CS4303 Video Games Final Game

180013715

May 2022

1 Overview

The objective of this final practical is to design and implement my own game idea. This report describes the design of the game in detail and also contains a reflection of the game in the context of other games. Additionally, it discusses possible extensions of the game going forward.

2 Design

My game is called **Métier** (meaning an occupation in which one is good at) and it can be considered a survival game. The player poses as a crafts person who has recently started their own art studio. The bills the player must pay to sustain this lifestyle increases each week, and the player must stay afloat through crafting creations and selling them to customers.

The player can craft creations by going out and scavenging/foraging for raw material. There are three implemented locations - mines, park, and lake - each of which will contain raw materials that are specific to that location. For example, coal can be found in the mines, and straws can be found in the lake. Note that some raw materials can be found in multiple locations.

Each piece of raw material has unique properties that will play a part in the quality of the creation it aids in producing. This is described in detail later on. Customers will come into the studio on Sundays and offer to buy the crafted creations. The player can haggle (i.e. ask for a higher price) with the customers to get the right price for their creations.

The progression of the game can be measured in terms of *weeks*. From Monday to Saturday, the player will go out to scavenge for raw materials and also craft creations. Then comes Sunday, which is business day, meaning customers will arrive to make

offers on the crafted creations.

The game contains several mechanics, including scavenging, crafting, and haggling. I will attempt to describe each of them in detail.

2.1 Crafting

Figure 1: Studio

The studio is where the player crafts and sells creations. The studio is represented by a rectangle which contains slots that act as **inventory spaces** that can store raw material items and creations.

There is a **craft box** in the centre of the studio. Raw materials can be dragged into this box and the output will be a creation that has a certain quality (which determines the price of the offer the customer ends up making).

The *Studio* class instance keeps a list of available indices representing the free inventory slots and this is important in determining whether the player is allowed to pick up new

items. During scavenging, then the player picks up a raw material item, it is added to a free (randomly selected) inventory slot. When raw materials are dragged to the craft box, the spaces in which they occupied will now become free and will be re-added to the list of free slots. Similarly, when a creation is crafted, it will take up a slot in the inventory space and that index is removed from the list of free slots. Then, when Sunday comes, it will either be sold or tossed away, and the slot will become free again.

2.1.1 Dragging Items

Figure 2: Crafting

When the player is in **craft mode**, the raw material items in the studio inventory can be dragged to the craft box to make a new creation. The player can drag as many items as they want into the craft box, and when they are ready, they can press SPACE to craft a creation. The **CraftBox** class instance keeps a list of items it is currently holding, and a creation can only be crafted if this list is not empty. When a creation is crafted, this list is cleared and reset for the next group of raw material items.

A boolean variable *DRAG_STATE* is introduced to tell the game we are currently dragging an item. Initially, this may seem redundant because technically detecting a collision between the mouse and the item is enough to inform the game this item is being dragged. However, an implementation issue was encountered here: if the mouse moves too quickly, then the collision between the mouse and the item will no longer be detected, and the item will no longer move. I believe this is to do with the frame rate, meaning the item is not perfectly synchronised with the movement of the mouse. To solve this issue, *DRAG_STATE* was introduced. As long as its value is true, the item currently being dragged will alter its position based on the movement of the mouse (using mouseX, mouseY, pmouseX, and pmouseY which are variables that Processing provides). Behind the scenes, the item can be understood to be chasing the the mouse. But this is not visible to the human eye and it appears as if the dragging process is smooth. The boolean variable is set to false only when the mouse is released.

The space/slot in the inventory in which the item being dragged originally occupied is also remembered. When an item fails to be dragged to the destination craft box (i.e. mouse is let go before it reaches the box), it can return to its slot in the inventory. This is achieved through checking whether the point at which the mouse is released is on/part of the craft box. If so, then the item is added to the craft box. If not, then the item is returned to the inventory.

Note that dragging items is not enabled all the time. It is only allowed when the player is in craft mode. Game states ensure this condition and they are discussed in detail later on.

2.1.2 Quality of Creation

The raw material items used in crafting determine the quality of the final creation. There are three properties to raw material items, which all contribute to the quality of the final creation:

- **rarity** - the more rare the raw materials are, the higher the quality of the final creation
- **mood** - the more compatible the moods (e.g. descriptions) of the raw materials are, the higher the quality of the final creation
- **sturdiness** - the more sturdy the raw materials are, the longer they will last and therefore the higher the quality of the final creation

Rarity is measured out of 10, and is directly related to how often a raw material spawns at a location. The lower the value of a raw material, the more rare it is. The spawning process is described in detail later on.

