basically what this does is it calls the std::map upgardeMap and asks for the values from “up\_MaxSheep,” which will give the current price and scaling factor. Then it calls applyUpgrade with that string

**applyUpgrade(std::map<float, float>& stats)** - In charge of the upgrades scaling factor and applying the specific upgrades that gets passed in

 When this function is called and an upgrade is passed in, it’ll look at the first entry which is the upgrades current price, and then it’s second entry which is the scaling factor. This will deduct the money from the player and then multiply the old price by the scalar. Then swap the old price for the new one.

**setUpUpgradeMaps()** - This is responsible for mapping the string to the std::pair of the current price and scalar. It’s called at the start by World’s constructor. It works by assigning pairs to values in upgradeStats. Then it places the linked pair into upgradeMap, by saying upgradeKey[x] links to this std::pair.

**update(sf::Vector2i mousePos)** - Handles the sheers following the cursor and used to handle passive income

-Interactions-

**World** – Economy flags World so it knows when to spawn sheep, recall sheep, combine, etc.

**HUD** – All button states and visual feedback are controlled via Economy

**Sheep** – On shearing, addFunds() is called with a woolSell type, using sheep.myStats.woolBonus

-State Managment-

The Economy kinda has two states. One being when the sheers are in the player’s hands, and the other being when the pop out menu is popped out

-Rendering-

Economy doesn’t draw anything itself, it uses HUD to draw all buttons and visuals.

The Economy will only have a bit of the Pop-out menu drawn at first until Tab is pressed, then it flags to draw the whole thing. This is so the screen is always super cramped.

**Combiner**

-Purpose-

The purpose of the Combiner is to have a sort of ‘reset’ that’s seen in many idle games. It’s a way of restarting the gameloop, but rewarding the player as well. Most other games do this by giving bonuses to income or somehting similar. In my game you are rewarded with an improved, prestiged sheep with buffed stats

-Responsibilities-

The Combiner class is responsible for:

* Sorting and selecting the top 4 sheep based off the “greatness” stat
* Merging the top 4 sheep into one buffed sheep
* Assigning prestige level
* Small chance to introduce an infected zombie

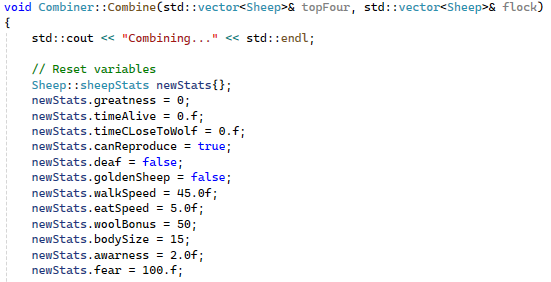
-Data members-

N/A

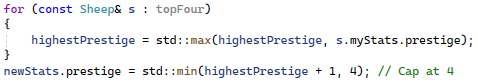
The Combiner class doesn’t actually hold anything. It simply looks at the Sheep’s stats, combines them and that’s all. Nothing is stored.

-Methods-

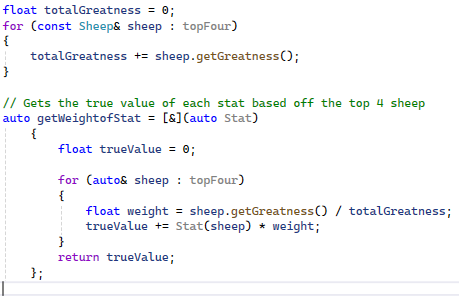
**Combine(std::vector<Sheep>& topFour, std::vector<Sheep>& flock)** - This is the Combiner’s main purpose and only function.



At the start, the Combiner will completely wipe the stats, resseting all of them into a new variable ”newStats,” which will eventually be given to the new Sheep.



Then it looks through the top 4 sheep to find the highest prestige. The new made Sheep will always be +1 above the highest prestige, capped at 4.



Then for calculating the Sheep new stats, I didnt want to just have a boring random function that picks from the top 4’s stats and assigns them. I wanted to have somehting more interesting and a function that would actually ‘combine’ the sheep into one. My solution was to find the “true value” or the weighted avergae of all the sheeps stats. The higher a sheep’s greatness, the higher his influence.

In the for loop, I take a four sheep multiply the stat by a weight, which is the sheep’s greatness against the total greatness. This is where having higher greatness effects the result. This will give us a percentage average of how much the sheep should affect the final say

If a Sheep has 80/100 greatness with a walkSpeed of 40.f the result would be 40.f \* 0.8 = 32

And another Sheep has 20/100 greatness with a walkSpeed of 100.f it’d be 100.f \* 0.2 = 20

The result would be skewed far greater in the first sheeps favour and the “true value” would be 52.f

-Interactions-

**World** – Calls Combine() when the player triggers the combine button and the flock meets the requirements

**Economy** – Triggers combine via the Combine button

**Sheep** – Passes in top-performing sheep and receives a new combined sheep

-State Managment-

N/A

-Ai logic-

N/A

-Rendering-

N/A

**HUD**

-Purpose-

The HUD class manages all on-screen user interface elements, including buttons, upgrades, icons, and money display. It doesn’t contain any logic itself, but provides all inputs and visual feedback. Tightly knit with Economy.