To calculate the quality of a creation, the rarity property values (i.e. 10 - rarity value) of the raw materials used in crafting are first multiplied together. Then, the sturdiness property values of the raw materials are added on. Lastly, a compatibility multiplier is calculated and applied to the running result.

2.1.3 Calculating Compatibility

The compatibility of objects can be thought of as how matching the *energy* the objects give off are. In this game, I assign descriptions to each raw material item, and compatibility is calculated based on how the descriptions match up. However, the issue with natural languages is that there are many words that describe almost the same feeling but are ever so slightly different. This is accounted for through setting up three classes, each representing a group of words that give off a certain feeling:

- class A - this class accounts for playful, childlike raw materials, like marbles
  - childhood
  - playful
  - colourful
  - adventure
  - friends
- class B - this class accounts for calm, relaxing raw materials, like a blanket
  - peaceful
  - nature
  - daydream
  - comfort
  - bliss
- class C - this class accounts for gloomy, dark raw materials, like a shovel
  - frightening
  - disaster
  - cage
  - mysterious
  - journey

Note that this method of assigning the belonging class by looking at descriptions is inspired by the game *Doki Doki Literature Club*, which has a poetry writing feature. To give a brief summary of this feature, the player can impress a character of their

choosing by writing the poem with words that the character is partial to. Each word is assigned a set number of points and is associated with a certain character. At the end of the poem writing session, the character with the most points is the most impressed one.

In this game, the process works in a similar manner. The number of descriptions used for the raw material pieces belonging to each of class A, B, and C are tallied up, and compatibility is measured to be the majority class's total number of descriptions divided by the total number of descriptions. Then, the compatibility multiplier is calculated depending on how overwhelming the majority class is. For example, if all classes have the same number of descriptions, then the result for the majority class will be 1/3, which yields a multiplier of 0.5. On the other hand, if the majority class has overwhelming usage over the other two classes (e.g. result of ¿ 0.9), then the multiplier is set to 3.

In conclusion, compatibility is defined to be how *synonymous* (in reference to the three classes) the words used in describing the raw material items are.

2.2 Scavenging

The player can only scavenge once per day. This is to ensure a base level of difficulty to the game. If the player were able to go out and find as much raw material as they want, then there would no way for them to lose the game.

On each day, if the player has not gone out yet, then when they approach the door leading out of the studio they will be informed they can leave by pressing SPACE. If they have gone out that day and have returned to the studio, this message will not appear and the player will not be allowed to leave again until the next day.

The player can go to one of three locations to scavenge for raw materials:

- mines
- lake
- park

Each location is split up into rooms which the player can freely roam. Example of a room in a location:

Figure 3: Mines

2.2.1 Procedural Generating Location Layout

The location layout generation of this game is heavily inspired by the level generation in the game *The Binding of Isaac*. The key is to not make the layout too busy by bunching rooms too close together, i.e. there needs to be space between rooms.

Figure 4: Example of an Isaac Level Layout

There are secret rooms and super secret rooms in Isaac (the question marked rooms in the above image) which are generated separately from normal rooms. We will not be generating these, only normal rooms. The algorithm is simple for how well its works, and can be described as follows:

- define an array (e.g. I use a size of 400 to avoid index out of bound) and fill with zeroes initially. Note that zero represents the lack of a room, and one represents the presence of a room
- the starting room is fixed (i.e. for our array of size 400 the starting index is set to be 199), this index is added to a queue
- loop through the queue and add rooms until the generation process is complete (i.e. when the pre-determined number of rooms has been reached)

- pop an element (i.e. room) off the queue, and determine if its neighbours (in the left, right, up, and down directions) are available. A neighbouring room is defined to be available if it does *not* have more than one filled neighbour
- 50% chance of adding each available neighbour to the queue for the next iteration
- repeat until we have generated enough rooms

The source for this algorithm is linked in **Resources**. A JavaScript implementation was provided but I ended up writing my own (according to the algorithm given above) because it didn't work.

Note that due to the 50% chance, it is possible that the last element on the queue is popped, but no available neighbouring rooms are added. In this case, the code will be stuck in an infinite loop because there is nothing left on the queue to process, but the termination condition (i.e. generate a certain number of rooms) has not been reached. To handle this, we will add a random neighbouring room of the current room (given that the room is available) to the queue when there is nothing left on the queue.

In each room, there will be a variety of raw material items whose probability of spawning is governed by its rarity attribute. There will also be obstacles.

Care is taken when spawning these items and obstacles so that firstly, they are not spawned on top of one another. Secondly, they are not spawned too close to the door. This is because when the player moves to a new room, they will appear in a position that is slightly further towards the centre than the door is, so spawning an object too close to the door would mean the player might be teleported on top of that object.

2.2.2 Spawning Items

A HashMap is used to store each item's name and its rarity attribute (i.e. numerical value out of 10). An array is then filled with the item's names, and the rarity attribute determines how many copies are inserted into the array. This array is used in selecting a random item to spawn for a certain room. For example, if we had "spoon - 5" and "bandage - 7", then in the array we would have five spoons and seven bandages. So when it comes to spawning an item for the room, we can generate a random number between 0 and the length of the array, and the item at that index is the item to be spawned. Each item's probability of spawning is intrinsically normalised with this method.

A final note about this spawning process is that the number of items to spawn for this room is first determined, then for each one, a specific item is generated according to the probability distribution. The alternative is to have a probability of spawning associated with each item, and go through each one and spawn according to that probability. For

example, *spoon* could have a probability of 0.5, and **bandage** could have a probability of 0.2. In this case, when the current room goes through its spawning process, it will have a one in two chance of spawning a spoon and a one in five chance of spawning a bandage. The downside of this method is that because all the item probabilities are independent of each other, a room could end up with an overwhelming number of items. The first method fixes the total number of items to spawn, which limits the complexity of the room.

This process is implemented in *SpriteManager.pde*.

2.2.3 Spawning Obstacles

Each location will also have obstacles that are specific to that location. For mines, we have bats; For the lake, it is seaweed; For the park, it is dog poo. The player will lose money if they encounter these obstacles. The money they lose is random (within an appropriate range).

2.3 Haggle

Figure 5: Haggling

On Sundays, customers will come to the studio and make offers on pieces the player has created during the course of that week.

The mechanic is very much a game of luck. For each creation, a customer will make an initial offer based on the quality of that creation. The player can then decide if they want to accept the offer or ask for a higher price. In the latter case, the customer may get pissed off and leave, and the player will lose both the creation and the offer. The more the player asks for a higher price, the more likely the customer is to leave.

The idea of losing the creation when the player fails at haggling is not completely realistic. In real life, a piece of art can take years to find itself an owner. However, I believe implementing the haggling process in this manner raises the stakes and makes the game more fun to play. Maybe it can be understood as, the player is mad that the customer will not pay more, so in their rage, tosses the creation away.

2.4 Game Progression

To make the game more interesting, it should become harder as the weeks go on. This is achieved through a number of ways:

- bills increase in value - the player has to pay more for rent, water, and electricity at the end of each week
- the number of obstacles increase for each room with each new week while the number of raw material items remain constant
- the money that the player drops when encountering an obstacle increasing with each new week

Transition screens are important in informing the player of the current state of the game and allowing them time to process the events that have transpired.

2.4.1 Timer Class

For this, the *Timer* class is introduced. It is constructed with a time limit specifying how the time to wait for an action/event for. When the player decides they are done for the day (i.e. no more crafting to do or scavenging to do), the game state is changed to the transition screen which displays the new day (e.g. Tuesday). This screen will be displayed for a short period, as specified by the time limit of the timer, then the player returns to the studio to continue the game.

2.4.2 Message Class

This is another class that was introduced to make it easy to inform the player of what is occurring in the game. The usage of messages is tied to the actions of the player. There are three usages so far and the encapsulation of messages into its own class makes certain functionalities easy to extend.

When the player approaches the door (i.e. the player is within a certain distance of the door), a message object is constructed and it will inform the player that they can press SPACE to leave the studio (given that they have not yet left that day). This message will remain for as long as the player is near the door. If the player walks away from the door, the message will fade over time, this is achieved through decreasing the opacity of it in every draw() operation. Note that on the next iteration of the game state's draw() function, messages with an opacity of 0 (i.e. not visible) are removed. This is so the list of messages does not keep getting longer and longer.

A similar process occurs when the player approaches the craft box. The player will be asked if they want to craft using raw materials through a message object.