-Responsibilities-

Drawing buttons

Updating player current funds

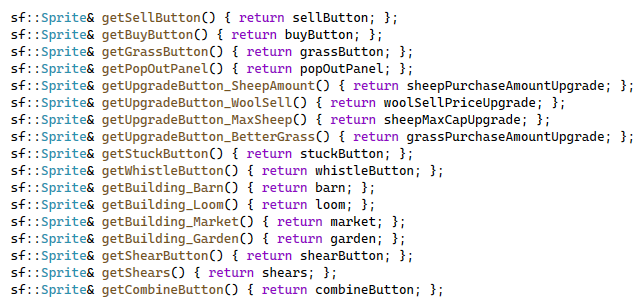
Providing access to UI buttons, mainly to Economy

Handles the Pop-out menu

-Data members-

Holds all Sprites and Textures for anything UI related

-Methods-



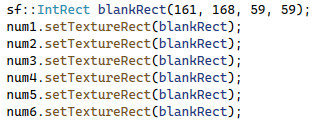
In charge of all getter funtions

**updateHUDMoney(int)** - Takes in the player’s current funds and display it in the top left.

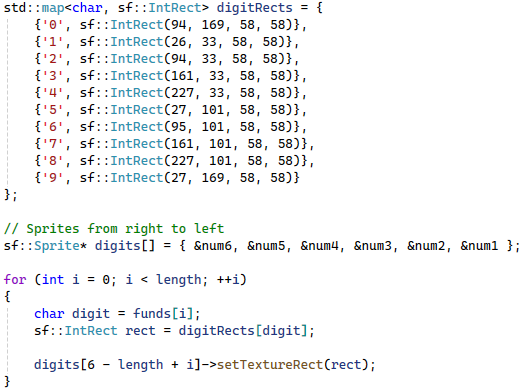
Converts the current funds to a number



Then gets the lenght of that string to use in a for loop later



Default all numbers to blanks



In the for loop here digit reads each character in funds and tells rect what to draw. So if char digit = a “5”, it’ll go up to digitRects and grab the coords mapped to the key “5”

Then “digits[6 - length + i]->setTextureRect(rect);” Tells the HUD where to place that “5” texture.

So if 42 is passed in, it’d be [6 – 2 + 0] for getting the position of “4” in “42.” This will result in the 4th slot, and because it’s read right to left it’ll end up being the second from the right

-Interactions-

**Economy** – HUD provides access to all UI components to Economy, which reads button and mouse clicks, and updates money display.

**World** – Calls Draw() to render HUD

-State Managment-

N/A

-Rendering-

Uses sf::Sprite for all buttons and icons

Money is displayed using sf::IntRects on a number texture sheet

**Particle**

-Purpose-

The particle system provides visual feedback and makes the game look a little better. It plays effects for sheep death, wolf death, shearing, and upgrades.

There are two systems involved:

* **ParticleManager:** general-purpose animated particles
* **WoolParticle:** particle class that spawns orbs. Used in wool sheering and wolf’s cash orb

-Responsibilities-

ParticleManager:

* Spawns particles after wolf, sheep and infected sheep deaths
* Updates and removes particles each frame
* Handles fade-out and motion of simple particle types

WoolParticle:

* Spawns orbs
* Orbs move towards mouse

-Data members-

Particle:

* **sf::Vector2f velocity** – Direction and speed of movement
* **float lifetime, maxLifetime** – Determines fade-out and removal timing

ParticleManager:

* **std::vector<std::unique\_ptr<Particle>> particles** – Holds an array of all active particles

WoolParticle:

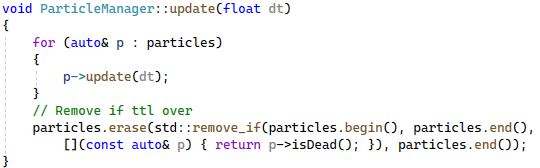
* **bool canCollect, float collectDelay** – Tied together, there’s a small delay when a particle drops as to not collect immediately and show the wool/cash orb falling away for a second
* **Sheep\* sourceSheep** – Used to calculate bonus payout based on wool stats of sheered sheep

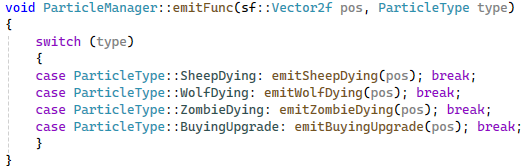
-Method-

Particle:

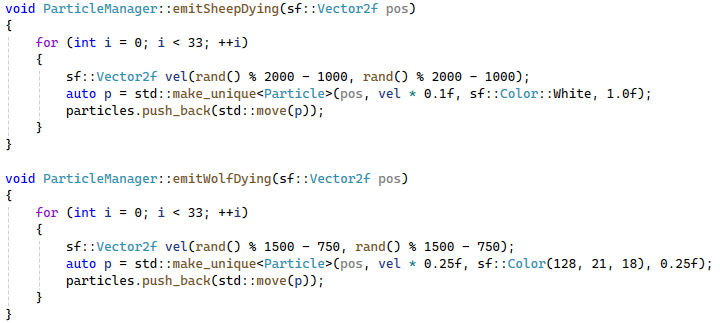
* **Particle::update(float dt)** - Decreases it’s alpha value overtime to give a ‘fading away’ effect. Handles moving with velocity.

ParticleManager:

* **update(float dt)** - Controlls the particles
* Calls each particle’s update()
* Removes particles once their lifetime ends
* 
* **emitFunc (sf::Vector2f pos, ParticleType type)**



Other classes will call this and pass in an enum type. Then depending on the enum type passed in, it will do a different effect.



The wolf will explode red, and the sheep will pop white. Each function passes in velocities to each partcle, colour and time to live.

### -Interactions-

**World** – Owns and updates both particles and wool particles

**Sheep, Wolf** – Triggers emitFunc() when dying or being sheared

**Mouse** – Directly interacts with WoolParticles to trigger pickup

**FloatingText** – Spawned alongside wool pickup to show earned value

### -State Management-

Particle manages alive and erase state with the fading function

WoolParticle tracks when it's ready to be collected using collectDelay

### -Rendering-

All particles are rendered as sf::CircleShape

Colors vary by who’s calling the emitFunc

Particles fade out

Rendered from World::Draw()