The final usage of this class is when the player is out scavenging. There are obstacles in each room, which will cause the player to lose money. When a collision occurs between the player and an obstacle, a message will appear informing the player of how much money they lost, this will be a random value within a certain range depending on the week number.

2.5 More on the Implementation

2.5.1 Collision Detection - in Studio

The craft box is enclosed with a circle, and collision between the player and this circle is checked to ensure the player doesn't walk through the craft box.

2.5.2 Collision Detection - in Room

The player does not need to press any keys to move to an adjacent room through a door. This is done through checking for collision between the player and a door. If collision is detected, the player is teleported to the room (different Room object) corresponding to that door.

2.5.3 Credit

The player cannot go into negative credit.

Usually, when the player encounters an obstacle while scavenging, they will lose some money. However, if the player's remaining credit is less than what is due to be lost, then the player's credit will simply go down to 0.

At the end of each week at the bill paying stage, if the player does not have enough to pay for their bills then the game is over.

2.5.4 Game States

The role of game states is crucial to the playability of the game. This game consists of many mechanics and states, and certain actions in certain states will lead to other states. Each state is characterised by its own visuals and events. To implement the various mechanics, we define a number of states/screens and the parent draw() function (in Game.pde) will access the current state and call the corresponding function which draws the relevant objects.

- pre-game state - prompts the player to press SPACE to start the game

- studio state - the relevant objects here are the player, the studio, and the messages.
- craft state - the relevant object here is the studio
- choose location state - the player is asked where they want to go to scavenge
- mines/lake/park states - the relevant objects (i.e. play area, player, location, and messages) of these three states are the same. The only difference is in the colour scheme used to represent each location and the sprites being used for the obstacles and items
- double check state - the game will ask the player if they really are finished with scavenging when the player attempts to leave the location
- haggle state - this state is entered every Sunday. An offer will be made for each creation in the player's inventory, and the player can choose to accept the offer or ask for a higher price
- transaction complete/failed states - theses are transition screens which will inform the player of the result of the haggling process. A transaction is complete when the player accepts an offer, and it fails when the player asks for a higher price but the customer leaves
- end of week state - this state is entered after every Sunday. It presents a summary of the bills for that week and the player's remaining credit
- game over state - this state is entered when the player does not have enough money remaining to pay their bills. It can only be entered after the week is over

2.5.5 Sprites

Sprites are important in conveying the location of the player and makes the game visually more impactful. The graphics for the items, obstacles (bar one), and craft box are taken from *The Binding of Isaac*. The seaweed obstacle is from `https://www.kindpng.com/imgv/ooixxi_seaweed-pixel-art-transparent-hd-png-download/.`

2.6 Design Evolution

In my pitch, I spoke about a reputation system where good creations would lead to higher offers being made in the future. This proved hard to implement, as I couldn't think of a good way of connecting the *goodness* of a creation to the player's reputation. I thought about defining the goodness of a creation as the quality of it, but the quality is linked directly to the offers that are made for it so we would end up using quality twice in the calculation of the offer price. I ended up removing this idea from the game.

3 Context

On each new day, the player can choose one of three locations to scavenge at. The layout of the chosen location is procedurally generated, similar to how levels are procedurally generated in *The Binding of Isaac*. The generation algorithm is described in the Design section.

The algorithm for calculating the quality of a creation is inspired by the poetry feature in **Doki Doki Literature Club**. This process is also described in the Design section.

Generally speaking, this game is not similar to that many games. But I should mention the game *Passpartout: Starving Artist*, which is what sparked the idea for this game. *Passpartout* also has the haggling feature where customers could come and the player can then ask for a higher price. However, I have implemented it very differently - in *Passpartout*, the art piece does not get tossed away when the customer leaves.

4 Demo

Time stamps:

- 0:02 - 0:10 basic movement showcase
- 0:11 select to go to mines to scavenge for raw material
- 0:13 - 0:32 wandering around and picking up items, notice the number of spaces left in the inventory decreases with pickups
- 0:33 game double checks when the player presses Q that they really want to leave
- 0:34 if the player says no, then they are returned to the location screen
- 0:35 if the player says yes, then they are returned to the studio
- 0:36 player is back at the studio and the raw materials they picks up in mines are now in the inventory
- 0:38 player goes near the craft box and presses SPACE to start the crafting process. The enlarged craft box signals the crafting process has started
- 0:39 - 0:43 three items are dragged to the craft box and a creation is crafted using these when the player presses SPACE
- 0:44 - 0:53 more crafting
- 0:54 player press Q to finish crafting. The craft box is back to its normal size

- 0:55 player goes towards the door but can't leave, this is because they have already gone out for the day
- 0:56 player presses Q to continue to the next day
- 0:57 transition screen describing what day it is
- 1:00 - 1:09 scavenging process again but this time at the park
- 1:10 player walks into an obstacle and drops money. Notice the remaining credit
- 1:11 - 1:25 scavenging continues. Showcasing layout of the procedurally generated location here
- 1:28 - 1:54 more crafting
- 1:54 continue to Wednesday
- 1:57 - 2:24 scavenge at the lake
- 2:27 player's remaining credit is 12, they walk into an obstacle and loses this remaining value (credit cannot go into negative)
- 2:34 - 2:36 creations cannot be dragged to the craft box
- 2:37 - 2:40 letting go of the item before it reaches the craft box returns it to its inventory position
- 3:00 - 3:08 skip to Sunday
- 3:11 - 3:15 offer is made for the first creation (notice the differently coloured inventory space), player accepts. Then taken to a transition screen which informs the player that the transaction was successful
- 3:16 player attempts to ask for a higher price for the second creation but the customer rejects this
- 3:22 player haggles successfully the first time but fails the second time when they ask for a higher price
- 3:23 - 3:40 some more haggling
- 3:44 end of week screen provides a summary of the bills that the player must pay, and how much they have remaining
- 3:49 player presses SPACE to continue to week 2

5 Evaluation

I tested the game through playing it. I would have liked to get others to test it, an unbiased pair of eyes may be able to spot bugs, contradictions, or general issues more easily. However, I did not have enough time to get people to test the game.

An issue that I considered when playing it was that a new player may struggle to tell the difference between an obstacle and a raw material item. For example, they may try to pick up a bat but be hurt by it. I personally don't think this is a big issue because when the player does encounter an obstacle, a message will pop up informing them that they have lost money, so the player will learn to not do that next time.

Given more time, I would like to investigate the formula for determining the quality of a creation based on its raw materials. In the demo playthrough, I didn't think about pairing raw material items with each other to maximise the quality of the creation. I simply dragged random items into the craft box. This works and I managed to get through the first week, so I wonder if the difficulty can be altered by adjusting the craft box formula for determining quality.

Currently, a creation cannot be used as a raw material in crafting. However, it would be interesting to extend the game to allow for this to happen. Consider layering creation on top of creation on top of creation to produce a super high quality creation. The formula for calculating quality would need to be altered as it currently only considers pre-defined properties sturdiness, rarity, and mood. The stakes are also raised when it comes to haggling a layered creation because the player would not want it to go to waste.

A bug was encountered in generating the inventory layout of the studio. The inventory spaces are generated starting with the top and bottom rows, then the side columns. The issue came from the fact that the corners would have duplicate spaces because the indexing was slightly off. This became an issue later on when the player goes out to scavenge for raw material items. Items are added to a random spot in the available indexes, but some of these available indexes point to the same inventory space (in the case of the four corners). This means, some items will end up in the same slot.

In its current state, the game is quite relaxing to play, as the player simply wanders about and presses some keys occasionally. With that being said, the game can be easily extend to include moving obstacles. This would make sense for bats in particular, as they could fly towards the player to attack them. In the case of seaweeds, I think a melee attack would work. Keeping the properties and attacks of the obstacles different also make the game more interesting. The player can decide what locations are the easiest for them and prioritise those when they are low on credit.

With obstacles/enemies being allowed to move and attack, the game should also be extended to allow the player to protect themselves. I think to start off with, a palette knife (melee attack) and a paint brush (long range attack) would be suitable. As the player becomes more successful as an artist, maybe they could have more tools to protect themselves with.

6 Resources

https://bindingofisaacrebirth.fandom.com/wiki/Collection_Page_(Afterbirth_%E2%80%A0) http://www.jeffreythompson.org/collision-detection/line-rect.php http://www.jeffreythompson.org/collision-detection/rect-rect.php https://processing.org/reference/ https://www.boristhebrave.com/2020/09/12/dungeon-generatio https://www2.cs.sfu.ca/CourseCentral/166/tjd/drag_and_drop.html https://primagames.com/tips/doki-doki-literature-club-plus-poem-words-guide-each-characters-f https://riseupgamer.com/ultimate-beginner-guide-the-binding-of-isaac-repentance/